Stochastic Dual Dynamic Programming

ANTHONY PAPAVASILIOU

Contents

Preface

The intended audience of these notes are students and researchers who are interested in learning about or conducting research in optimization under uncertainty. Readers are assumed to have an elementary background in operations research. The learning of the material relies on solving problems using a mathematical programming language. Although traditional mathematical programming languages such as AMPL or GAMS could be used for the task, experience suggests that the use of a broader-purpose scientific language (e.g. Matlab) that accommodates solver libraries (e.g. CVX or Yalmip) is more appropriate for the task, since the algorithms developed in these notes blend simulation with optimization and therefore require software that can accomplish both.

The algorithms covered in the notes are particularly applicable to a specific class of optimization problems under uncertainty, referred to as stochastic programming problems. Some of the algorithms that are presented in the notes are applicable to more general classes of large-scale optimization problems. The notes use electricity markets as a domain of application that motivates these algorithms, however the reader will easily recognize that the concepts and methods are transferable to broader applications. The notes are developed in order to support a single-semester graduate level course on stochastic programming.

These notes provide an introduction to the stochastic dual dynamic programming algorithm (SDDP). The SDDP algorithm is used for solving a specific class of optimization problems under uncertainty, namely multistage stochastic linear programs. Despite its special purpose, the study of the SDDP algorithm provides insight into the design of a broad class of decomposition algorithms, as discussed in the opening section of chapter 1. Chapter 1 further introduces optimization under uncertainty by presenting a classical application, the optimal planning of production capacity under uncertain demand.

Chapter 2 reviews mathematical concepts that are used repeatedly in the course notes. Linear programming and basic definitions in probability are summarized. Duality is covered as stand-alone material as it is particularly useful for developing decomposition algorithms. Subgradients are also covered as stand-alone material as they are used extensively in the developed algorithms.

Chapter 3 reviews the dynamic programming algorithm. The dynamic programming algorithm can be used for solving an extremely wide family of problems. A specific class of problems that can be tackled through dynamic program-

ming are sequential decision making problems under uncertainty. The remaining notes focus on yet a more restricted family of decision making problems under uncertainty, namely stochastic linear programs.

Chapter 4 introduces stochastic linear programs, which are linear optimization problems with random parameter inputs. Two-stage stochastic linear programs are reviewed first, followed by the more general class of multi-stage stochastic linear programs.

Chapter 5 introduces a family of cutting plane methods which can be used for solving two-stage stochastic linear programs. The presentation of Benders decomposition, originally developed as an approach for solving mixed integer linear programs, is followed by the L-shaped method, which is specifically tailored to problems with an L-shaped constraint coefficient matrix, such as two-stage stochastic linear programs. The chapter concludes with the multi-cut L-shaped method.

The algorithms presented in chapter 5 are extended in chapter 6 in order to tackle multi-stage stochastic linear programs. The focus is on nested decomposition, which is an exhaustive version of the L-shaped method for multiple stages. Although non-scalable in practice, this algorithm provides the basic ingredients that are required for the implementation of the SDDP algorithm.

Chapter 7 presents the culmination of all the preceding theory into the SDDP algorithm. The SDDP algorithm is a simulation-based variant of the nested decomposition algorithm, which can be used as a scalable approach for tackling multi-stage stochastic linear programs..

The implementation of the algorithms presented in these notes requires a substantial investment of time and effort. Depending on the application at hand, such an overhead may not be justified because simpler heuristics may perform adequately. This is often the case when there is not nough uncertainty' in the problem to warrant a stochastic programming approach. This is linked to the so-called value of the stochastic solution. One way to limit the uncertainty faced by the decision maker is by acquiring more accurate forecasts, which motivates the study of the expected value of perfect information. The estimation of these metrics is discussed in chapter 8.

1 Introduction

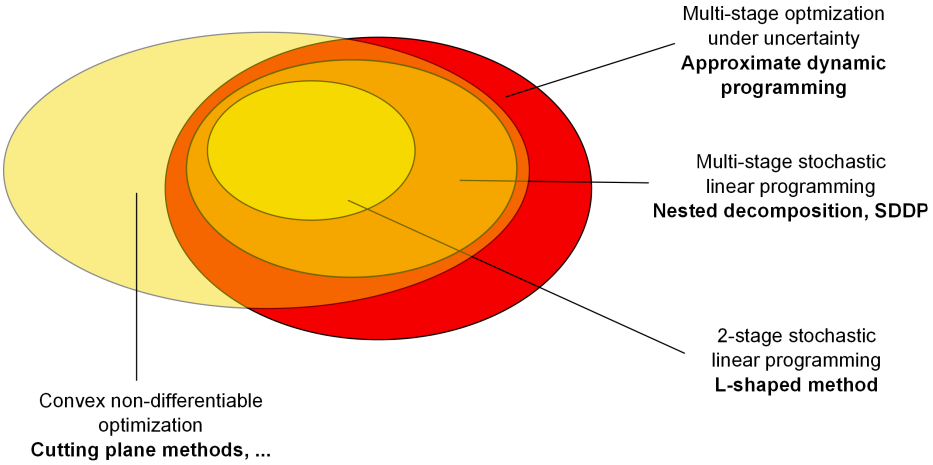
While conducting a literature review in preparation for my qualifying exams at Berkeley in the fall of 2009, I stumbled upon a publication by Pereira and Pinto, *Multi-Stage Stochastic Optimization Applied to Energy Planning*. The topic seemed highly relevant to the problem that would be addressed in my PhD thesis, and as I started reading through the publication I was so deeply immersed into the content that I proceeded to the ultimate sacrifice: staying home on a Saturday night in order to read a journal paper. The algorithm that is developed in the paper, Stochastic Dual Dynamic Programming (SDDP), has found great success in medium-term energy planning. The goal of these notes is to provide a gentle introduction to the algorithm, while also developing in further detail some of the mathematical methods that the algorithm relies on. These methods are generally applicable in areas beyond stochastic programming.

1.1 Why Study SDDP?

The SDDP algorithm is aimed at resolving multi-stage stochastic linear programs. The SDDP algorithm sits on the intersection of two very interesting methods. On the one hand, SDDP belongs to a broader family of algorithms that blend simulation and optimization, titled approximate dynamic programming algorithms. The underlying idea of these methods is to overcome the curse of dimensionality of solving optimization problems under uncertainty by resorting to simulation. On the other hand, SDDP relies on additional structure by virtue of the fact that at each stage a linear program is being solved, and is as such a specific instance of cutting plane methods. The study of the SDDP algorithm is therefore a first step towards being introduced to a wide family of methods that can be applied for tackling optimization problems under uncertainty (often tackled by approximate dynamic programming), as well as large-scale non-differentiable convex optimization (often tackled by cutting plane methods).

Figure 1.1 provides a roadmap of the optimization problems and algorithms that are studied in these notes. The SDDP algorithm is geared towards resolving multi-stage stochastic linear programs. These are sequential decision-making problems under uncertainty, which correspond to a specific class of multi-stage decision making problems under uncertainty. Multi-stage decision making prob-

Figure 1.1 The classes of problems and corresponding algorithms that are developed in these notes. SDDP sits at the intersection of two broad solution methods, approximate dynamic programming and cutting plane methods.

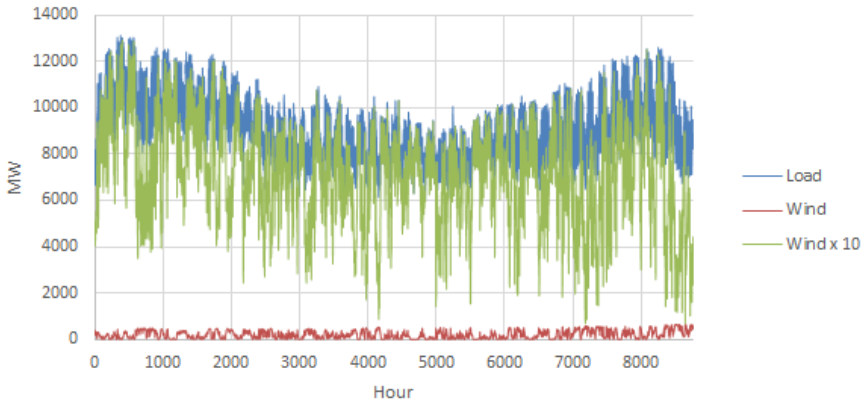


lems under uncertainty can generally be attacked by approximate dynamic programming algorithms, of which SDDP is a specific instance. These notes build up the theory towards SDDP by first focusing on two-stage stochastic linear programs, which are solved by the L-shaped method. The L-shaped method can be used as a building block for the nested decomposition algorithm, which is an exhaustive approach towards solving multi-stage stochastic linear programs. The SDDP algorithm is a scalable adaptation of the nested decomposition algorithm which is based on Monte Carlo simulation. The L-shaped method, which is the building block of SDDP, is a specific instance of cutting plane algorithms which can be used for solving convex non-differentiable optimization problems.

The study of the SDDP algorithm develops familiarity with the notion of decomposition in mathematical programming. Decomposition is the idea of breaking a problem into smaller pieces in order to accelerate its (approximate) solution. Decomposition is becoming increasingly relevant at present due to the advent of parallel computing.

Apart from its theoretical beauty, the SDDP algorithm has found great commercial success. The algorithm is the principal tool used for the purpose of scheduling hydrothermal systems, namely planning the level of water in hydro reservoirs in order to meet the annual demand of a system at minimum expected cost. Numerous applications could be cast as multi-stage stochastic linear programs and tackled by SDDP, examples will be provided later in the notes.

Figure 1.2 The chronological evolution of load and wind power in Belgium in 2013.



1.2 An Example: Capacity Expansion

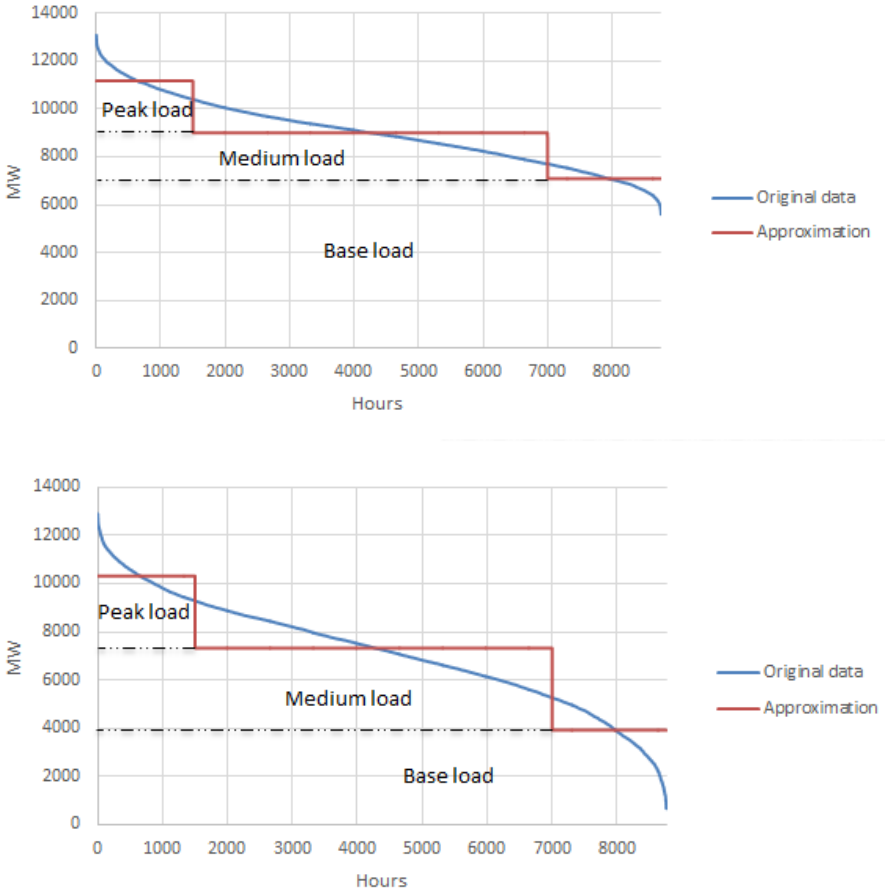
The capacity expansion planning model is a typical problem of multi-stage optimization under uncertainty. The goal of the problem is to plan the roll-out of production capacity over multiple time stages, in order to minimize expected investment and operating costs while ensuring that uncertain future demand is satisfied.

To put the problem into context, consider the chronological evolution of system load and wind power production in Belgium in 2013, shown in figure 1.2. The net load of the country is obtained by subtracting total load from wind power production. Sorting the time series of net load in descending order yields the *load duration curve* of the system, which is shown in figure 1.3. This curve describes the number of hours in the year that load was greater than or equal to a given level. For example, net load was at or above 10000 MW for 2000 hours during the year.

The same figure presents a step-wise approximation of the load duration curve. Using the load duration curve, load can be stratified horizontally, rather than chronologically. Base load corresponds to the lower horizontal block in the figure and amounts to 7086 MW. This corresponds to loads that last for the entire year. The medium horizontal block corresponds to loads between 7086 and 9004 MW. These loads last for 7000 hours. Peak load corresponds to the upper horizontal block and ranges between 9004 MW and 11169 MW. These loads last for 1500 hours.

The set of technologies that can be used for serving demand is shown in table 1.1. Fuel cost corresponds to the variable cost of each plant and is proportional to the amount of energy produced. Investment cost is independent of output and proportional to the total capacity built. The investment cost has been discounted

Figure 1.3 Upper panel: the load duration curve of Belgium in 2013, and a step-wise approximation. Lower panel: the load duration curve if wind production were 10 times greater.



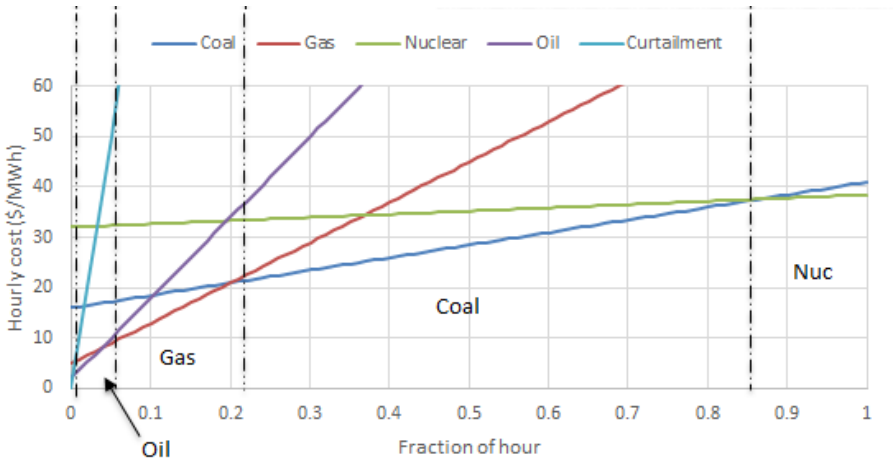
in the table in order to reflect the *hourly* cash flow that is required in order to support an investment in 1 MW of the given technology. Demand response refers to the option of not serving demand. This can be viewed as a 'technology' with a zero investment cost (since no capacity is actually built) but a very high variable cost.

The optimal investment problem can be posed as follows: find the mix of technologies that should be built in order to minimize the total fixed and variable cost of serving demand. A graphical solution to the problem can be derived from the *screening curve* of figure 1.4. The total cost of using 1 MW of each technology depends on the amount of time that the technology is used. In order to determine

Table 1.1 The set of options for serving demand. DR corresponds to demand response (i.e. not serving demand).

Technology	Fuel cost ($/MWh)	Inv cost ($/MWh)
Coal	25	16
Gas	80	5
Nuclear	6.5	32
Oil	160	2
DR	1000	0

Figure 1.4 A screening curve describes the total hourly cost of each technology as a function of the fraction of time that it is producing.



optimal investment, it is therefore useful to view load as a collection of horizontal slices. The only difference among slices is their duration, and this will determine the technology that should be used for serving each slice. Base load should be served by a technology with low variable cost (even if fixed cost may be relatively high), whereas peak load should be served by a technology with low fixed cost (even if variable cost may be relatively high).

The exact fraction of time for which each technology should be functioning can be found from the lower envelope of the total cost function:

- For DR: $1000 \cdot f \leq 2 + 160 \cdot f \Leftrightarrow f \leq 0.0024$. This corresponds to horizontal load slices that last 0-21 hours.
- For oil: $f > 0.0024$ and $2 + 160 \cdot f \leq 5 + 80 \cdot f \Leftrightarrow f \leq 0.0375$. This corresponds to horizontal load slices that last 21-328 hours.
- For gas: $f > 0.0375$ and $5 + 80 \cdot f \leq 16 + 25 \cdot f \Leftrightarrow f \leq 0.2$. This corresponds to horizontal load slices that last 328-1752 hours.

Table 1.2 Two possible realizations of the load duration curve. The reference load duration curve materializes with a probability of 10%, while the 10x wind scenario materializes with a probability of 90%.

	Duration (hours)	Level (MW) Reference scenario	Level (MW) 10x wind scenario
Base load	8760	0-7086	0-3919
Medium load	7000	7086-9004	3919-7329
Peak load	1500	9004-11169	7329-10315

- For coal: $f > 0.2$ and $16 + 25 \cdot f \leq 32 + 6.5 \cdot f \Leftrightarrow f \leq 0.8649$. This corresponds to horizontal load slices that last 1752-7576 hours.
- For nuclear: $0.8649 \leq f \leq 1$. This corresponds to horizontal load slices that last 7576-8760 hours.

It follows from the load duration curve of the system that the least cost capacity investment plan is the following:

- Base load is assigned to nuclear: 7086 MW
- Medium load is assigned to coal: 1918 MW
- Peak load is assigned to gas: 2165 MW
- No load is assigned to oil: 0 MW
- No load is assigned to DR: 0 MW

This investment decision corresponds to the step-wise approximation of the load duration curve. If the actual load duration curve had been used instead, there would have also been an investment in peaking gas units.

Suppose, now, that the actual wind production in the system is 10 times greater than the production assumed previously, as shown in figure 1.2. The load duration curve of the system is shown in the lower panel of figure 1.3. It can be noted that the load duration curve has now become less flat, and there are hours when the net load of the system is near-zero. Since the load duration curve decreases more steeply, the required amount of coal and gas capacity also increases. The step-wise approximation of the two load duration curves under varying levels of wind penetration are summarized in table 1.2.

In order to introduce uncertainty into the problem, consider a situation where, due to unpredictable weather, wind parks produce the amount of wind that corresponds to the 10x scenario in figure 1.2 every 90% of the days in the year, and the amount of wind power corresponding to the reference scenario of figure 1.2 every 10% of the days in the year. What would be an *optimal* expansion plan in this case? *Optimal* refers here to the expansion plan that minimizes the *expected* total cost.

There are two natural approaches towards this problem. The simplest approach would be to compute the expected demand of each hour, and invest in order to minimize the cost of satisfying this expected demand. This formulation is referred

Figure 1.5 The true load duration curve, versus the expected load duration curve, in the case where the 10x wind scenario materializes for 90% of the hours of the year.

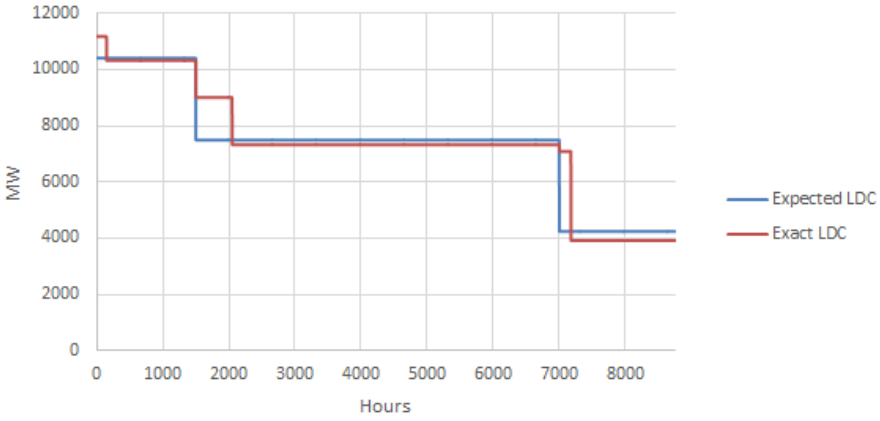


Table 1.3 Optimal assignment of capacity for the 6-block load duration curve.

	Duration (hours)	Level (MW)	Technology
Block 1	8760	0-3919	Nuclear
Block 2	7176	3919-7086	Coal
Block 3	7000	7086-7329	Coal
Block 4	2050	7329-9004	Coal
Block 5	1500	9004-10315	Gas
Block 6	150	10315-11169	Oil

to as the *expected (or mean) value problem*. The average load duration curve is shown in figure 1.5.

A more careful approach would recognize that, in the presence of wind production uncertainty, one faces a composite load duration curve which mixes the two original load duration curves of figure 1.3. Since the reference curve occurs with a frequency of 10%, it can be compressed along the time axis to get a new load duration curve, which has a length of 10% of the original curve. Similarly, the load duration curve of the 10x scenario is compressed along the time axis to 90% of its original length. The two load duration curves are then merged to the 6-block curve of figure 1.5. This approach results in a formulation of the problem as a *stochastic program*.

A graphical solution to the stochastic program and the deterministic equivalent can again be derived based on screening curves. The optimal assignment of capacity is shown in tables 1.3 and 1.4.

The total investment in capacity and the corresponding investment cost is shown in table 1.5. Note how the investment plans deviate. The stochastic pro-

Table 1.4 Optimal assignment of capacity for the expected load duration curve.

	Duration (hours)	Level (MW)	Technology
Base load	8760	0-4235	Nuclear
Medium load	7000	4235-7496	Coal
Peak load	1500	7496-10401	Gas

Table 1.5 Optimal investment and fixed cost for the stochastic program and the expected value problem.

	SP investment (MW)	EV investment (MW)	SP fixed cost ($/h)	EV fixed cost ($/h)
Coal	5085	3261	81360	52176
Gas	1311	2905	6,555	14525
Nuclear	3919	4235	125408	135520
Oil	854	0	1708	0
Total	11169	10401	215031	202221

gramming solution invests in oil in order to ensure that block 6 can be served. Instead, the expected value model does not 'see' block 6 and therefore does not build capacity in order to satisfy it. The nuclear capacity in both models is similar. On the other hand, the stochastic programming model invests less in gas and relies more on coal. This is mostly driven by block 4 which lasts for 2050 hours and is therefore best served by coal rather than gas units. Instead, the expected value model does not 'see' block 4 and aggregates it to the medium load block, which is substantially lower than the level of block 4.

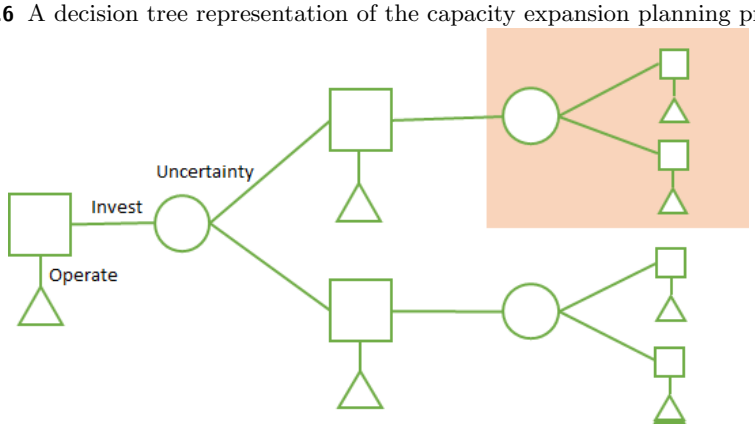
It can be observed that the total investment cost of the EV (expected value) model is lower than the SP (stochastic programming) model. However, the SP plan performs better than the EV in terms of variable costs. The variable costs of the two plans can be computed by using the *merit order dispatch rule*. The merit order dispatch rule moves through units in order of increasing variable cost, and assigns them to load blocks of decreasing duration, until either all load blocks are satisfied or all generating capacity is exhausted. The proof of optimality of the merit order dispatch rule is left as an exercise. Following merit order dispatch, the variable costs are obtained in table 1.6. Although EV serves block 2 at lower cost, block 4 costs much more due to the fact that it is served largely by gas instead of coal and block 6 costs much more due to demand curtailment. The total cost of EV amounts to 353055 $/h. The total cost of SP amounts to 340316 $/h. The difference, referred to as the *value of stochastic solution*, amounts to 12739 $/h.

It is worth considering how the problem could be tackled in a multi-stage setting. The relative timing and information in a multi-stage setting is described in figure 1.6. Squares represent decisions, circles represent the realization of uncer-

Table 1.6 Variable cost for the SP and EV models.

	SP var cost ($/h)	EV var cost ($/h)
Block 1	25473	25473
Block 2	64858	60070
Block 3	4854	4854
Block 4	9799	29209
Block 5	17960	17959
Block 6	2340	13268
Total	125285	150834

Figure 1.6 A decision tree representation of the capacity expansion planning problem.



tainty and triangles represent payoff. The orange shaded area is a sub-structure that repeats itself as one move backwards in the diagram. This recursive form of multi-stage optimization under uncertainty makes the problem amenable to solution by *dynamic programming*. Note that decisions are categorized between investment and operations decisions. Investment decisions influence the future state of the system, whereas operations decisions are pointing downwards since they have no influence on the future state of the system. The lower triangles may in fact contain complex optimization problems that can be treated in isolation. The structure of this problem, whereby decisions can be classified according to whether or not they influence the future state of the system, is called *block separability* and can be exploited computationally.

The mathematical programming formulation of the two-stage capacity expansion problem can lead to an interesting insight about the application of dynamic programming for solving the problem in multiple stages. Technology i is characterized by its investment cost I_i and its variable cost C_i. Load block j is characterized by its height D_j and its duration T_j. The capacity expansion planning

problem can then be formulated as follows:

$$\min_{x,y} \sum_{i=1}^{n} (I_i \cdot x_i + \sum_{j=1}^{m} C_i \cdot T_j \cdot y_{ij}) \tag{1.1}$$

$$\text{s.t.} \sum_{i=1}^{n} y_{ij} = D_j, j = 1, \ldots, m$$

$$\sum_{j=1}^{m} y_{ij} \leq x_i, i = 1, \ldots n-1 \tag{1.2}$$

$$x, y \geq 0$$

Note that a variable y_{ij} has been defined in order to describe the amount of capacity of technology i that is allocated for serving block j. The objective function (1.1) includes fixed and variable costs, where the possibility of not serving load is included as one of the technology options (load shedding is option n, which is why constraint (1.2) is only enforced up to technology option $n-1$). Uncertainty regarding the load duration curve is captured by using 6 blocks instead of 3.

Consider, now, how the optimal cost varies as a function of the investment decision x. It can be observed that the dual optimal multipliers of the linear program $\min_{y \geq 0}\{\sum_{i=1}^{n}(I_i \cdot x_i + \sum_{j=1}^{m} C_i \cdot T_j \cdot y_{ij}) : \sum_{i=1}^{n} y_{ij} = D_j, \sum_{j=1}^{m} y_{ij} \leq x_i\}$, when x has been fixed, depend on x. It will be shown in the sequel that the optimal value of this linear program is a convex function of x. Since there is a finite set of dual optimal multipliers, it can be concluded that the optimal value is a piecewise linear function of x. This structure can be exploited in the design of a dynamic programming algorithm for solving the problem.

Problems

1.1 For the investment section, run the SP and EV models in a math programming language and confirm the results that we obtained by hand. Is it equivalent to express the duration of each block in hours (per year), or percent of the year? If not, then which of the two is the right choice?

1.2 Prove the optimality of the merit order dispatch rule.

1.3 Consider the **newsvendor** or **newsboy problem**, which is a classical problem in operations research. A newsboy needs to buy newspapers at a cost c in order to serve an uncertain demand d, which is randomly distributed according to a continuous distribution function f. The newsboy can only sell up to the amount of news papers that are bought, and excess newspapers can be sold back to the newspaper distributor at a salvage value s. The goal is to determine the optimal quantity of newspapers that need to be sold.

- Write the newsboy problem as an optimization problem. What is the first-stage decision? What is the second-stage decision?
- What is the optimal quantity of newspapers that need to be purchased?

Bibliography

Section 1.2. The stochastic capacity expansion planning problem was originally studied by (Louveaux & Smeers 1988). Block separability is exploited in a parallel optimization scheme for capacity expansion planning by (Dantzig & Glynn 1990).

2 Mathematical Background

This chapter covers background mathematical material that is used repeatedly in the notes. Section 2.1 presents background material in probability and convex analysis in summary form. Linear programming is presented in more detail in section 2.2. Although the material in linear programming is not developed in self-contained form, certain aspects of linear programming which are important for the subsequent development are developed in some amount of detail. Duality is introduced in section 2.3. This material is essential for discussing Benders decomposition and its derivative cutting plane methods. Subgradients are introduced in section 2.4, and are needed for developing cutting plane methods.

2.1 Short Reviews

This section introduces definitions that the reader should already be familiar with. Section 2.1.1 covers introductory notions from probability. The only non-standard information in this section is how sample spaces are defined for discrete-time multi-stage decision-making problems under uncertainty with a finite sample space. This material is based on (Puterman 2014). Convex analysis is covered in section 2.1.2.

2.1.1 Probability and Random Variables

This section presents certain definitions of probability theory that are required in the sequel. Since these definitions are fairly abstract, they are illustrated in the context of a scenario tree (a discrete time model of a discrete random vector), so as to illustrate the notions in a tangible fashion.

Probability spaces are the building blocks of optimization models under uncertainty. The definition of probability spaces requires defining a sigma-algebra.

DEFINITION 2.1 Given a **sample space** Ω, a **sigma-algebra** $\mathcal{A}$ is a set of subsets of Ω such that

- $\Omega \in \mathcal{A}$
- if $A \in \mathcal{A}$ then also $\Omega - A \in \mathcal{A}$
- if $A_i \in \mathcal{A}$ for $i = 1, 2, \ldots$ then also $\cup_{i=1}^{\infty} A_i \in \mathcal{A}$

In general, given a certain set Ω, there is not a unique sigma-algebra of Ω. For example, $\{\emptyset, \Omega\}$ and the set of all subsets of Ω (which is defined as the **power set** of Ω and denoted 2^{Ω}) are both sigma-algebras of Ω. The sets Ω that are discussed in these notes are typically finite, and the power set of Ω will be denoted as $\mathcal{B}(\Omega)$.

A sigma-algebra encodes information. Elements of the sigma-algebra are referred to as **events**, and they can be distinguished by decision makers, which is why they should be understood as the amount of information that a decision maker can access.

The problems of multi-stage decision making under uncertainty that will be studied in these notes are formally defined in chapter 3. For the sake of introducing concepts, it is mentioned here that these problems take place over a sequence of finite time periods $t = 0, \ldots, H$, where the state of the world at each time period is defined through a set of states, denoted here as S_t, and a set of actions, denoted here as A_t. The way that the gradual revelation of information is modeled is by defining, for each stage t, a sigma-algebra which is a subset of the power set of $\Omega = (S_0, A_0) \times \ldots \times (S_{H-1}, A_{H-1}) \times S_H$.

In multi-stage stochastic linear programs, which are the main focus of this text, the specification of uncertainty is more specific. Typically in the problems studied in the text, there will be a finite set S_t at each stage. The sample space is then defined as $\Omega = S_0 \times \ldots \times S_H$. Note that the action set is not involved in the definition of Ω, in contrast to the more general definition in the previous paragraph.

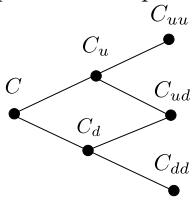
Example 2.1 Consider the evolution of a stock price, which will be encountered in more detail in section 3.4.4. A stock price evolves according to the *lattice* shown in figure 2.1. The state space of the lattice at each stage can be defined as the set of values that the stock price can take at each stage. Thus, one obtains $S_0 = \{C\}$, $S_1 = \{C_u, C_d\}$, and $S_2 = \{C_{uu}, C_{ud}, C_{dd}\}$. It is possible to define three sigma algebras using this lattice. First, one needs to define the sample space. Following the previous discussion, the sample space is defined as

$$\Omega = S_0 \times S_1 \times S_2 = \{(C, C_u, C_{uu}), (C, C_u, C_{ud}), (C, C_u, C_{dd}), (C, C_d, C_{uu}), (C, C_d, C_{ud}), (C, C_d, C_{dd})\}.$$

From this sample space, one obtains the set of all subsets of Ω, which can be written as follows:

$$\mathcal{B}(\Omega) = \{\emptyset, \{(C, C_u, C_{uu})\}, \ldots, \{(C, C_u, C_{uu}), (C, C_u, C_{ud})\}, \ldots,$$
$$\{(C, C_u, C_{uu}), (C, C_u, C_{ud}), (C, C_d, C_{ud})\}, \ldots,$$
$$\{(C, C_u, C_{uu}), (C, C_u, C_{ud}), (C, C_u, C_{dd}), (C, C_d, C_u u)\}, \ldots,$$
$$\ldots$$
$$\{(C, C_u, C_{uu}), (C, C_u, C_{ud}), (C, C_u, C_{dd}), (C, C_d, C_{uu}), (C, C_d, C_{ud}), (C, C_d, C_{dd})\}\}$$

Figure 2.1 The evolution of stock price in example 2.1



where $\{(C, C_u, C_{uu})\}$ is a single-element subset of Ω, $\{(C, C_u, C_{uu}), (C, C_u, C_{ud})\}$ is a two-element subset of Ω, $\{(C, C_u, C_{uu}), (C, C_u, C_{ud}), (C, C_d, C_{ud})\}$ is a three-element subset of Ω, ..., going up to a single six-element subset of Ω. Note that any event in $\mathcal{B}(\Omega)$ can be identified as having occurred or not on the basis of the information that is available up to stage 2. For example, the event 'the stock price in period 2 is C_{ud}' corresponds to the following element of $\mathcal{B}(\Omega)$: $\{(C, C_u, C_{ud}), (C, C_d, C_{ud})\}$. In period 0, the following sigma algebra can be defined:

$$\mathcal{A}_0 = \{\emptyset, \Omega\}.$$

This encodes the fact that the only non-empty event that the decision maker can distinguish in stage 0 is that the stock price is C. Note that this is a valid sigma-algebra on Ω, because it satisfies all three conditions of the definition of a sigma-algebra.
In period 1, the following sigma algebra can be defined:

$$\begin{aligned}\mathcal{A}_1 = \{&\emptyset,\\ &\{(C, C_u, C_{uu}), (C, C_u, C_{ud}), (C, C_u, C_{dd})\},\\ &\{(C, C_d, C_{uu}), (C, C_d, C_{ud}), (C, C_d, C_{dd})\},\\ &\Omega\}.\end{aligned}$$

This encodes the set of events that the decision maker can distinguish based on observations up to period 1. For example, the second element in the set is 'the stock price in period 0 is C, and in period 1 it is C_u'. This is an event that the decision maker can identify as having occurred or not, on the basis of observations up to that moment in time. By contrast, the event 'the stock price in period 0 is C, in period 1 it is C_u, and in period 2 it is C_{uu}' is *not* an event that the decision maker can identify on the basis of information in period 1, because the price in period 2 may or may not be C_{uu}. The reader should again check that $\mathcal{A}_1$ is indeed a sigma algebra on Ω by verifying that the properties of the definition of a sigma algebra are satisfied.

DEFINITION 2.2 A **measurable probability space** is the triplet $(\Omega, \mathcal{A}, \mathbb{P})$,

where Ω is the sample space, $\mathcal{A}$ is a sigma-algebra of Ω, and $\mathbb{P} : \mathcal{A} \to [0, 1]$ is the probability measure that obeys the following properties:

- $\mathbb{P}(\emptyset) = 0$,
- $\mathbb{P}(\Omega) = 1$, and
- $\mathbb{P}(\cup_{i=1}^{\infty} A_i) = \sum_i \mathbb{P}(A_i)$ if A_i are disjoint.

In what follows, the notation $\mathbb{P}[\cdot]$ will often be interchanged with the notation $\mathbb{P}(\cdot)$.

The sample space of a decision-making problem under uncertainty may not be fully revealed to the decision maker. What the decision maker can observe, and react to, are *random variables* and *random vectors*. Since Ω will be assumed to be a countable (in fact, finite) set throughout this text, random variables can be defined as follows. A **random variable** is a function $\xi : \Omega \to \mathbb{R}$. A **random vector** is a function $\xi : \Omega \to \mathbb{R}^n$.

Example 2.2 In the stock pricing example, one can introduce three random variables: the price ξ_0 at stage 0, the price ξ_1 at stage 1, and the price ξ_2 at stage 2. All these random variables are defined on Ω, but are different mappings from Ω to $\mathbb{R}$ (assuming that C, C_u, C_d, C_{uu}, C_{ud}, C_{dd} are real numbers). For period 0,

$$\xi_0(\omega) = C, \omega \in \Omega$$

For period 1,

$$\xi_1(\omega) = C_u, \omega = (C, C_u, \cdot)$$
$$\xi_1(\omega) = C_d, \omega = (C, C_d, \cdot)$$

Finally, for period 2,

$$\xi_2(\omega) = C_{uu}, \omega = (C, \cdot, C_{uu})$$
$$\xi_2(\omega) = C_{ud}, \omega = (C, \cdot, C_{ud})$$
$$\xi_2(\omega) = C_{ud}, \omega = (C, \cdot, C_{ud})$$

Having defined random variables, it is possible to define their distribution function, and subsequently their expectation and variance. The **cumulative distribution function** of a random variable ξ is defined as $F_\xi(x) = \mathbb{P}[\xi \leq x]$. For discrete random variables, the **probability mass function** p is defined as $p(\xi_\omega) = \mathbb{P}[\xi = \xi_\omega], \omega \in \Omega$ with $\sum_{\omega \in \Omega} p(\xi_\omega) = 1$. The **expectation** of a random variable is defined as $\mu = \sum_{k \in K} \xi^k f(\xi^k)$ for discrete random variables. The **variance** of a random variable is defined as $\mathbb{E}[(\xi - \mu)^2]$. The **support** of a

random variable is the set of values that the random variable can attain with a non-zero probability.

The gradual revelation of inormation in multistage decision-making under uncertainty implies that information is updated at every stage. To represent this, it is useful to introduce filtrations and adapted stochastic processes. Adapted stochastic processes are interpreted as **non-anticipative**, meaning that they cannot 'see into the future'. In other words, adapted processes are consistent with the information that is available at every stage of the decision-making problem. For discrete-time problems, which are the focus of these notes, the following definition of a **filtration** can be provided.

Given $(\Omega, \mathcal{B}(\Omega))$, a **filtration** is an increasing sequence of sigma-algebras $\{\mathcal{A}_t\}_{t\in\mathbb{Z}}$ where

$$t_1 \leq t_2 \Rightarrow \mathcal{A}_{t_1} \subseteq A_{t_2}.$$

Example 2.3 Continuing on example 2.1, the sequence $(\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2)$, where $\mathcal{A}_0$ and $\mathcal{A}_1$ are defined in example 2.1, and $\mathcal{A}_2 = \mathcal{B}(\Omega)$, defines a filtration on $(\Omega, \mathcal{B}(\Omega))$.

The observable sequence of random vectors that a decision-maker reacts to is a *stochastic process*. Given an index set $T \subseteq \mathbb{N}$, and a measurable space $(\Omega, \mathcal{A})$, a **stochastic process** is a collection of $\mathbb{R}^n$-valued random vectors, which can be written as $(X_t, t \in T)$, such that $X_t : \Omega \to \mathbb{R}^n$.

Assuming a discrete sample space, which is the focus of these notes, an adapted process can be defined. Let $(\Omega, \mathcal{A})$ be a measurable space with a countable sample space Ω. Denote $\{\mathcal{A}_t\}_{t\in T\subseteq\mathbb{N}}$ as a filtration of the sigma algebra. Consider a process $X : \Omega \to S^T$, where $S \subset \mathbb{R}^n$ is countable. Then X is **adapted** to $\{\mathcal{A}_t\}_{t\in\mathbb{N}}$ if $X_t^{-1}(c) = \{\omega : X_t(\omega) = c\} \subseteq \mathcal{A}_t$ for all $c \in \mathbb{R}^n$.

A process is therefore adapted to a filtration if it is impossible to gain additional information from the observation of the process in time t beyond the information which is contained in the filtration $\{\mathcal{A}_t\}_{t\in T}$.

Example 2.4 Consider the stochastic process ξ_t which is defined in example 2.2. The process is adapted with respect to the filtration defined in example 2.3. For instance,

$$\xi_1^{-1}(C_u) = \{(C, C_u, C_{uu}), (C, C_u, C_{ud}), (C, C_u, C_{dd})\} \subseteq \mathcal{A}_1,$$

meaning that the information revealed by the fact that $\xi_1 = C_u$ is consistent with the information which is available in stage 1. By contrast, the process ζ_t defined as $\zeta_0 = \xi_0$, $\zeta_1 = \xi_2$, and $\zeta_2 = \xi_2$ is *not* adapted to $\{\mathcal{A}_t\}$. To see this, note that $\zeta_1^{-1}(C_{uu}) = \{(C, C_u, C_{uu}), (C, C_d, C_{uu})\} \notin \mathcal{A}_1$. The interpretation is

that the observation at stage 1 of ζ_1 being equal to C_{uu} reveals information that is not available in stage 1, namely that the value of ξ_2 in stage 2 will also be C_{uu}. This clairvoyance is impossible in multi-stage decision making, therefore the processes which are represented in multi-stage decision making problems need to be adapted to the filtration of the underlying probability space.

Finally, the introduction of the dynamic programming algorithm in the sequel will require the definition of conditional probability and conditional expectation.

The **conditional probability** of event A given event B is defined as

$$\mathbb{P}[A|B] = \begin{cases} \frac{\mathbb{P}[A\cap B]}{\mathbb{P}[B]}, & \mathbb{P}[B] > 0 \\ 0, & \mathbb{P}[B] = 0 \end{cases}$$

For a probability space with a countable sample space Ω, which is the focus in these notes, the **conditional expectation** can be defined as follows.

DEFINITION 2.3 Given a probabiity space $(\Omega, \mathcal{A}, \mathbb{P})$ and a random variable $X : \Omega \to \mathbb{R}^n$, the **conditional expectation** of X given an event $A \in \mathcal{A}$ with $\mathbb{P}[A] > 0$, denoted as $\mathbb{E}[X|A]$, is a vector in $\mathbb{R}^n$ defined as follows:

$$\mathbb{E}[X|A] = \frac{\sum_{\omega\in A} \mathbb{P}[\omega] \cdot X(\omega)}{\mathbb{P}[A]}.$$

Example 2.5 Consider the event $A = \{(C, C_u, C_{uu}), (C, C_u, C_{ud}), (C, C_u, C_{dd})\}$, namely the event that the stock price is C_u in stage 1. Further assume that, at any given stage, the probability of an upward movement of the stock price is 0.6, and the probability of a downward movement is 0.4. Then the conditional probability of the price in stage 2, given event A, is computed as follows:

$$\begin{aligned} &\mathbb{E}[X|A] = \\ &\frac{\mathbb{P}[(C, C_u, C_{uu})] \cdot C_{uu} + \mathbb{P}[(C, C_u, C_{ud})] \cdot C_{ud} + \mathbb{P}[(C, C_u, C_{dd})] \cdot C_{dd}}{\mathbb{P}[\{(C, C_u, C_{uu}), (C, C_u, C_{ud}), (C, C_u, C_{dd})\}]} \\ &= \frac{0.6 \cdot 0.6 \cdot C_{uu} + 0.6 \cdot 0.4 \cdot C_{uu} + 0.6 \cdot 0 \cdot C_{uu}}{0.6} = 0.6 \cdot C_{uu} + 0.4 \cdot C_{ud} \end{aligned}$$

The notion of a conditional expectation can be further from events to random variables as follows.

Consider two random vectors $x : \Omega \to \mathbb{R}^n$ and $y : \Omega \to \mathbb{R}^m$. Given two subsets $X \subseteq \mathbb{R}^n$ and $Y \subseteq \mathbb{R}^m$, the following conditional probability can be defined.

$$\mathbb{P}[X|Y] = \mathbb{P}[\{\omega : x(\omega) \in X\}|\{\omega : y(\omega) \in Y\}].$$

Using the above definition, the **conditional expectation** of random variable

x given that random variable y is equal to $v \in \mathbb{R}^m$ is given by

$$\mathbb{E}[x|v] = \sum_{i=1}^{\infty} x_i \cdot \mathbb{P}[\{\omega : x(\omega) = x_i\}|\{\omega : y(\omega) = v\}]$$

Note that $\mathbb{E}[x|v]$ is a random variable, and that

$$\mathbb{E}[\mathbb{E}[x|v]].$$

Example 2.6 Coming back to the stock price example, observe that ξ_3 conditional on ξ_2 is a random variable. To do this, obtain the mapping $\mathbb{E}[\xi_3|\xi_2] : \Omega \to \mathbb{R}$ by scanning over Ω. Then show that $\mathbb{E}[\mathbb{E}[\xi_3|\xi_2]] = \mathbb{E}[\xi_3]$.

2.1.2 Convex Analysis

DEFINITION 2.4 A **polyhedron** is a set in $\mathbb{R}^n$ which can be expressed as $\{x : Ax \leq b\}$, where $A \in \mathbb{R}^m \times \mathbb{R}^n$ and $b \in \mathbb{R}^m$.

DEFINITION 2.5 Consider a set of points $x_i \in \mathbb{R}^n, i = 1, \ldots, n$. A **convex combination** of these points is a point $\sum_{i=1}^n \lambda_i x_i$, such that $\sum_{i=1}^n \lambda_i = 1$ and $\lambda_i \geq 0, i = 1, \ldots, n$.

DEFINITION 2.6 X is a **convex set** if it contains any convex combination of points $x_i \in X$.

Geometrically, a convex combination of two points is a point that lies on the line connecting these points. A set is non-convex if it does not contain all the convex combinations of its points. These concepts are illustrated graphically in figure 2.2.

DEFINITION 2.7 An **extreme point** of a convex set is a point which cannot be expressed as the convex combination of two distinct points in the set.

DEFINITION 2.8 A point $r \in \mathbb{R}^n$ is a **ray** of a polyhedron P if and only if for any point $x \in P$, $\{y \in \mathbb{R}^n : y = x + \lambda r, \lambda \geq 0\} \subseteq P$.

DEFINITION 2.9 A ray r of P is an **extreme ray** if it cannot be expressed as a convex combination of other rays of P.

DEFINITION 2.10 The **convex hull** of a set of points is the set of all convex combinations of these points.

DEFINITION 2.11 f is a **convex function** if for all $0 \leq \lambda \leq 1$ and any x_1, x_2 we have $f(\lambda x_1 + (1 - \lambda)x_2) \leq \lambda f(x_1) + (1 - \lambda)f(x_2)$. f is **concave** if $-f$ is convex.

The definition of a convex function is illustrated graphically in figure 2.3.

Figure 2.2 Consider points $x_1, \ldots x_5 \in \mathbb{R}^2$. The point y is a convex combination of x_3 and x_5 because it can be expressed as $y = 0.5x_3 + 0.5x_5$. The set $X \subset \mathbb{R}^2$ is a convex set because any point in this set can be expressed as a convex combination of $x_1, \ldots, x_5$. The set $Y \subset \mathbb{R}^2$ is not convex, because there are convex combinations of points in Y which do not belong in Y, for example $y = 0.5y_1 + 0.5y_2$ does not belong in Y.

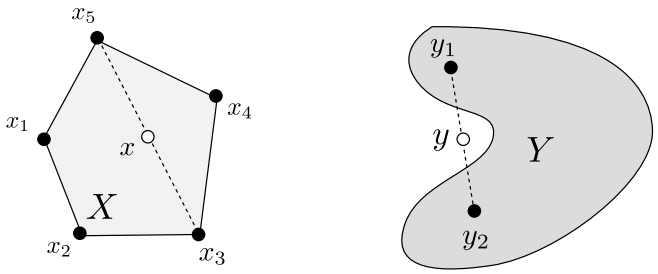
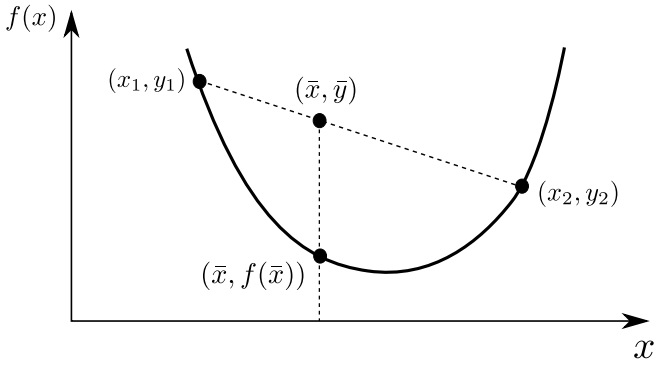


Figure 2.3 A graphical illustration of a convex function with a domain in $\mathbb{R}$. Consider points $x_1, x_2 \in \mathbb{R}$ with $y_1 = f(x_1)$ and $y_2 = f(x_2)$. f is a convex function because, for any points x_1, x_2, and for any choice of $0 \leq \lambda \leq 1$, the value of the function at $\bar{x} = \lambda x_1 + (1 - \lambda)x_2$, $f(\bar{x})$, is no greater than the convex combination $\bar{y} = \lambda y_1 + (1 - \lambda)y_2$.



DEFINITION 2.12 f is an **additively separable function** if it can be written as $f(x_1, x_2, \ldots, x_n) = f_1(x_1) + f_2(x_2) + \ldots + f_n(x_n)$.

The **domain** of f, dom f, is the set where the function is defined. For a function defined as $\mathbb{R}^n \to \mathbb{R}$, values in $\mathbb{R}^n$ which map to ∞ or $-\infty$ therefore do not belong to the domain of the function.

A continuous function f is **piecewise linear** if it can be written as

$$f(x) = \max_{i=1,\ldots,n} (a_i^T x + b_i)$$

for all $x \in$ dom f, where $a_i \in \mathbb{R}^n$, $b_i \in \mathbb{R}$, and n a finite integer number.

An **optimization problem** is the problem of finding the minimum[1] of a function f over a set $X \subset \mathbb{R}^n$:

$$z^\star = \min f(x)$$
$$\text{subject to } x \in X$$

X is the **feasible set** of the problem. f is the **objective function** of the problem. Any $x \in X$ is a **feasible solution**. Any $x^\star \in X$ such that $f(x^\star) \leq f(x)$ for any $x \in X$ is an **optimal solution**. 'Subject to' is abbreviated to 's.t.' or omitted in the sequel. If the optimization problem is infeasible, then by convention we have that $z^\star = +\infty$. If the problem is unbounded, then we have $z^\star = -\infty$.

DEFINITION 2.13 A **convex optimization problem** is an optimization problem with a convex objective function and a convex set of feasible solutions.

DEFINITION 2.14 A **supporting hyperplane** of a function $f(x)$ at x_0 is a set of points $a^T x + b$ such that $f(x_0) = a^T x_0 + b$ and $f(x) \geq a^T x + b$ for all x.

2.2 Linear programming

Linear programs represent an important class of convex optimization problems where the objective function and constraints are linear functions of decisions variables. Linear programs in **standard form** are expressed only in terms of non-negative variables and equality constraints:

$$(P) : \min z = c^T x$$
$$\text{s.t. } Ax = b$$
$$x \geq 0$$

Any linear program can be expressed in standard form by introducing slack variables. $x \in \mathbb{R}^n$ is the decision vector, $c \in \mathbb{R}^n$ is the vector of objective function coefficients, $A \in \mathbb{R}^{m \times n}$ is the matrix of constraint coefficients and $b \in \mathbb{R}^m$ is the vector of right-hand side parameters. A solution is a vector x such that $Ax = b$. A feasible solution is a solution with $x \geq 0$. An **optimal solution** is a feasible solution $x^\star$ such that $c^T x^\star \leq c^T x$ for all feasible solutions x.

A **basis** is a choice of n linearly independent columns of A, with $A = [B, N]$. Each basis has an associated **basic solution** $\begin{bmatrix} x_B \\ x_N \end{bmatrix}$ with $x_B = B^{-1}b$ and $x_N = 0$. The corresponding objective function value is computed as $z = c_B^T B^{-1} b$. Basic feasible solutions correspond to extreme points of the feasible region $\{x | Ax = b, x \geq 0\}$. A basis is feasible if $B^{-1}b \geq 0$ and it is **optimal** if $c_N^T - c_B^T B^{-1} N \geq 0$, as shown in the following proposition.

[1] The distinction between minimum and infimum operators will be ignored in the subsequent notes since the feasible sets considered in the problems presented in these notes are typically closed and bounded.

Table 2.1 The unit of nutrients in each dish for the diet problem of example 2.7.

	Dish 1	Dish 2	Dish 3
Nutrient 1	0.5	4	1
Nutrient 2	2	1	2

PROPOSITION 2.15 *A basic solution is optimal if* $B^{-1}b \geq 0$ *and* $c_N^T - c_B^T B^{-1} N \geq 0$.

Proof The feasibility of the basic solution follows immediately from the fact that $x_B = B^{-1}b \geq 0$ and $x_N = 0$. Consider how the objective function value changes in the neighborhood of the current solution. For this purpose, it is necessary to substitute basic variables for non-basic variables in the objective function $c^T x$. Note that

$$\begin{aligned} \begin{bmatrix} B & N \end{bmatrix} \begin{bmatrix} x_B \\ x_N \end{bmatrix} = b \Leftrightarrow \\ Bx_B + Nx_N = b \Leftrightarrow \\ x_B = B^{-1}(b - Nx_N) \end{aligned} \tag{2.1}$$

Substituting equation (2.1) into the objective function,

$$\begin{aligned} c^T x &= c_B^T x_B + c_N^T x_N \\ &= c_B^T B^{-1} b + (c_N^T - c_B^T B^{-1} N) x_N \end{aligned} \tag{2.2}$$

Note that the first term is a constant. Since non-basic variables can only increase when moving away from the current solution while remaining feasible, the second term of equation (2.2) cannot decrease the objective function when moving to *any* feasible solution. It therefore follows that the solution (x_B^T, x_N^T) is optimal.

□

The result of the above proposition therefore provides sufficient conditions for the optimality of a candidate solution. The non-negativity of reduced cost is also sufficient for guaranteeing the optimality of a solution, provided the basic solution is non-degenerate, meaning that none of the basic variables are equal to zero (see theorem 3.1 in page 86 of (Bertsimas & Tsitsiklis 1997)).

For the algorithms that are developed in the text, it is important to understand how the optimal value of a linear program depends on the right-hand side parameters b. Insights can be gained by examining both the primal linear program, as well as its dual. First, consider the following example.

Example 2.7 Consider the following instance of the diet problem. A least-cost diet is sought, that consists of three dishes, x_1, x_2 and x_3, and includes two types of nutrients in quantities b_1 and b_2 respectively. The first, second and third dish

cost 1 \$, 2 \$, and 1 \$ per unit respectively. The amount of each nutrient contained in each dish is presented in table 2.1. The problem can be expressed as follows, parametrically in b:

$$\begin{aligned} \min\ & x_1 + 2x_2 + x_3 \\ \text{s.t.}\ & 0.5x_1 + 4x_2 + x_3 = b_1 \\ & 2x_1 + x_2 + 2x_3 = b_2 \\ & x_1, x_2, x_3 \geq 0 \end{aligned}$$

Note that the problem is already expressed in standard form. Since the columns of the constraint matrix A are linearly independent, three possible bases can be constructed. They are as follows:

$$B_1 = \begin{bmatrix} 0.5 & 4 \\ 2 & 1 \end{bmatrix}, B_2 = \begin{bmatrix} 0.5 & 1 \\ 2 & 2 \end{bmatrix}, B_3 = \begin{bmatrix} 4 & 1 \\ 1 & 2 \end{bmatrix}$$

Note that the reduced cost corresponding to bases B_1, B_2 and B_3 is -0.2, 1.5, and 0.2143 respectively. Therefore, unless any basic solutions are degenerate, only the second and third bases stand a chance at being optimal. The bases lead to the following basic solutions, parametrized with respect to (b_1, b_2):

$$\begin{aligned} x_{B_1} &= \begin{bmatrix} -0.1333b_1 + 0.5333b_2 \\ 0.2667b_1 - 0.0667b_2 \end{bmatrix} \\ x_{B_2} &= \begin{bmatrix} -2b_1 + b_2 \\ 2b_1 - 0.5b_2 \end{bmatrix} \\ x_{B_3} &= \begin{bmatrix} 0.2857b_1 - 0.1429b_2 \\ -0.1429b_1 + 0.5714b_2 \end{bmatrix} \end{aligned}$$

In order for a basic solution to be feasible, it is necessary that b be such that $x_B \geq 0$. Denoting $R_i = \{(b_1, b_2) : x_{B_i} \geq 0\}$, the following polyhedra are obtained:

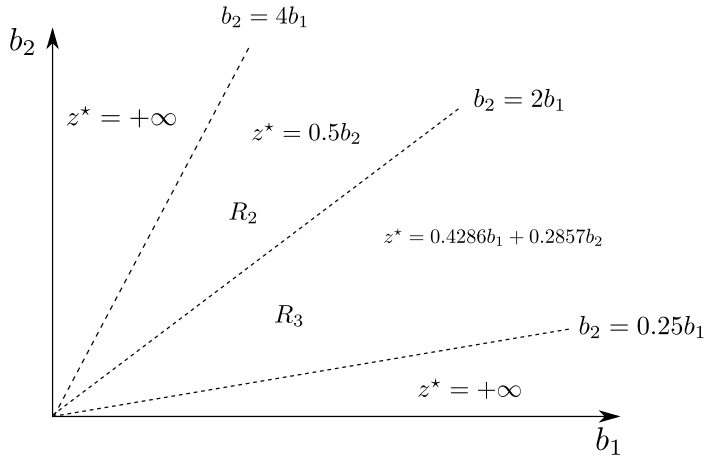
$$\begin{aligned} R_1 &= \{0.25b_1 \leq b_2 \leq 4b_1\} \\ R_2 &= \{2b_1 \leq b_2 \leq 4b_1\} \\ R_3 &= \{0.25b_1 \leq b_2 \leq 2b_1\} \end{aligned}$$

The cost of each basic solution, parametrized with respect to (b_1, b_2), can be expressed as follows:

$$\begin{aligned} c_{B_1}^T x_{B_1} &= 0.4b_1 + 0.4b_2 \\ c_{B_2}^T x_{B_2} &= 0.5b_2 \\ c_{B_3}^T x_{B_3} &= 0.4286b_1 + 0.2857b_2 \end{aligned}$$

It has been shown in proposition 2.15 that if a basis is feasible and results in non-negative reduced cost, then the corresponding basic solution is optimal. From this it can be concluded that x_{B_2} is optimal in region R_2, and x_{B_3} is optimal in region R_3, as shown in figure 2.4. Note that the optimal value of the problem is a piecewise linear function of the right-hand side vector b. Region R_2 corresponds

Figure 2.4 The space of right-hand side parameters b of the diet problem of example 2.1 can be partitioned into subsets R_i where each basis is optimal. The optimal value $z^\star$ of the diet problem is piecewise linear with respect to b.



to a mixture of dishes 1 and 3. Region R_3 corresponds to a mixture of dishes 2 and 3.

The dual of problem (P) is the following linear program:

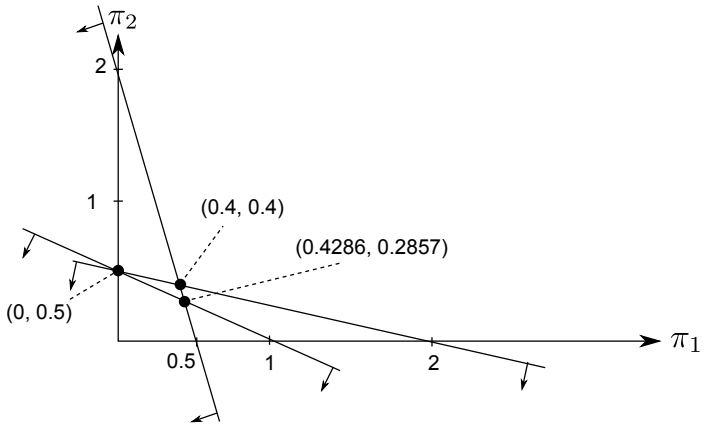
$$\begin{aligned}(D) : \max\, & \pi^T b \\ \text{s.t. } & \pi^T A \leq c^T\end{aligned}$$

If the dual problem is unbounded, the primal problem is infeasible and if the primal problem is unbounded then the dual problem is infeasible. Both the primal and dual problem can be infeasible. If x is a primal feasible solution and π is a dual feasible solution, then the objective function of the primal problem is an upper bound on the objective function of the dual problem: $c^T x \geq \pi^T b$. There exists a primal optimal solution $x^\star$ if and only if there exists a dual optimal solution $\pi^\star$, in which case **strong duality** holds: $c^T x^\star = (\pi^\star)^T b$. In this case, **complementary slackness** also holds: $(\pi^\star)^T A \perp c^T$. The complementarity operator $\perp$ applied to scalars implies that their product is zero (i.e. $a \perp b$ is equivalent to $a = 0$, $b = 0$ or both). The complementarity operator applied to vectors implies that the entries of the vectors are complementary componentwise. The mnemonic rules for deriving the dual of a linear program can be found in table 2.2, which is sourced from (Bertsimas & Tsitsiklis 1997).

Table 2.2 Linear programming duality mnemonic table.

Primal	Minimize	Maximize	Dual
Constraints	$\geq b_i$	≥ 0	Variables
	$\leq b_i$	≤ 0	
	$= b_i$	Free	
Variables	≥ 0	$\leq c_j$	Constraints
	≤ 0	$\geq c_j$	
	Free	$= c_j$	

Figure 2.5 The dual feasible region of example 2.8. Each black dot is a basic solution of the dual feasible region and corresponds to a basis of the primal problem in standard form.



Example 2.8 Consider again the diet problem of example 2.7. Using the mnemonic of table 2.2, the dual problem is obtained as follows:

$$\begin{aligned} \max\ & b_1\pi_1 + b_2\pi_2 \\ \text{s.t.}\ & 0.5\pi_1 + 2\pi_2 \leq 1 \\ & 4\pi_1 + \pi_2 \leq 2 \\ & \pi_1 + 2\pi_2 \leq 1 \end{aligned}$$

The feasible region is shown in figure 2.5.

A **basic solution** of a polyhedron $P \subset \mathbb{R}^n$ (not necessarily in standard form) that is defined by linear equalities and inequalities is defined as a vector x such that (i) all equality constraints are active and (ii) out of the constraints that are active at x, n are linearly independent.

For linear programs in standard form, each basis B of the primal coefficient

matrix A corresponds to a basic solution of the dual feasible set, according to the relationship $\pi^T = c_B^T B^{-1}$. Conversely, for every basic solution of the dual feasible region there exists a basis of the primal matrix such that the previous relationship holds. If a dual optimal solution exists, then one dual basic solution must be optimal. Therefore, the dual problem can be written equivalently as

$$\max_{i=1,\ldots,r} \pi_i^T b$$

where r indicates the number of bases of A (there are finitely many of them) and $\pi_i^T = c_{B_i}^T B_i^{-1}$ is the corresponding dual basic solution. It can therefore be concluded that if the optimal value of the linear program, $z(b)$, is parametrized with respect to b, then it is a piecewise linear function of b. One can argue similarly for linear programs in non-standard form by converting them to linear programs in standard form.

Example 2.9 Recall from example 2.7 that three bases correspond to the primal problem. From these bases, the following basic solutions can be computed for the dual feasible region, according to the relation $\pi^T = c_B^T B^{-1}$:

$$(\pi_1, \pi_2) = (0.4, 0.4), (\pi_1, \pi_2) = (0, 0.5), (\pi_1, \pi_2) = (0.4286, 0.2857)$$

The dual basic solutions are indicated in figure 2.5.

2.3 Duality

This section presents elementary definitions and results in duality that are useful for developing the algorithms that are presented in the text.

Duality theory is based on the notion of allowing certain constraints of an optimization problem to be violated at a penalty. This gives rise to a new optimization problem, called the dual problem, whose optimal solution will be used in the design of the cutting plane algorithms that will be presented in this text. In order to define the dual problem, it is necessary to first define a Lagrangian function.

Consider the following optimization problem:

$$\begin{aligned} p^\star &= \min f_0(x) \\ &\text{s.t. } f_i(x) \leq 0, i = 1, \ldots, m \\ &h_i(x) = 0, i = 1, \ldots, p \\ &x \in \text{dom } f_0 \subset \mathbb{R}^n \end{aligned}$$

Clearly, any vector x that has a chance at being optimal should lie within dom f. The m inequality constraints, $f_i(x) \leq 0$, and p equality constraints, $h_i(x) = 0$, further restrict the feasible set of vectors that should be considered.

Consider, now, the possibility of *relaxing* the inequality and equality constraints: allowing them to be violated, but penalizing their violation. In particular, consider the following function **Lagrangian function**, whose domain is a subset of $\mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^p$:

$$L(x, \lambda, \nu) = f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \sum_{i=1}^{p} \nu_i h_i(x)$$

The Lagrangian function can be interpreted as a new objective function which adds to the objective function f_0 a weighted sum of the constraint functions that are relaxed. The weights λ_i (for inequality constraints) and μ_i (for equality constraints) are referred to as **Lagrange multipliers**, and can be interpreted as the penalty coefficients by which the violation of the relaxed constraints is penalized.

In the remainder of the notes, the multipliers that correspond to a certain set of constraints will be placed to the left of the corresponding constraints in the formulation of the problem, as in the following notation:

$$\begin{aligned} & \min f_0(x) \\ & \text{s.t. } x \in \text{dom } f_0 \subset \mathbb{R}^n \\ (\lambda_i) : & f_i(x) \leq 0, i = 1, \ldots, m \\ (\mu_i) : & h_i(x) = 0, i = 1, \ldots, p \end{aligned}$$

Now consider fixing the Lagrange multipliers λ and ν to a certain value. The following **Lagrange dual function** is defined:

$$\begin{aligned} g(\lambda, \nu) &= \min_{x \in \text{dom } f_0} L(x, \lambda, \nu) \\ &= \min_{x \in \text{dom } f_0} (f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \sum_{i=1}^{p} \nu_i h_i(x)) \end{aligned}$$

Note that the dual function $g(\lambda, \nu)$ can be $-\infty$ for some λ, ν. The values of (λ, ν) for which the dual function is not $-\infty$ correspond to the domain of the dual function.

It is easy to check that the dual function provides a lower bound to the original optimization problem if $\lambda \geq 0$. To see this, note that the minimization of the dual function permits a violation of the relaxed constraints, but if they are respected then any feasible solution x receives a 'bonus' in the Lagrangian function whenever $\lambda \geq 0$, in the sense that L evaluates to something less than $f_0(x)$. This argument is developed formally in the following proposition:

PROPOSITION 2.16 *If $\lambda \geq 0$ then $g(\lambda, \nu) \leq p^\star$.*

Proof Consider an $\bar{x}$ such that $f_i(\bar{x}) \leq 0$ and $h_i(\bar{x}) = 0$. For $\lambda \geq 0$,

$$f_0(\bar{x}) \geq L(\bar{x}, \lambda, \nu) \geq \min_{x \in \text{dom } f_0} L(x, \lambda, \nu) = g(\lambda, \nu).$$

Minimizing over all $\bar{x}$ such that $f_i(\bar{x}) \leq 0$ and $h_i(\bar{x}) = 0$ results in the desired inequality, $p^\star \geq g(\lambda, \nu)$.

□

The dual function $g(\lambda, \nu)$ is *always* a concave function, even if $f_0(x)$, $f_i(x)$ or $h_i(x)$ are not necessarily convex functions.

PROPOSITION 2.17 *$g(\lambda, \nu)$ is a concave function.*

Proof Consider any (λ_1, ν_1), (λ_2, ν_2). Consider $\alpha \in [0, 1]$. Then,

$$\begin{aligned}
&g(\alpha\lambda_1 + (1-\alpha)\lambda_2, \alpha\nu_1 + (1-\alpha)\nu_2) \\
&= \min_{x \in \text{dom } f_0} (f_0(x) + \sum_{i=1}^{m} (\alpha\lambda_{1,i} f_i(x) + (1-\alpha)\lambda_{2,i} f_i(x)) \\
&+ \sum_{i=1}^{p} (\alpha\nu_{1,i} h_i(x) + (1-\alpha)\nu_{2,i} h_i(x))) \\
&\geq \alpha \min_{x \in \text{dom } f_0} (f_0(x) + \sum_{i=1}^{m} \lambda_{1,i} f_i(x) + \sum_{i=1}^{p} \nu_{1,i} h_i(x)) \\
&+ (1-\alpha) \min_{x \in \text{dom } f_0} (f_0(x) + \sum_{i=1}^{m} \lambda_{2,i} f_i(x) + \sum_{i=1}^{p} \nu_{2,i} h_i(x)) \\
&= \alpha g(\lambda_1, \nu_1) + (1-\alpha) g(\lambda_2, \nu_2)
\end{aligned}$$

□

Example 2.10 Consider the following stochastic program:

$$\begin{aligned}
&\min_{x,y} \ f1(x) + \mathbb{E}[f2(y(\omega), \omega)] \\
&\text{s.t.} \quad h1_i(x) \leq 0, i = 1, \ldots, m_1, \\
&\qquad h2_i(x, y(\omega), \omega) \leq 0, i = 1, \ldots, m_2, \omega \in \Omega
\end{aligned}$$

where Ω is a sample set, $x \in \mathbb{R}^{n_1}$ is a set of first-stage decisions that are made prior to the revelation of uncertainty (which is why these decisions are not indexed by ω), $f_1 : \mathbb{R}^{n_1} \to \mathbb{R}$ is a cost function depending on first-stage decisions, $y(\omega) \in \mathbb{R}^{n_2}$ are second-stage decisions that are made after uncertainty is revealed (which is why they are indexed by ω), $f_2 : \mathbb{R}^{n_2} \times \Omega \to \mathbb{R}$ are second-stage costs, $h1_i : \mathbb{R}^{n_1} \to \mathbb{R}, i = 1, \ldots, m_1$ are first-stage constraint functions, and $h2_i : \mathbb{R}^{n_2} \to \mathbb{R}, i = 1, \ldots, m_2$ are second-stage constraint functions. Consider the

following reformulation of the problem:

$$\begin{aligned} \min \quad & \mathbb{E}[f1(x) + f2(y(\omega), \omega)] \\ \text{s.t.} \quad & h1_i(x(\omega)) \leq 0, i = 1, \ldots, m_1, \omega \in \Omega \\ & h2_i(x(\omega), y(\omega), \omega) \leq 0, i = 1, \ldots, m_2, \omega \in \Omega \\ (\nu(\omega)) : & \; x(\omega) = x, \omega \in \Omega \end{aligned} \tag{2.3}$$

Constraints (2.3) are referred to as **non-anticipativity constraints**, and they enforce the fact that first-stage actions cannot depend on an outcome that has not yet been observed. Consider relaxing the non-anticipativity constraints of equation (2.3). Then the following dual function is obtained:

$$g(\nu) = g1(\nu) + \mathbb{E}[g2(\nu, \omega)]$$

where

$$\begin{aligned} g1(\nu) = \min_x \; & f1(x) + (\sum_{\omega \in \Omega} \nu(\omega))^T x \\ \text{s.t.} \; & h1_i(x) \leq 0, i = 1, \ldots, m_1, \end{aligned}$$

and

$$\begin{aligned} g2(\nu, \omega) = \min_{x(\omega), y(\omega)} \; & f2(y(\omega), \omega) - \nu(\omega)^T x(\omega) \\ \text{s.t.} \quad & h2_i(x(\omega), y(\omega), \omega) \leq 0, i = 1, \ldots, m_2 \end{aligned}$$

with $\nu^T = (\nu(\omega_1)^T, \ldots, \nu(\omega_{|\Omega|})^T)$ denoting the vector of dual multipliers. The benefit of this relaxation is that the computation of $g(\nu)$ for a given choice of ν can be broken into $|\Omega| + 1$ *independent* optimization problems that should in general be easier to compute than the original optimization problem (especially for very large sets $|\Omega|$).

It is natural to maximize the dual function in order to obtain the tightest possible lower bound from the relaxation of the problem. This gives rise to the **Lagrange dual problem:**

$$\begin{aligned} & d^\star = \max g(\lambda, \nu) \\ & \text{s.t. } \lambda \geq 0 \end{aligned}$$

In general, this problem can be easier to solve compared to the original optimization problem, because the dual function is concave and there exists a large family of efficient algorithms for convex optimization problems, i.e. for minimizing convex functions (or maximizing concave functions) over convex sets.

Since $g(\lambda, \nu) \leq p^\star$ for any λ, ν, it follows that $p^\star \geq d^\star$. This relationship is referred to as **weak duality** and holds for any optimization problem, whether it is convex or not. In fact, $d^\star$ often yields non-trivial bounds to difficult problems. **Strong duality** refers to the case where $p^\star = d^\star$ and does not hold in general.

If the relationship holds, then the original optimization problem is equivalent to solving the maximization of the dual. Strong duality usually holds for convex problems. The conditions that guarantee strong duality in convex problems are called constraint qualifications.

Example 2.11 Use duality to prove the mnemonic rules of table 2.2.

Solution

First note that every linear program can be written in the following form

$$\begin{aligned}
&\min c_1^T x_1 + c_2^T x_2 + c_3^T x_3 \\
&\text{s.t. } A_{11}x_1 + A_{12}x_2 + A_{13}x_3 \geq b_1, (\lambda_1) \\
&A_{21}x_1 + A_{22}x_2 + A_{23}x_3 \leq b_2, (\lambda_2) \\
&A_{31}x_1 + A_{32}x_2 + A_{33}x_3 = b_3, (\lambda_3) \\
&x_1 \geq 0, (\sigma_1) \\
&x_2 \leq 0, (\sigma_2)
\end{aligned}$$

This results in the following Lagrangian function:

$$L(x, \lambda, \sigma) = \sum_{j=1}^{3} c_j^T x_j - \sum_{i=1}^{3} \lambda_i^T \left(\sum_{j=1}^{3} A_{ij} x_j - b_i \right) - \sum_{j=1}^{2} \sigma_j^T x_j$$

For $L(x, \lambda, \sigma)$ to be a lower bound to the problem solution, it must be that $\lambda_1 \geq 0$, $\lambda_2 \leq 0$, $\sigma_1 \geq 0$ and $\sigma_2 \leq 0$. Furthermore, as x_j are free variables (since the non-negativity constraints have been relaxed), the dual function $g(\lambda, \sigma)$ will be finite if and only if

$$\begin{aligned}
&c_j^T - \sum_{i=1}^{3} \lambda_i^T A_{ij} - \sigma_j^T = 0, \quad j = 1, 2 \\
&c_3^T - \sum_{i=1}^{3} \lambda_i^T A_{i3} = 0
\end{aligned}$$

In this domain, the dual function becomes $g(\lambda, \sigma) = \sum_{i=1}^{3} b_i^T \lambda_i$. Taking into account the sign of σ, the dual problem can be stated as

$$\begin{aligned}
&\max \quad b_1^T \lambda_1 + b_2^T \lambda_2 + b_3^T \lambda_3 \\
&\text{s.t.} \quad A_{11}^T \lambda_1 + A_{21}^T \lambda_2 + A_{31}^T \lambda_3 \leq c_1 \\
&A_{12}^T \lambda_1 + A_{22}^T \lambda_2 + A_{32}^T \lambda_3 \geq c_2 \\
&A_{13}^T \lambda_1 + A_{23}^T \lambda_2 + A_{33}^T \lambda_3 = c_3 \\
&\lambda_1 \geq 0 \\
&\lambda_2 \leq 0
\end{aligned}$$

which proves the mnemonic rules.

Mathematical programming software that is used for developing decomposition algorithms such as SDDP relies on manipulating dual optimal multipliers. The optimality of dual multipliers is typically verified through the Karush Kuhn Tucker conditions (KKT conditions), which are conditions for characterizing the optimal solution of an optimization problem. The following proposition is useful for the subsequent development:

Consider again the following optimization problem:

$$\begin{aligned} &\min f_0(x) \\ &\text{s.t. } x \in \text{dom } f_0 \subset \mathbb{R}^n \\ (\lambda_i) &: f_i(x) \leq 0, i = 1, \ldots, m \\ (\mu_i) &: h_i(x) = 0, i = 1, \ldots, p \end{aligned}$$

PROPOSITION 2.18 *Suppose that strong duality holds, with $x^\star$ optimal for the primal problem, and $\lambda^\star, \nu^\star$ optimal for the dual problem. Then*

- *$x^\star$ minimizes $L(x, \lambda^\star, \nu^\star)$*
- *$\lambda_i^\star f_i(x^\star) = 0$ for $i = 1, \ldots, m$*

Proof Strong duality implies the following:

$$\begin{aligned} f_0(x^\star) = g(\lambda^\star, \nu^\star) &= \min_x (f_0(x) + \sum_{i=1}^m \lambda_i^\star f_i(x) + \sum_{i=1}^p \nu_i^\star h_i(x)) \\ &\leq f_0(x^\star) + \sum_{i=1}^m \lambda_i^\star f_i(x^\star) + \sum_{i=1}^p \nu_i^\star h_i(x^\star) \\ &\leq f_0(x^\star) \end{aligned}$$

The first equality holds by strong duality. The second equality holds by definition of the dual function. The first inequality holds because $x^\star$ is not necessarily the minimizer of the Lagrangian function. The second inequality holds because $h_i(x^\star) = 0$ for all $i = 1, \ldots, p$, $f_i(x^\star) \leq 0$ for all $i = 1, \ldots, m$ and $\lambda_i^\star \leq 0$ for all $i = 1, \ldots, m$. Since the left and right-hand side are identical, the two inequalities above hold with equality. The conclusions follow.

□

DEFINITION 2.19 Consider a problem with differentiable objective function and constraints. The following four conditions are called **KKT conditions**:

- Primal feasibility: $f_i(x) \leq 0$, $i = 1, \ldots, m$, $h_i(x) = 0, i = 1, \ldots, p$
- Dual feasibility: $\lambda_i \geq 0$, $i = 1, \ldots, m$
- Complementary slackness: $\lambda_i f_i(x) = 0$, $i = 1, \ldots, m$
- Gradient of the Lagrangian function with respect to x vanishes:

$$\nabla f_0(x) + \sum_{i=1}^m \lambda_i \nabla f_i(x) + \sum_{i=1}^p \nu_i \nabla h_i(x) = 0.$$

Proposition 2.18 proves that the KKT conditions are necessary for problems for which strong duality holds. However, in general they are not sufficient conditions for optimality. For convex optimization problems with differentiable objective functions and constraints, the KKT conditions are sufficient, and further guarantee that the problem has a zero duality gap. The proof is left as an exercise.

The following result provides a useful mnemonic for writing out the KKT conditions of an optimization problem with linear constraints.

PROPOSITION 2.20 *Consider a maximization problem with differentiable objective function f and linear constraints:*

$$\begin{aligned} & \max f(x,y) \\ & s.t. \\ (\lambda): \ & Ax + By \leq b \\ (\mu): \ & Cx + Dy = d \\ (\lambda_2): \ & x \geq 0 \end{aligned}$$

Then the KKT conditions have the following form:

$$\begin{aligned} & Cx + Dy - d = 0 \\ & 0 \leq \lambda \perp Ax + By - b \leq 0 \\ & 0 \leq x \perp \lambda^T A + \mu^T C - \nabla_x f(x,y)^T \geq 0 \\ & \lambda^T B + \mu^T D - \nabla_y f(x,y)^T = 0 \end{aligned}$$

Proof The Lagrangian function can be expressed as

$$L(x,y,\lambda,\mu) = f(x) + \lambda^T(b - Ax - By) + \mu^T(d - Cx - Dy) + \lambda_2^T x$$

In order for the Lagrangian function to be an upper bound to the optimal value, it must be the case that $\lambda, \lambda_2 \geq 0$. These are the dual feasibility conditions. The complementarity condition can be stated as:

$$\begin{gathered} \lambda^T(b - Ax - By) = 0 \Leftrightarrow \lambda \perp b - Ax - By \\ \lambda_2^T x = 0 \Leftrightarrow \lambda_2 \perp x \end{gathered}$$

The stationarity condition can be stated as

$$\begin{aligned} \nabla_x L(x,y,\lambda,\sigma) &= \nabla f(x)^T - \mu^T C - \lambda^T A + \lambda_2^T = 0 \\ \nabla_y L(x,y,\lambda,\sigma) &= \nabla f(x)^T - \mu^T C - \lambda^T A = 0 \end{aligned}$$

Therefore, the KKT conditions can be expressed as

$$\begin{aligned}
&0 \leq \lambda \perp b - Ax - By \geq 0 \\
&Cx + Dy - d = 0 \\
&0 \leq \lambda_2 \perp x \geq 0 \\
&\lambda_2^T = \lambda^T A + \mu^T C - \nabla_x f(x,y)^T \\
&\lambda^T B + \mu^T D - \nabla_y f(x,y)^T = 0
\end{aligned}$$

or equivalently

$$\begin{aligned}
&Cx + Dy - d = 0 \\
&0 \leq \lambda \perp Ax + By - b \leq 0 \\
&0 \leq x \perp \lambda^T A + \mu^T C - \nabla_x f(x,y)^T \geq 0 \\
&\lambda^T B + \mu^T D - \nabla_y f(x,y)^T = 0
\end{aligned}$$

□

Example 2.12 Consider the diet problem of example 2.7, with $b_1 = 1$ unit of the first nutrient and $b_2 = 1$ unit of the second nutrient required in the diet:

$$\begin{aligned}
&\min x_1 + 2x_2 + x_3 \\
&(\pi_1) : 0.5x_1 + 4x_2 + x_3 = 1 \\
&(\pi_2) : 2x_1 + x_2 + 2x_3 = 1 \\
&x_1, x_2, x_3 \geq 0
\end{aligned}$$

The KKT conditions of the problem can be expressed by using the mnemonic of the previous proposition:

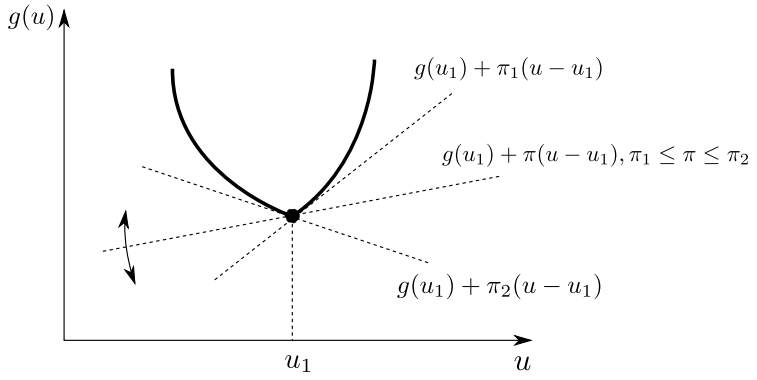
$$0.5x_1 + 4x_2 + x_3 = 1 \tag{2.4}$$
$$2x_1 + x_2 + 2x_3 = 2 \tag{2.5}$$
$$0 \leq x_1 \perp 0.5\pi_1 + 2\pi_2 + 1 \geq 0 \tag{2.6}$$
$$0 \leq x_2 \perp 4\pi_1 + \pi_2 + 2 \geq 0 \tag{2.7}$$
$$0 \leq x_3 \perp \pi_1 + 2\pi_2 + 1 \geq 0 \tag{2.8}$$

Equations (2.4) - (2.8) constitute the KKT conditions of the optimization problem. Note that the last three conditions can be replaced by the following conditions:

$$0 \leq x_1 \perp -0.5\pi_1 - 2\pi_2 + 1 \geq 0 \tag{2.9}$$
$$0 \leq x_2 \perp -4\pi_1 - \pi_2 + 2 \geq 0 \tag{2.10}$$
$$0 \leq x_3 \perp -\pi_1 - 2\pi_2 + 1 \geq 0 \tag{2.11}$$

since an equality constraint $h(x) = 0$ can be expressed equivalently as $-h(x) = 0$. The primal optimal solution of the problem is $(x^\star)^T = (0, 0.1429, 0.4286, 0)$. The

Figure 2.6 Consider the function $g : \mathbb{R} \to \mathbb{R}$ and notice that the function is non-differentiable at u_1. π_1 and π_2 are both subgradients at u_1. Any line $y = g(u_1) + \pi(u - u_1)$ passing through $(u_1, g(u_1))$ with a slope of $\pi \in \partial g(u_1)$ is below $g(u)$ for all u, whereas any line with a different slope cuts into' $g(u)$. $\partial g(u_1) = [\pi_1, \pi_2]$ is the subdifferential of g at u_1.



vector $(\pi^\star)^T = (0.4286, 0.2857)$ is optimal. The reader should verify that $x^\star$ and $\pi^\star$ satisfy equations (2.4) - (2.5) and (2.9) - (2.11).

Note that if the constraints or objective function of the problem are non-differentiable, then the KKT conditions can be generalized by replacing the gradient of the Lagrangian function with its subgradient[2] in the fourth condition. If strong duality holds, then these conditions remain necessary and sufficient for optimality.

2.4 Subgradients

Subgradients are used extensively in the cutting plane algorithms that are developed in the text. Subgradients generalize the notion of gradients for non-differentiable functions.

DEFINITION 2.21 Consider a function g, π is a **subgradient** of g at u if

$$g(w) \geq g(u) + \pi^T(w - u) \text{ for all } w. \tag{2.12}$$

If $g(w) \leq g(u) + \pi^T(w - u)$ for all w, then π is a **supergradient**.

The geometric interpretation of a subgradient is that it is a vector that defines a linear under-estimator of a function g, as shown in figure 2.6.

[2] Subgradients are defined and discussed in detail in section 2.4.

DEFINITION 2.22 The set of all subgradients of g at u is called the **subdifferential** of g at u, and is denoted as $\partial g(u)$.

As one can observe in figure 2.6, $\partial g(u)$ is a closed convex set. It should be intuitively obvious that if a function g is differentiable at a certain point u, then $\partial g(u) = \{\nabla g(u)\}$, i.e. the subdifferential consists of a single element which is the gradient of the function at that point. It is also easy to show that, for convex functions, $0 \in \partial g(u)$ implies that u is a global minimum of g.

The definition of subgradients is useful for describing the slope of the following function:

$$\begin{aligned} & c(u) = \min f_0(x) \\ & \text{s.t. } f_i(x) \leq u_i, i = 1, \ldots, m \\ & x \in \text{dom } f_0 \end{aligned}$$

The function $c(u)$, which is often referred to as the *value function* of the optimization problem $\{\min f_0(x) : x \in \text{dom } f, f_i(x) \leq 0\}$, has a natural interpretation. Given a perturbation u to the right-hand side of the constraints $f_i(x) \leq 0$, the value function describes how the optimal objective function value is impacted. Therefore, the slope of $c(u)$ at $u = 0$ can be interpreted as the sensitivity of the optimal objective function value of the original problem to an infinitesimal perturbation of the right-hand side of its constraints. It turns out that this sensitivity is equal to the dual optimal multipliers of the original problem, as the following proposition demonstrates. This result is exploited in the cutting plane algorithms which are developed in detail in the following chapters.

PROPOSITION 2.23 *Define $c(u)$ as the optimal value of the following mathematical program:*

$$\begin{aligned} & c(u) = \min f_0(x) \\ & f_i(x) \leq u_i, i = 1, \ldots, m \\ & x \in dom\ f_0 \end{aligned}$$

and suppose that dom f_0 is a convex set and and f_0, f_i are convex functions.

- *$c(u)$ is a convex function.*
- *Suppose strong duality holds, and denote $\lambda^\star$ as the maximizer of the dual function $\min_{x \in dom\ f_0}(f_0(x) - \lambda^T(f(x) - u))$ for $\lambda \leq 0$. Then $\lambda^\star \in \partial c(u)$.*

Proof To show convexity of $c(u)$, consider any u_1, u_2 and denote x_1, x_2 as the optimal solution when u_1, u_2 is used as input respectively. Similarly, consider any $a \in [0, 1]$ and denote x_a as the optimal solution when $au_1 + (1-a)u_2$ is used as input. Since $f(x_1) \leq u_1$ and $f(x_2) \leq u_2$, it follows from the convexity of f that $f(ax_1 + (1-a)x_2) \leq au_1 + (1-a)u_2$. Since dom f_0 is convex, it follows that $ax_1 + (1-a)x_2$ is an admissible solution when $au_1 + (1-a)u_2$ is used as input. From the optimality of x_a with respect to $au_1 + (1-a)u_2$ it follows

that $f_0(x_a) \leq f_0(ax_1 + (1-a)x_2)$. It then follows from the convexity of f_0 that $c(au_1 + (1-a)u_2) \leq ac(u_1) + (1-a)c(u_2)$.

Consider any $\bar{u}$ and denote $\bar{x}$ as the optimal solution when $\bar{u}$ is used as input. Also, denote $x^\star \in \arg\min_{x \in \text{dom } f_0}(f_0(x) - (\lambda^\star)^T(f(x) - u))$. The following relationships hold:

$$\begin{gathered} c(u) = f_0(x^\star) - (\lambda^\star)^T(f(x^\star) - u) \leq \\ f_0(\bar{x}) - (\lambda^\star)^T(f(\bar{x}) - u) = \\ f_0(\bar{x}) - (\lambda^\star)^T(f(\bar{x}) - \bar{u}) - (\lambda^\star)^T(\bar{u} - u) \leq \\ f_0(\bar{x}) - (\lambda^\star)^T(\bar{u} - u) = \\ c(\bar{u}) - (\lambda^\star)^T(\bar{u} - u) \end{gathered}$$

where the first relationship follows from strong duality, the second relationship follows from the definition of $x^\star$, the fourth relationship results from the fact that $f(\bar{x}) \leq \bar{u}$ and $\lambda^\star \leq 0$ and the last relationship follows from the definition of $\bar{x}$.

□

Example 2.13 Recall the diet problem of the previous section. The optimal value $z(b)$ as a function of non-negative right-hand side parameters $b \geq 0$ was determined as

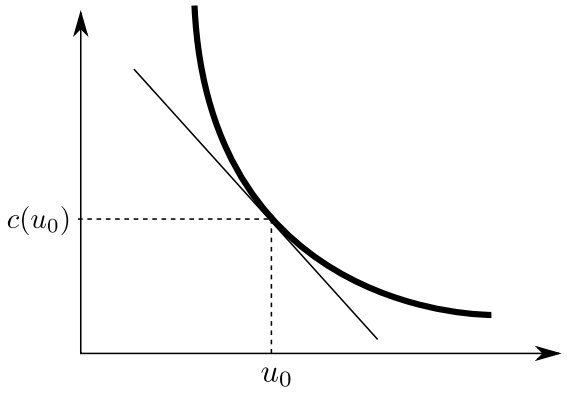
$$z(b) = \begin{cases} +\infty, & b_2 > 4b_1 \\ 0.5b_2, & 0.5b_1 \leq b_2 \leq 4b_1 \\ 0.4286b_1 + 0.2857b_2, & 0.25b_1 \leq b_2 \leq 2b_1 \\ +\infty, & b_2 < 0.25b_1 \end{cases}$$

This is a convex function of b.

A graphical illustration of the result is presented in figure 2.7. The result guarantees that when the optimal objective function value of a convex math program is parametrized with respect to the right hand side value of the constraints, then this objective function value is convex and its subgradient is obtained from the dual optimal multipliers of the math program. In the Benders decomposition algorithm of chapter 5, for example, the function $c(u)$ is the second-stage cost as a function of a first-stage decision u.

Notice that if $c(u)$ is differentiable at a certain point u, then the dual multiplier λ_i of a certain constraint is equal to $\nabla_{u_i} c(u)$, therefore λ_i is equal to the *sensitivity* of the objective function $c(u)$ to a marginal change in the right-hand-side of the constraint corresponding to λ_i.

Figure 2.7 Consider the function $c(u) = \{\min_{x \in \text{dom } f_0} f_0(x) : f_1(x) \leq u\}$. The function $c : \mathbb{R} \to \mathbb{R}$ is convex, and has a slope $\lambda^\star$ at u_0, where $\lambda^\star$ is the dual optimal multiplier of the relaxed constraint $f_1(x) \leq u_0$. The bold curve is the graph of $\{(u, c(u))\}$, the light solid curve corresponds to the graph of $\{(u, c(u_0) + \lambda^\star(u - u_0))\}$.



Example 2.14 Consider again the diet problem of example 2.7, with $b_1 = 1$ and $b_2 = 1$. From example 2.12 it is shown that $\pi_1^\star = 0.4286$ and $\pi_2^\star = 0.2857$ are dual optimal for the KKT system of equations (2.4) - (2.5) and (2.9) - (2.11). The sensitivity interpretation of $\pi_1^\star$ is that if the requirement for the first nutrient increases by ϵ, then the optimal cost increases by 0.4286ϵ. To see this, note that in order for both nutrient requirements to be satisfied the optimal quantity of each food becomes $x^\star = (0, 0.1429 + 0.2857\epsilon, 0.4286 - 0.1429\epsilon)$. The net cost effect of this change is $2 \cdot 0.2857\epsilon - 1 \cdot 0.1429\epsilon = 0.4286\epsilon$. It is important to note that the KKT conditions of the problem can also be expressed using equations (2.4) - (2.8), in which case the solution of the KKT system is $(-0.4286, -0.2857)$. Note the change in the sign of $\pi^\star$.

The previous example highlights that the dual optimal multiplier may be equal to the sensitivity, or minus the sensitivity, of the objective function value. As a general rule (as indicated by the proof of proposition 2.23), whether a Lagrange multiplier is equal to the sensitivity, or minus the sensitivity, of the objective function of an optimization problem depends on how the Lagrangian function of the problem is constructed. If the constraint $f_i(x) \leq 0$ (likewise for $h_i(x) = 0$) is relaxed by *subtracting* its weighted violation from the objective function (i.e. $L(x, \lambda) = f_0(x) - \lambda_i \cdot f_i(x)$), then the dual optimal multiplier is equal to the sensitivity of the objective function. By contrast, if the weighted violation is *added* to the objective function (i.e. $L(x, \lambda) = f_0(x) + \lambda_i \cdot f_i(x)$), then the dual

optimal multiplier is equal to *minus* the sensitivity of the objective function to a change in the right hand side of the constraint $f_i(x) \leq 0$ (likewise for $h_i(x) = 0$).

Problems

2.1 Derive the probability measure of period 2 for example **??**.

2.2 In the maximization problem of proposition 2.20, the dual multipliers μ are (prove your claim):

1. equal to the sensitivities of the objective function to the right-hand-side of the $Cx - d = 0$ constraints
2. equal to the sensitivities of the objective function to the right-hand-side of the $d - Cx = 0$ constraints

The dual multipliers λ are (prove your claim):

1. equal to the sensitivities of the objective function to the right-hand-side of the $Ax - b \leq 0$ constraints
2. equal to the sensitivities of the objective function to the right-hand-side of the $b - Ax \geq 0$ constraints

2.3 Consider the following optimization problem:

$$\begin{aligned} &\max x + 2y \\ &\text{s.t. } x \geq 0, (\lambda_1) \\ &x \leq 2, (\lambda_2) \\ &-y = -1, (\mu) \end{aligned}$$

What are the KKT conditions of the problem? What is the solution to the KKT system?

2.4 Prove the following properties of subgradients: if g is a convex function, then

- $\partial g(u)$ is nonempty, for $u \in$ relint dom g.
- $\partial g(u) = \{\nabla g(u)\}$ if g is differentiable at u.
- The above also holds in converse. If $\partial g(u) = \{\pi\}$, then g is differentiable at u and $\pi = \nabla g(u)$
- $\partial(ag) = a\partial g$.
- $\partial(g_1 + g_2) = \partial g_1 + \partial g_2$, where the right hand side operation corresponds to the addition of sets.
- If $f(u) = g(Au + b)$, then $\partial f(u) = A^T \partial g(Au + b)$
- If $g = \max_{i=1,\ldots,m} g_i$, then

$$\partial g(u) = \text{Co}(\cup\{\partial g_i(u) | g_i(u) = g(u)\})$$

where Co denotes the convex hull of a set of points. In words, the subdifferential is the convex hull of the union of the subdifferentials of 'active' functions at u.

Bibliography

Section 2.2.
Table 2.2 is based on (Bertsimas & Tsitsiklis 1997).

Section 2.3.
This material is based on (Boyd & Vandenberghe 2008).

Section 2.4.
This material is based on (Boyd & Vandenberghe 2008).

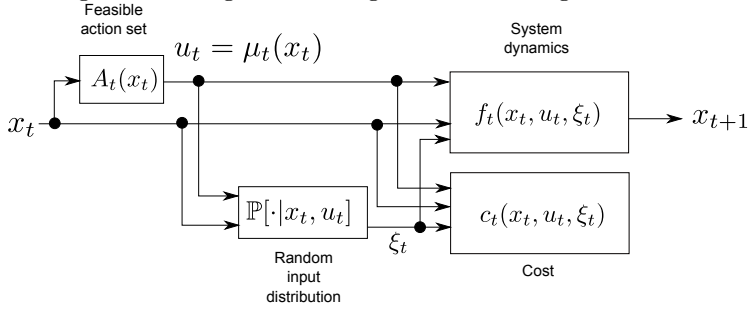
3 Dynamic Programming

This chapter presents a very general class of optimization problems, which can be described as multi-stage optimization problems under uncertainty (section 3.1), as well as a very general algorithm for tackling these problems, dynamic programming (section3.2). Section 3.3 provides an explanation of the mechanics of the dynamic programming algorithm, and how it delivers computational savings by intelligently storing information about the problem which does not need to be recomputed. The SDDP algorithm is a specific type of dynamic programming algorithm, and the link will be made clear later in the notes. However, it is important to emphasize that the dynamic programming algorithm is in fact useful for solving a much broader class of problems than those for which SDDP is designed. An indication of the versatility of the dynamic programming algorithm is provided in section 3.4, where a variety of problems from diverse applications are solved through dynamic programming.

3.1 Multi-Stage Optimization under Uncertainty

As discussed in the introduction, multi-stage stochastic linear programs which are the target application of these notes are a specific instance of multi-stage optimization under uncertainty. The general setting of multi-stage decision making problems under uncertainty is presented in figure 3.1. The characteristics of the system under consideration are the following:

Figure 3.1 The general setting of multi-stage decision making under uncertainty.



- The system is dynamical, and evolves over H discrete time stages. These notes will consider a finite horizon H. More general theory exists for continuous evolution of time, and infinite horizon problems, however this is not the focus of these notes.
- The system is controlled, in the sense that its evolution can be influenced by a decision-maker. Let u_t denote the decision made at each stage. These notes will allow for u_t to take on continuous values.
- The system is uncertain, in the sense that its evolution is influenced by realizations of a random vector ξ_t. In these notes, this random vector will typically be discretized (i.e. it will take values in a finite set), and will follow a discrete probability distribution, although the general theory of dynamic programming can accommodate continuously valued uncertainty.
- The state of the system is summarized in a state vector x_t. In the majority of applications considered in these notes, this state vector will be allowed to assume continuous values. The **state** of the system encodes *all* the information, apart from the action of the decision maker u_t and the random disturbance ξ_t, which is required in order to describe the evolution of the system in the next time stage. The dynamic evolution of the system is described by the following transition equation:

$$x_{t+1} = f_t(x_t, u_t, \xi_t), t = 0, \ldots, H-1.$$

- The system is Markovian, in the sense that the evolution of the system uncertainty depends only on the current state of the system, and not past states of the system. More formally, it is possible to define a probability distribution $\mathbb{P}[\cdot|x_t, u_t]$ for ξ_t, which may depend on x_t and u_t, but not on past information.
- The cost incurred by the decision maker is additive over time, in the sense that the decision maker incurs at each stage a cost $c_t(x_t, u_t, \xi_t)$, and the goal of the decision maker is to minimize the *average* accumulated cost over time. This will be further clarified in the following paragraphs.
- The control is limited to a set of values which depend on the current state of the system, i.e. $u_t \in A_t(x_t)$.

From figure 3.1 it is clear that the model is defined consistently, in the sense that each of the functions and sets defined in the model can be determined based on the information that is already available. The process depicted in figure 3.1 is repeated identically at every stage, until the end of the optimization horizon. Note that the only information affecting the process at stage t is information related to stage t, and not earlier stages, which is why the process is referred to as a Markov decision process. There is a specific timing of information which is implied by the model. In a given time stage, (i) the decision maker first observes the state of the system x_t, and (ii) decides on u_t *after* observing x_t. (iii) Uncertainty ξ_t is then realized and observed by the decision maker. The probability distribution of the uncertainty may depend on the system state x_t as well as the

decision u_t. (iv) The decision maker incurs a cost, and (v) the system moves to a new state x_{t+1}.

It still remains to clarify the exact objective of the decision maker. In common optimization problems, given a set of input parameters, one seeks an optimal decision vector. By contrast, in the setup presented in figure 3.1, the solution to the problem is not an optimal decision *vector*, but an optimal decision *policy*. The **policy** $\mu_t(x_t)$ is a function for each stage of the problem which maps the reaction of the decision maker to the state x_t of the system. In this sense, the problem being defined is far more complex than a common optimization problem since it is solving over functions rather than vectors of $\mathbb{R}^n$.

Having introduced policies, it is now possible to define in a concrete way the objective of the decision maker. Consider a choice of policies $\mu_t(x_t)$. Then one can define a probability distribution for the sequence $(x_t, \xi_t), t = 0, \ldots, H$, in the sense that the dynamics of the system are the consequence of random realizations of ξ_t and reactions of the decision maker to the new state at which the system arrives, x_t. Therefore, it is also possible to define expected cost as

$$\mathbb{E}[\sum_{t=0}^{H-1} c_t(x_t, \mu_t(x_t), \xi_t) + c_H(x_H)],$$

where the expectation operator is over the distribution of the process $(x_t, \xi_t), t = 0, \ldots, H$, and where it is assumed that the final-period cost is only a function of the final-period state.

The goal of the decision maker will then be to find the best possible policy, in the sense of minimum expected cost. This can be expressed as follows:

$$\begin{aligned}(MP) : \min_{\mu_t} \mathbb{E}[&\sum_{t=0}^{H-1} c_t(x_t, \mu_t(x_t), \xi_t) + c_H(x_H)] \\ &\mu_t(x_t) \in A_t(x_t) \\ &x_{t+1} = f_t(x_t, \mu_t(x_t), \xi_t)\end{aligned}$$

The following set of examples illustrates the application of these concepts.

Example 3.1 Consider a hydrothermal scheduling problem. An electric power system needs to satisfy a certain amount of electricity demand D_t. The system is equipped with two technologies, hydro power and thermal units. Denote p_t as the power generated from thermal units at a marginal cost of C. Denote q_t as the power generated from hydro units, and assume that it is generated at zero cost. Suppose that the system is subject to uncertain rainfall, which is distributed independently from one period to the next and identically according to a normal distribution with mean μ and standard deviation σ. The hydro reservoir can store energy, but can only carry an amount E of energy. The thermal generators can produce up to P units of power per period. Finally, suppose that the system can shed load at a high cost V, and denote l_t as the amount of power that is

curtailed from consumers in stage t. Define the problem in terms of the (MP) formulation of equation (3.1).

Solution

The action vector can be defined as $u_t = (p_t, q_t, l_t)$. The action vector therefore exists in $\mathbb{R}^3$, and is a *continuous* decision. It is necessary to define the state of the vector, which is the information that is required in order to be able to define the transition of the system. For this purpose, it is necessary to introduce a *continuous* state vector x_t, which exists in $\mathbb{R}$, and which denotes the level of the reservoir at stage t. Given the sequence of events implied by figure 3.1, x_t can be specifically defined to denote the energy in the hydro reservoir at the *beginning* of stage t. The action set $A_t(x_t)$ can be defined as

$$\begin{aligned} &A_t(x_t) = \{(p_t, q_t, l_t) : \\ &q_t \leq x_t \\ &p_t + q_t + l_t = D_t \\ &p_t \leq P \\ &p_t, q_t, l_t \geq 0\} \end{aligned}$$

The random disturbance ξ_t of the system can be defined as the rainfall, and exists in $\mathbb{R}$. Note that in contrast to the standard assumptions of the notes, this disturbance is assumed to be *continuous*. The transition probability function of the rainfall can be defined independently of x_t and u_t, as follows:

$$\mathbb{P}[\xi_t \leq R] = \Phi(\frac{R-\mu}{\sigma}), \tag{3.1}$$

where $\Phi(\cdot)$ is the cumulative distribution function of a standard normal random variable. The transition function of the system can be defined as follows, assuming it is possible to throw away water that exceeds the capacity of the hydro dam:

$$x_{t+1} = f_t(x_t, u_t, \xi_t) = \min(E, x_t + \xi_t - q_t).$$

Finally, the cost incurred at each stage can be expressed as follows:

$$c_t(x_t, u_t, \xi_t) = C \cdot p_t + V \cdot l_t.$$

Example 3.2 Consider the same problem as before, except that the rainfall r_t in each period follows an autoregressive process:

$$r_t = c + \phi \cdot r_{t-1} + w_t$$

where c and ϕ are fixed parameters, and w_t is distributed independently over time stages and identically according to a normal distribution with mean μ and standard deviation σ. Adjust the formulation of the previous example in order to model the problem according to equation (3.1).

Solution

First it is necessary to redefine the random disturbance of the system, which is now equal to $\xi_t = w_t$. The random disturbance still lives in $\mathbb{R}$ and follows the distribution function of equation (3.1). In order to accommodate the autoregressive behavior of the rainfall, it is important to observe that the description of the future state of the system requires knowledge of the current level of rainfall, r_t. Denote e_t as the level of energy stored in the reservoir, then the state vector becomes $x_t = (e_t, r_t)^T$ and now lives in $\mathbb{R}^2$. The dynamic function of the system also needs to be redefined, as follows:

$$x_{t+1} = (e_{t+1}, r_{t+1})^T = f_t(x_t, u_t, \xi_t) = \begin{bmatrix} \min(E, x_t + \xi_t - q_t) \\ c + \phi \cdot r_t + \xi_t \end{bmatrix}.$$

Example 3.3 Consider the multi-stage capacity expansion problem presented in section 1.2. The action vector requires two elements. The first element of u_t is the amount of capacity constructed at a given stage, which is denoted here as $z_{it} = (z_{1t}, \ldots, z_{n-1,t})$, and is a vector in $\mathbb{R}^{n-1}$ (assuming that no capacity needs to be constructed for the demand response technology). The second element is the amount of power that is allocated from technology i to block j, which is denoted as $y_t = (y_{11t}, \ldots, y_{1mt}, \ldots, y_{n1t}, \ldots, y_{nmt})$, and is a vector in $\mathbb{R}^{nm}$. The action vector $u_t = (z_t, y_t)$ is therefore continuous, and exists in $\mathbb{R}^{nm+n-1}$. The state vector is also continuous, and includes the amount of capacity that has been constructed so far for each technology, which is denoted as $v_t = (v_{1t}, \ldots, v_{n-1,t})$. Thus, the state vector $x_t = v_t$ exists in $\mathbb{R}^{n-1}$. Assume that the demand of each block $D_t = (D_{1t}, \ldots D_{mt})$ is uncertain, and follows a *discrete* distribution $\mathbb{P}[\cdot]$ which is independent of x_t and u_t. Thus, the uncertain disturbance of the system $\xi_t = D_t$ is an element in $\mathbb{R}^m$. The feasible action set of the system is described as follows:

$$\begin{aligned} A_t(x_t) = \{&(z_t, y_t) : \\ &\sum_{j=1}^{m} y_{ijt} \leq x_{it}, i = 1, \ldots, n-1, \\ &\sum_{i=1}^{n} y_{ijt} = D_j, j = 1, \ldots, m \\ &y_t \geq 0, z_t \geq 0\} \end{aligned}$$

The transition equation of the system is given by

$$x_{i,t+1} = x_{it} + z_{it}, i = 1, \ldots, n-1.$$

The cost incurred at each stage is given by

$$c_t(x_t, u_t, \xi_t) = \sum_{i=1}^{n-1} I_i \cdot z_{it} + \sum_{i=1}^{n} \sum_{j=1}^{m} C_i \cdot T_j \cdot y_{ijt},$$

where I_i is the investment cost of technology i, C_i is the marginal cost of technology i, and T_j is the (deterministic) duration of block j. Note that this model assumes that the capacity built in period t cannot be used for satisfying the demand of that period, but can be used as of period $t+1$.

Example 3.4 Consider the problem of scheduling a machine that produces P units of output when it is on. A cost of C is paid for every period that the machine is on. The output produced by the machine earns a time-varying price λ_t. The machine needs to stay on for at least 3 hours once it is started up. Formulate the problem in the framework of equation (3.1).

Solution

This is an example of a problem without uncertainty, where the state and action set are discrete. The action space can be represented as $\mathbb{A} = \{\text{Stay}, \text{Change}\}$, where 'Stay' corresponds to staying in the existing on/off state of the machine and 'Change' corresponds to moving from on to off or vice versa. The state of the system is the number of hours that have elapsed since the machine was last turned on, with 0 indicating that the machine is off. The state of the system therefore belongs to the set $\mathbb{Z} = \{0, 1, 2, \ldots\}$. The feasible action set can be described as

$$\begin{aligned} &A_t(0) = \{\text{Stay}, \text{Change}\}, \\ &A_t(x_t) = \{\text{Stay}\}, x_t = 1, 2 \\ &A_t(x_t) = \{\text{Stay}, \text{Change}\}, x_t \geq 3 \end{aligned}$$

which represents the fact that (i) if the machine is off, then it can either be kept off or started up, (ii) if the machine has been on for 3 or more hours, then there is the freedom to keep it on or shut it down, and (iii) if the machine has been on for less than three hours, then the machine must stay on. The system transition function can be described as

$$\begin{aligned} x_{t+1} &= f_t(0, \text{Stay}) = 0 \\ x_{t+1} &= f_t(x_t, \text{Stay}) = x_t + 1, x_t \geq 1 \\ x_{t+1} &= f_t(0, \text{Change}) = 1 \\ x_{t+1} &= f_t(x_t, \text{Change}) = 0, x_t \geq 1 \end{aligned}$$

The cost of the system can be expressed as

$$\begin{aligned} &c_t(x_t, u_t) = (C - \lambda_t \cdot P), x_t \geq 1 \\ &c_t(0, u_t) = 0 \end{aligned}$$

Note that the definition of the state and action spaces is non-trivial in this example. For example, defining the action set as the decision to have the machine being on or off at a given period would not work, because it would not be possible to express the dynamics of the system.

Example 3.5 Intraday trading

3.2 The Dynamic Programming Algorithm

The **dynamic programming algorithm** is a very general algorithm that can be used for resolving, among other classes of problems, problems of multi-stage decision making under uncertainty. Whereas most people who are familiar with optimization are most likely to have encountered a problem of optimizing over vectors, optimizing over functions, or policies μ_t, may appear exotic at first sight. The dynamic programming algorithm, which was originally proposed by Bellman, is devised for optimizing over functions, and is based on the very intuitive dynamic programming principle, which will be explained subsequently. The dynamic programming principle allows breaking the problem into single-period optimization problems, and moving backwards from the end of the horizon to the beginning of time, one step at a time, until the optimal policy for each period is obtained.

In order to state the algorithm, it is first necessary to define a key notion for the resolution of the problem, the **value function**. The value function at stage t, $V_t(x_t)$, maps the state of the system at stage t, x_t, to the least expected cost that would be incurred if optimal decisions were to be made from stage t until the end of the optimization horizon, *given* that the state of the system at time t is x_t.

DEFINITION 3.1 The **dynamic programming algorithm** can be stated as follows:

- Starting from $t = H$, for all $x_t \in A_t(x_t)$, compute

$$V_H(x_H) = c_H(x_H).$$

- Moving backwards in time, for all $t = H-1, \ldots, 0$, for all $x_t \in A_t(x_t)$, compute

$$V_t(x_t) = \min_{u_t \in A_t(x_t)} \mathbb{E}_{\xi_t}[(c_t(x_t, u_t, \xi_t) + V_{t+1}(f_{t+1}(x_t, u_t, \xi_t)))|x_t, u_t] \tag{3.2}$$

where the expectation is over the distribution of ξ_t given u_t and x_t.

Note that the expectation is taken over ξ_t, whose distribution can depend on the state vector x_t and the action u_t. If ξ_t is independent of u_t and x_t, the conditional expectation operator is replaced by an expectation operator in the expression above. If ξ_t obeys a Markov model, then x_t must include the previous realization of uncertainty, ξ_{t-1}.

The intuition of the dynamic programming principle is that an optimal policy over a horizon $\{0, \ldots, H\}$ should be such that, if this policy were implemented from stage t until the end of the horizon, then the policy should be optimal when

considering the problem over the interval $\{t, \ldots, H\}$. This is the **principle of optimality**. The definition of the value functions $V_t(x_t)$ allows the decomposition of the multi-stage optimization problem into single-stage optimization problems.

An object which is often used in dynamic programming is the Q **functions**, which represents the cost of selecting action u_t given state x_t:

$$Q_t(x_t, u_t) = \mathbb{E}_{\xi_t}[c_t(x_t, u_t, \xi_t) + V_{t+1}(f_t(x_t, u_t, \xi_t))|x_t, u_t]$$

It is then possible to derive the value function from the Q function:

$$V_t(x_t) = \min_{u_t \in A_t(x_t)} Q_t(x_t, u_t).$$

In order to reason about the complexity of the dynamic programming algorithm, it is useful to consider a discretization of each component of x_t, u_t and ω_t. Suppose that each dimension of the state space, action space and random input space is discretized into d points each, and that the state vector x_t consists of m components, the action vector u_t consists of n components and the random vector ξ_t consists of p components.

At stage t, $V_t(x_t)$ needs to be computed for all d^m possible values of x_t. Each evaluation of the value function requires the computation of an expectation, which requires summing d^p values. Each summand is a minimization over all possible actions, therefore it is necessary to compare d^n values. Thus, each stage of the dynamic programming algorithm requires $O(d^{m+n+p})$ operations, resulting in an overall complexity of $O(H \cdot d^{m+n+p})$. This is prohibitive for problems with large state spaces, action spaces, or random input spaces. The explosion of the problem complexity as a result of increasing the dimensionality of the state, action and random input spaces is referred to as the **curse of dimensionality** in dynamic programming.

The situation is even more discouraging in the case where the state space, action space, or random input space is continuous. The SDDP algorithm which is developed in these notes deals with each of these issues separately: (i) the random disturbances are discretized, and outcomes are simulated, instead of enumerating all possible realizations of uncertainty (ii) the action space, although continuous, can be searched efficiently since in SDDP the problem solved at each stage t in equation (3.2) is a linear program; (iii) the state space is indeed continuous, however it is possible to interpolate the values of the value functions V_t (for reasons that will become clear in the following chapters); moreover, the SDDP algorithm should, in principle, concentrate on the most 'promising' region of the state space and ignore parts of the state space that are not expected to yield low overall cost.

Coming soon: dynamic programming applied to the machine scheduling example.

Coming soon: proof of optimality of dynamic programming algorithm.

3.3 Why Is Dynamic Programming Any Good?

The reason why the dynamic programming algorithm delivers computational savings is because it avoids unnecessary re-computation, by storing data that is used repeatedly in the resolution of a problem, instead of re-computing this data whenever it is required. The following example illustrates the idea in the context of the traveling salesman problem.

Consider an instance of the traveling salesman problem, shown in figure 3.2, whereby a traveling salesman starting from a given city 0 seeks a tour that goes through a set of cities exactly once and returns to 0, while ensuring that the total distance of the tour is minimized. The distance from city i to city j is c_{ij}.

A naive approach of solving the problem considers the full set of tours, and selects the tour of minimum distance. This is shown in table 3.1. For a problem with $H+1$ cities, this requires the examination of $H!$ permutations. The computation of the cost of each permutation requires H summations. The complexity of enumeration is therefore $O(H! \cdot H)$.

Instead, one can interpret each city that is visited as one 'stage' in a multi-stage decision process. In order to proceed with a decision at stage t, it is necessary to know which cities R_t still need to be visited and what the current city i is. This comprises the state of the system, i.e. $x_t = (R_t, i)$. An economy in computation is possible by storing the value $V_t(x_t)$, which represents the most efficient way of visiting the cities in R_t exactly once, starting from i and ending up at 0. The reason that this economy in computation is possible is that $V_t(x_t)$, which is the value function of stage t, can be computed recursively as follows:

$$V_t(R_t, i) = \min_{j \in R_t} c_{ij} + V_{t+1}(R_t - \{j\}, j). \tag{3.3}$$

Thus, the data $V_t(R_t, i)$ is reused. More specifically, the fact that the system state is described as the set of cities that still need to be visited, but without special attention to the exact order that they need to be visited in, is what delivers savings in computation. Note that V_t needs to be computed for H different values of i (since for any $t > 0$ it must be the case that $i \neq 0$), and $\begin{pmatrix} H \\ H-t \end{pmatrix}$ different values of R_t (this is the number of different subsets of cities $\{1, \ldots, H\}$ of size $H-t$). The minimization in equation (3.3) requires the comparison of $H-t$ values, since R_t consists of $H-t$ members. The total number of values that are computed is therefore H for $V_0(\{1, \ldots, H\}, 0)$ for stage 0, and $\sum_{t=1}^{H} H \cdot \begin{pmatrix} H \\ H-t \end{pmatrix} \cdot (H-t)$ for the remaining stages, resulting in a complexity of $O(H^2 \cdot 2^H)$. Although the complexity remains exponential, it is not factorial and therefore performs better than enumeration. The recursive application of equation (3.3) is shown in table 3.2.

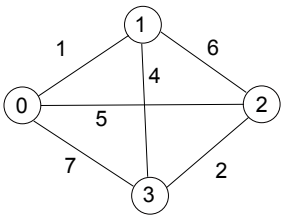


Figure 3.2 An instance of the traveling salesman problem with 4 cities, used in the example of section 3.3. The distance between cities is indicated on the arcs. An enumeration of all possible tours that begin from and end in city 0 is shown in table 3.1.

Table 3.1 An enumeration of all possible tours and their distance in the example of figure 3.2. Each tour can be reversed, which is why there are 6 tours but only 3 different costs. The shortest tour is 01320, or equivalently its reverse tour, 02310.

Tour	Distance
01320	12
02310	12
01230	16
03210	16
02130	22
03120	22

3.4 Examples

The following examples demonstrate applications of the dynamic programming algorithm in determinsitic and stochastic problems with discrete sets of actions. This should serve as a way to reinforce the intuition of the reader before moving to the more abstract setting of continuous action and state spaces.

3.4.1 The Knapsack Problem

Consider the knapsack problem, where the goal is to select out of a set of items which ones should be included in a knapsack of finite capacity so as to maximize benefit. Mathematically,

$$
\begin{aligned}
& p^\star = \max_x \sum_{i=1}^{n} v_i x_i \\
& \sum_{i=1}^{n} w_i x_i \leq W \\
& x_i \in \{0,1\} \quad i = 1, \ldots, n,
\end{aligned}
$$

Table 3.2 An application of the dynamic programming algorithm to the problem of figure 3.2. $V_t(R_t, i)$ is the cheapest way of starting from i, covering the cities in R_t, and ending up in 0.

Value function	Evaluation
$V_3(\emptyset, 1)$	1
$V_3(\emptyset, 2)$	5
$V_3(\emptyset, 3)$	7
$V_2(\{2\}, 1) = c_{12} + V_3(\emptyset, 2)$	11
$V_2(\{3\}, 1)$	11
$V_2(\{1\}, 2)$	7
$V_2(\{3\}, 2)$	9
$V_2(\{1\}, 3)$	5
$V_2(\{2\}, 3)$	7
$V_1(\{2,3\}, 1) = \min\{c_{12} + V_2(\{3\}, 2), c_{13} + V_2(\{2\}, 3)\}$	11
$V_1(\{1,3\}, 2)$	7
$V_1(\{1,2\}, 3)$	9
$V_0(\{1,2,3\}, 0) = \min_{x \in \{1,2,3\}}(c_{0x} + V_1(\{1,2,3\} - \{x\}, x))$	12

where v_i is the benefit of item i, w_i is its volume, W is the capacity of the knapsack and x_i indicates whether item i is chosen or not. Assume that all data are integer. Also assume that $w_i > 0$ for all $i = 1, \ldots, n$.

Suppose that summing two numbers requires one unit of computation time, and taking the minimum of two numbers requires negligible computation time. Then the computation of $\sum_{i=1}^{n} w_i x_i$ requires $n-1$ summations, and the same is true for $\sum_{i=1}^{n} v_i x_i$. In order to solve the problem by complete enumeration, there are 2^n combinations to test, hence a total run time of $2^n \cdot (2n-2)$. Therefore the run time of complete enumeration is $O(n \cdot 2^n)$.

In order to implement a dynamic programming algorithm for solving the problem, start by defining $V(i, w)$, where V is defined in $\{0, \ldots, , n\} \times \{0, \ldots, W\}$, as the best attainable value for a knapsack with capacity w and to which items from the set $\{1, \ldots, i\}$ can be inserted.

By definition of V, the boundary values for $V(0, w)$ and $V(i, 0)$ are $V(0, w) = 0$ and $V(i, 0) = 0$. These equalities can be interpreted as follows: if there are no items to include in the knapsack, then no value van be assigned. If there is no capacity in the knapsack, no items can be included.

The dynamic programming principle of optimality requires that

$$V(i, w) = \max\{V(i-1, w - w_i) + v_i, V(i-1, w)\},$$

where the first case in the maximization corresponds to including the item in the knapsack, and the second case in the maximization corresponds to not including the item in the knapsack.

The full algorithm for solving the problem can then be expressed as

Table 3.3 The data for the knapsack problem.

i	1	2	3	4
v_i	10	40	30	50
w_i	5	4	6	3

```
for i = 1 : n
  for w = 1 : W

    V(i, w) = max{V(i − 1, w − −w_i) + v_i, V(i − 1, w)};

  end
end
```

Each evaluation of $V(i, w)$ requires 2 time units (because w_i is subtracted from w, and v_i is added to the value function). This is repeated $n \cdot W$ times, therefore the total number of required operations is $2 \cdot n \cdot W$ operations. The optimal objective function value of the problem is then derived from the value function as $p^\star = V(n, W)$.

Since the function V does not indicate which items should be included in the knapsack, the following routine can be used for determining these items. Define $K(i)$ as an indicator function of whether item i should be included in the knapsack. Then $K(i)$ can be computed as follows:

```
w ← W;
for i = n : 1 do
  K(i) ← 0;
  if v_i + V(i − 1, w − w_i) ≥ V(i − 1, w) then
    K(i) ← 1;
    w ← w − w_i;
  end if
end for
```

Each iteration requires 3 operations, the total run time is therefore $3 \cdot n$.

Applying this algorithm to an example with a knapsack of weight $W = 7$ and the items given in table 3.3, the resulting value function is shown in table 3.4.

In order to determine the items that should be included, start from $n = 4$. Note that $v_4 + V(3, 7 - w_4) = 50 + V(3, 4) = 50 + 40 \geq V(4, 7) = 90$. Therefore, item 4 should be included. For $n = 3$, one obtains $v_3 + V(2, 3 - w_3) = 30 + V(2, -3) = -\infty < V(3, 4)$, so item 3 should not be included. Proceeding analogously, item 2 should be included, and item 1 should not be included.

Table 3.4 The value function of the knapsack problem of section 3.4.1, with entry (i, j) corresponding to $V(i, j)$.

	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	10	10	10
2	0	0	0	0	40	40	40	40
3	0	0	0	0	40	40	40	40
4	0	0	0	50	50	50	50	90

3.4.2 The Shortest Path Problem

3.4.3 The Monty Hall Problem

Consider the Monty Hall problem: a player participates in a TV game show in which (s)he is presented with three curtains. Behind two of the curtains there is a goat, behind the third curtain there is a sports car. The game proceeds as follows: (i) the player is asked to pick a curtain; (ii) the host opens up a curtain with a goat behind it; (iii) the player can keep the curtain that (s)he chose originally, or switch to the remaining curtain; (iv) the player keeps the content behind the curtain that was selected in step (iii). The Monty Hall problem can be stated as follows: should the player change curtains in step (iii), or not? An important assumption that can save a lot of confusion and allows a formal exposition of the argument is that when the host opens a curtain in step (ii), the host will choose curtains with equal likelihood if both of the curtains not chosen by the player hide a goat.

TODO: generalize argument for the case where the host chooses curtain 2 with probability α.

The problem can be cast as a three-stage decision making problem under uncertainty. Note that since the curtains are completely symmetrical in the beginning of the game, one can assume without loss of generality that curtain 1 is chosen in the first stage. The decision process of the problem is then presented in figure 3.3. The payoff of the player is normalized to equal $+1$ in the case that the player receives the sports car, and 0 in case the player receives the goat. Note that the realization of uncertainty is *not* described in terms of the curtain that the sports car has been assigned to, because this cannot be observed by the player. Instead, uncertainty is described by what the player can observe, which is the curtain that is opened by the host.

The tricky aspect in solving the problem is to compute the transition probabilities correctly. For the transition probability of the first stage, relying on the symmetry of the problem gives an equal probability of revealing curtains 2 and 3. Once the player selects to stay or switch in the second stage, it is necessary to compute the transition probabilities of winning (sports car) or losing (goat). Bayes' theorem can now be applied as follows:

$$\begin{aligned}
&\mathbb{P}[\text{Car in C1}|\text{Host shows C2}] = \\
&\frac{\mathbb{P}[\text{Car in C1, Host shows C2}]}{\mathbb{P}[\text{Host shows C2}]} = \\
&\frac{\mathbb{P}[\text{Host shows C2}|\text{Car in C1}] \cdot \mathbb{P}[\text{Car in C1}]}{\mathbb{P}[\text{Host shows C2}]} = \\
&\frac{1/2 \cdot 1/3}{1/2} = \frac{1}{3},
\end{aligned}$$

So the probability of winning if the player remains with the original choice is 1/3, which is equal to the original probability that the player wins the sports car if the game terminates in stage 1. On the other hand,

$$\begin{aligned}
&\mathbb{P}[\text{Car in C3}|\text{Host shows C2}] = \\
&\frac{\mathbb{P}[\text{Car in C3, Host shows C2}]}{\mathbb{P}[\text{Host shows C2}]} = \\
&\frac{\mathbb{P}[\text{Host shows C2}|\text{Car in C3}] \cdot \mathbb{P}[\text{Car in C3}]}{\mathbb{P}[\text{Host shows C2}]} = \\
&\frac{1 \cdot 1/3}{2/3} = \frac{2}{3},
\end{aligned}$$

which implies that the chances of winning double if the player switches curtains! An intuitive way to understand the above result is that the fact that the host deliberately leaves one door unrevealed strongly increases the chances of this door carrying the prize.

One might be tempted to argue on an intuitive basis that switching gives a 50/50 chance of winning, because it is like picking from two doors (the remaining ones), either of which may have the car. This intuition is incorrect.

The expected payoffs of each decision are shown in the tree of figure 3.3 in bold font above the circles which indicate realizations of randomness, and the optimal decision of each stage are underlined. The probability of the player winning the sports car, if the optimal strategy is chosen, is 2/3, as indicated in the expected payoff of the first stage. Decision trees such as the one of figure 3.3 offer a graphical way of applying Bellman's dynamic programming algorithm for problems with small numbers of actions and a small number of realizations of uncertainty.

3.4.4 Pricing Financial Derivatives

An *American call option* is a financial instrument that allows its owner to buy a certain asset (for example, a stock) at a *strike price* at or before a certain *expiration date*. Since the holder of the call option can exercise the option and immediately turn around and sell the asset to the market, the call option at time

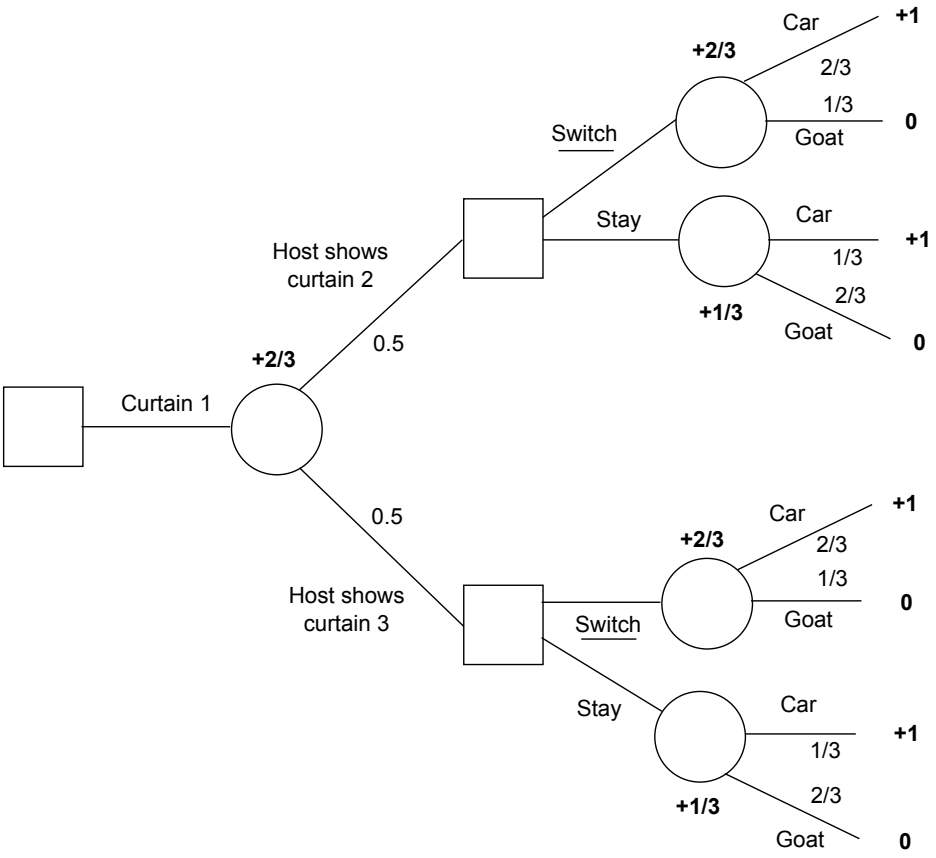


Figure 3.3 The decision process for the Monty Hall problem. Squares represent decisions, circles represent a realization of uncertainty. The payoffs in the end of the game are indicated in bold font.

t is worth $\max(S_t - k, 0)$, where S_t is the price of the asset at time t and k is the strike price of the option. In this example dynamic programming is used in order to determine how much an agent who is maximizing expected payoff would be willing to pay at time $t = 0$ for an American call option with expiration at $H = 5$ and strike price $k = 60$.

As in the Monty Hall example of figure 3.3, the idea is to start from period 5 and move backwards. At each node of the lattice of figure 3.4 one needs to decide between exercising the option and being paid $\max(S_t - k, 0)$, or holding the option and waiting for the next stage. Denote $V_t(i)$ as the value of the stock at stage t, and state i, where i corresponds to one of the nodes in the lattice of

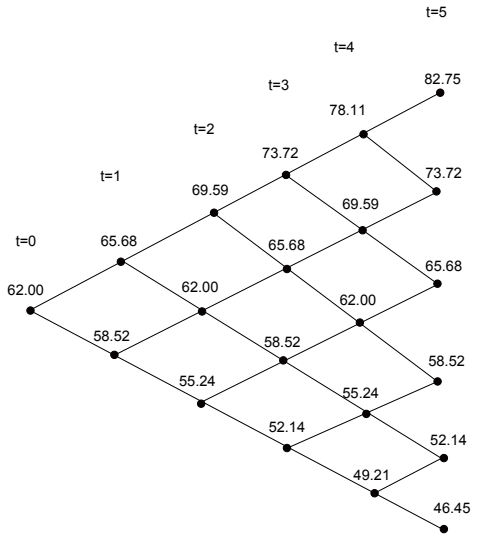


Figure 3.4 The lattice model of the stock of the example in section 3.4.4. The upward transition probability at each stage is $q = 0.5577$, the downward transition probability is $1 - q$.

figure 3.4. One then obtains

$$
\begin{aligned}
&V_5(1) = 82.75 - 60 = 22.75\\
&V_5(2) = 73.72 - 60 = 13.72\\
&V_5(3) = 65.68 - 60 = 5.68\\
&V_5(4) = 0\\
&V_5(5) = 0\\
&V_5(6) = 0\\
&V_4(1) = \max(78.11 - 60, \mathbb{E}[V_5(j)|i = 1]) = 18.7560\\
&V_4(2) = \max(69.59 - 60, \mathbb{E}[V_5(j)|i = 2]) = 10.1639\\
&V_4(3) = \max(62 - 60, \mathbb{E}[V_5(j)|i = 3]) = 3.1677\\
&V_4(4) = 0\\
&V_4(5) = 0\\
&V_3(1) = \max(73.72 - 60, \mathbb{E}[V_4(j)|i = 1]) = 14.9557\\
&V_3(2) = \max(65.68 - 60, \mathbb{E}[V_4(j)|i = 2]) = 7.0695\\
&V_3(3) = \max(0, \mathbb{E}[V_4(j)|i = 3]) = 1.7666\\
&V_3(4) = 0\\
&V_2(1) = \max(69.59 - 60, \mathbb{E}[V_3(j)|i = 1]) = 11.4676\\
&V_2(2) = \max(62 - 60, \mathbb{E}[V_3(j)|i = 2]) = 4.7240\\
&V_2(3) = 0.9852\\
&V_1(1) = \max(65.68 - 60, \mathbb{E}[V_2(j)|i = 1]) = 8.4849\\
&V_1(2) = \max(0, \mathbb{E}[V_2(j)|i = 2]) = 3.0703\\
&V_0(1) = \max(62 - 60, \mathbb{E}[V_1(j)|i = 1]) = 6.09
\end{aligned}
$$

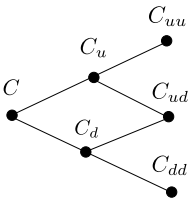


Figure 3.5 The three-stage lattice used in question 3.2.

Now suppose that the lattice model of the stock described in figure 3.4 obeys the following pattern: given the stock price S_t at a given stage, an upward (respectively downward) transition in the next stage results in stock price $u \cdot S_t$ with $u > 1$ (respectively $d \cdot S_t$ with $d < 1$), and the probability q of an upward transition is given by

$$q = \frac{1-d}{u-d},$$

which is chosen so that the expected value of the asset in stage $t+1$ equals its value in stage t. Then it is possible to use the lattice of figure 3.5 to show that $C_u \geq u \cdot S - k$ and $C_d \geq d \cdot S - k$, where C_u (respectively C_d) is the value of waiting in stage 2 in case of an increase (respectively decrease) in stock price and S is the price of the stock at stage 1.

To see this, and referring to figure 3.5, note that the value of waiting at stage 1, denoted C_u, is

$$\begin{aligned} C_u &= qC_{uu} + (1-q)C_{ud} \\ &\geq q\max(u^2S - k, 0) + (1-q)\max(udS - k, 0) \\ &\geq \max(q(u^2S - k) + (1-q)(udS - k), 0) \\ &= \max(uS - k, 0) \\ &\geq uS - k \end{aligned}$$

where the first inequality follows from the fact that the expected payoff of the option is at least as much as exercising the option in stage 3, the second inequality follows from the convexity of $\max(x, 0)$, and the equality after that follows after simple algebra given the definition of q. The same reasoning applies for showing that $C_d \geq dS - k$.

One concludes that for the given lattice model of stock evolution, it is preferable to hold the option at stage 2 rather than exercising it. By induction, it can be shown that if the underlying asset behaves according to this specific model, then it is optimal to hold on to an American call option until its expiration.

Problems

3.1 Describe a dynamic programming algorithm that solves the traveling salesman problem moving forward, instead of backwards.

Bibliography

Section 3.1. The exposition is based on (Bertsekas 1995).
Section 3.2. The exposition is based on (Bertsekas 1995).
Section 3.4. The asset pricing example is based on (Luenberger 1998).

4 Stochastic Linear Programming

This chapter addresses the modeling of uncertainty in linear programs and provides a number of motivating examples that are used in the sequel. The representation of uncertainty receives particular attention since it affects the development of algorithms for resolving these problems. Section 4.1 introduces two-stage stochastic linear programs. When moving to multi-stage models, it is useful to provide graphical representations of uncertainty. This is achieved through scenario trees and lattices, which are presented in section 4.2. The property of serial independence, which is exploited by SDDP, is also introduced in this section. Once the modeling of uncertainty has been clarified, multi-stage stochastic linear programs are introduced in section 4.3. Section 4.4 shows how to apply the dynamic programming algorithm, which is introduced in chapter 3, to multi-stage stochastic linear programs.

4.1 Two-Stage Stochastic Linear Programs

Stochastic linear programs are linear programs with uncertain data. The element that makes such optimization problems challenging to solve is **recourse**: the ability to react after uncertainty has been revealed to the decision maker.

The typical structure of two-stage stochastic linear programs involves a set of

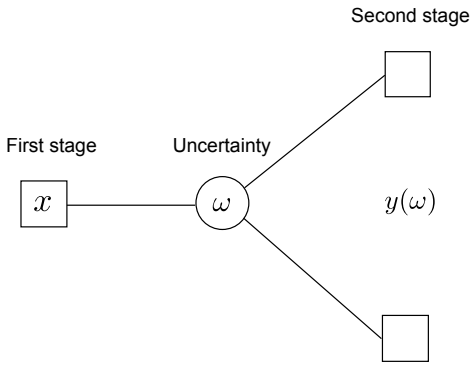


Figure 4.1 Sequence of events in two-stage stochastic linear programs.

first-stage decisions that need to be fixed prior to the revelation of uncertainty. Once uncertainty is revealed, second-stage decisions are determined, based on the realization of uncertainty. The sequence of events is presented in figure 4.1.

Consider a probability space $(\Omega, \mathcal{B}(\Omega), \mathbb{P})$. Two-stage stochastic linear programs can then be formulated as follows:

$$\begin{aligned} &\min c^T x + \mathbb{E}[\min q(\omega)^T y(\omega)] \\ &\text{s.t. } Ax = b \\ &T(\omega)x + W(\omega)y(\omega) = h(\omega) \\ &x \geq 0, y(\omega) \geq 0 \end{aligned}$$

First-stage decisions are denoted as $x \in \mathbb{R}^{n_1}$. For a given realization ω, second-stage decisions are denoted as $y(\omega) \in \mathbb{R}^{n_2}$. The deterministic parameters of the first stage are $c \in \mathbb{R}^{n_1}$, $b \in \mathbb{R}^{m_1}$, $A \in \mathbb{R}^{m_1 \times n_1}$. Second-stage data are $q(\omega) \in \mathbb{R}^{n_2}$, $h(\omega) \in \mathbb{R}^{m_2}$, $T(\omega) \in \mathbb{R}^{m_2 \times n_1}$ and $W(\omega) \in \mathbb{R}^{m_2 \times n_2}$.

Stochastic linear programs obey **fixed recourse** when W does not depend on ω. Fixed recourse permits the use of specific algorithms for the efficient resolution of stochastic linear programs.

Example 4.1 A classical example of two-stage stochastic programming is the newsboy problem. Consider the production planning problem whereby a product can be produced at time 1, and can be sold at time 2. However, the demand at time 2 is uncertain. Denote x as the amount of the product that is produced in period 1, s as the amount that is sold in period 2. C will denote the unit cost of production, P the sale price. The random demand is denoted as $D(\omega)$. The newsboy problem can then be written as follows:

$$\begin{aligned} &\min_{x, s(\omega) \geq 0} C \cdot x - \mathbb{E}[P \cdot s(\omega)] \\ &\text{s.t. } s(\omega) \leq x \\ &s(\omega) \leq D(\omega) \end{aligned}$$

The trade-off in this problem is that procuring too much of the product in the first stage may lead to excessive costs since there is no guarantee that the procured quantity will be sold, whereas if too little quantity is procured then there is the possibility of missing out on revenues in the second stage in case demand is high. Variants of this problem can be considered with salvage costs for excess inventory or a penalty that depends on the amount of unsatisfied demand.

Example 4.2 Consider the capacity expansion planning problem that was introduced in chapter 1. It was explained in section 1.2 that uncertainty in the second stage can be represented by adding blocks to the load duration curve.

Alternatively, the problem can be written as a stochastic program:

$$\begin{aligned}
&\min_{x,y\geq 0} \sum_{i=1}^{n}(I_i \cdot x_i + \mathbb{E}[\sum_{j=1}^{m} C_i \cdot T_j \cdot y_{ij}(\omega)]) \\
&\text{s.t.} \sum_{i=1}^{n} y_{ij}(\omega) = D_j(\omega), j = 1, \ldots, m \\
&\sum_{j=1}^{m} y_{ij}(\omega) \leq x_i, i = 1, \ldots n-1
\end{aligned}$$

where I_i, C_i represent the fixed and variable cost of technology i respectively, $D_j(\omega), T_j$ represent the height and width of load block j respectively, x_i represents the amount of investment in technology i, and $y_{ij}(\omega)$ represents the capacity of technology i allocated to block j for realization ω. Note that T_j is not indexed by ω. The solution of the problem is illustrated graphically in figure 4.2.

Example 4.3 The problem of hydro-thermal scheduling can be described as follows: consider an electric power system with a hydroelectric dam and a thermal power plant. The hydroelectric dam can store a limited amount of water, which can then be used to produce free electricity. However, the amount of rainfall that occurs at each period is random. On the other hand, the thermal power plant can produce electricity whose output is costly but can be controlled. At each time step, a known demand needs to be satisfied. Consider the hydrothermal scheduling problem over two time periods. In order to describe the problem as a mathematical program, denote by q_t the amount of power that is drawn from the dam and is converted into electricity. Denote by p_t the amount of power produced by the thermal power plant. The marginal cost of the thermal power plant is denoted by C, and the demand that needs to be satisfied at each time step is denoted by D_t. E represents the maximum amount of hydro power that can be stored in the dam. Denote x_t as the content of the dam at the *end* of a stage, and by r_t the amount of rain during stage t (the amount of rain in stage 1 is assumed to be non-random). Considering this notation, the problem can be stated as follows:

$$\begin{aligned}
&\min C \cdot p_1 + \mathbb{E}[C \cdot p_2(\omega)] \\
&p_1 + q_1 \geq D_1 \\
&x_1 \leq x_0 + r_1 - q_1 \\
&x_1 \leq E \\
&p_2(\omega) + q_2(\omega) \geq D_2 \\
&q_2(\omega) \leq x_1 + r_2(\omega) \\
&p_1, q_1, x_1, p_2(\omega), q_2(\omega) \geq 0
\end{aligned}$$

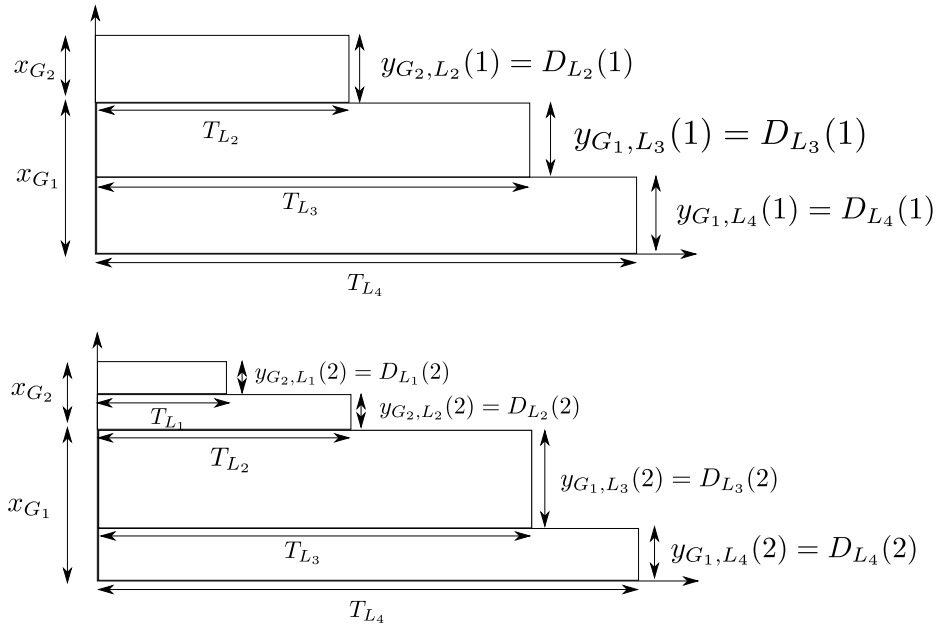


Figure 4.2 A graphical illustration of the solution of the capacity expansion problem of example 4.2. Note that the duration of each block remains constant. The first scenario corresponds to three blocks (block 1 is assigned zero demand), whereas the second scenario corresponds to four blocks. Note that the capacity decision x cannot depend on the scenario ω. The parameter $D_j(\omega)$ corresponds to the height of each block.

4.2 Scenario Trees, Lattices, and Serial Independence

Before proceeding with the introduction of multi-stage stochastic linear programs, it is important to clarify the model of uncertainty that is employed in MSLPs, and compare it to the more general model of uncertainty which was introduced in section 3. Recall that for the more general framework of chapter 3, uncertainty depends on the actions of the decision maker at every stage. For this reason, in Markov decision processes the sample space Ω depends on actions. This framework is unnecessarily general for MSLPs. Instead, in MSLPs we define a set S_t at each stage of the problem, and the sample space Ω as $\Omega = S_1 \times \ldots \times S_H$, where H is the horizon of the optimization. We then define a probability space $(\Omega, \mathcal{A}_H, \mathbb{P})$, where $\mathcal{A}_H = 2^{\Omega}$ is the set of events that can be distinguished by the decision maker at stage H, and $\mathbb{P}$ is a probability measure on these events. Traditionally, the literature in stochastic programming starts counting time at

$t = 1$, therefore the notation departs slightly from the notation used in the previous chapter (where time starts at $t = 0$), without any implications whatsoever in the development of the key ideas.

In practice, probabilistic models in MSLPs are rarely if ever defined by specifying the probability space $(\Omega, \mathcal{A}_H, \mathbb{P})$, because this is too tedious and often unnecessarily complex. Instead, more concise representations of uncertainty are typically employed.

Since we are focused on Markov processes $\{\xi_t\}_{t\in\mathbb{Z}}$ in these notes, we can represent them using **scenario trees**. A tree (N, E) is an acyclic connected graph, where N denotes its nodes and E denotes its edges. A rooted tree is a special type of tree where there exists a unique node which is designated as the *root*. A scenario tree is a rooted tree which is used in MSLP for describing uncertainty in the model. The nodes of a scenario tree represent histories of realizations of the random input $\xi_{[t]} = (\xi_1, \ldots, \xi_t)$, with each node characterized by a unique time index. The edges represent transition probabilities (typically non-zero) from a history $\xi_{[t]} \in \Xi_{[t]}$ in stage t to a history $\xi_{[t+1]} \in \Xi_{[t+1]}$ in stage $t + 1$. Here, Ξ_t corresponds to the support of ξ_t, and $\Xi_{[t]} = \Xi_1 \times \ldots \times \Xi_t$. The root of the tree corresponds to the first time step of the problem, $t = 1$. The **ancestor** of a node $\xi_{[t]}$, denoted $A(\xi_{[t]})$, is the *unique* adjacent node which precedes $\xi_{[t]}$, i.e. $A(\xi_{[t]}) = \{\xi_{[t-1]} : (\xi_{[t-1]}, \xi_{[t]}) \in E\}$. The **children** or **descendants** of a node, denoted as $C(\xi_{[t]})$, correspond to the set of nodes that are adjacent to $\xi_{[t]}$ and are indexed by a subsequent time stage, i.e. $C(\xi_{[t]}) = \{\xi_{[t+1]} : (\xi_{[t]}, \xi_{[t+1]}) \in E\}$.

A scenario tree implicitly defines a probability space $(\Omega, \mathcal{B}(\Omega), \mathbb{P})$ in a much more compact way than an explicit description of the probability space. In order to fully specify this probability space, it is sufficient to assign (i) for each node $\xi_{[t]} \in N$, a value for the random vector $\xi_{[t]}$, and (ii) for each edge, $(\xi_t, \xi_{t+1}) \in E$, a transition probability $\mathbb{P}[\xi_{[t+1]}|\xi_{[t]}]$.

A **lattice** is an alternative way to represent a Markov process. In a lattice, nodes are used to represent the realization ξ_t of a process, as opposed to its history $\xi_{[t]}$. Thus, a lattice consists of (i) a set of nodes, each of which is characterized by a time stage, and each of which corresponds to a realization of ξ_t, and (ii) a set of edges connecting *all* nodes in stage t to *all* nodes in stage $t+1$, corresponding to transition probabilities. An example of a lattice is provided in figure 4.4.

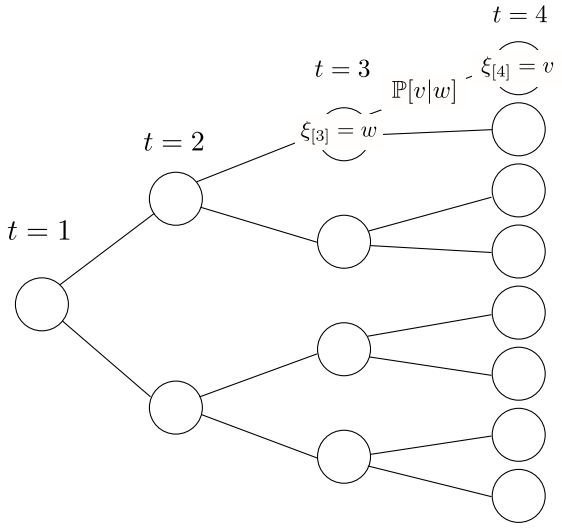
PROPOSITION 4.1 *Lattices and scenario trees are equivalent ways of representing a discrete time Markov process with discrete values.*

Proof Needs to be filled out. The idea is illustrated graphically in figure 4.5. □

The SDDP algorithm is typically described in terms of uncertainty that is represented on a lattice. In SDDP, there is a further assumption on the structure of uncertainty which is typically employed. This is the assumption of **serial independence**.

DEFINITION 4.2 The stochastic process $\{\xi_t\}$ satisfies serial independence if in

Figure 4.3 A graphical representation of a scenario tree. Each node $\xi_{[t]} \in N$ is associated with a history of realizations of the stochastic input, $\xi_{[t]}$, and a probability of realization. Each edge $(\xi_{[t]}, \xi_{[t+1]}) \in E$ is associated with a transition probability $\mathbb{P}[\xi_{[t+1]}|\xi_{[t]}]$.



each stage t it has a probability distribution that does not depend on the history of the process, i.e. one can define a probability measure $p_t(i)$ at each stage t, such that

$$\mathbb{P}[\xi_t = i|\xi_{[t-1]}] = p_t(i), \forall \xi_{[t-1]} \in \Xi_{[t-1]}, i \in \Xi_t, t = 2, \ldots, H.$$

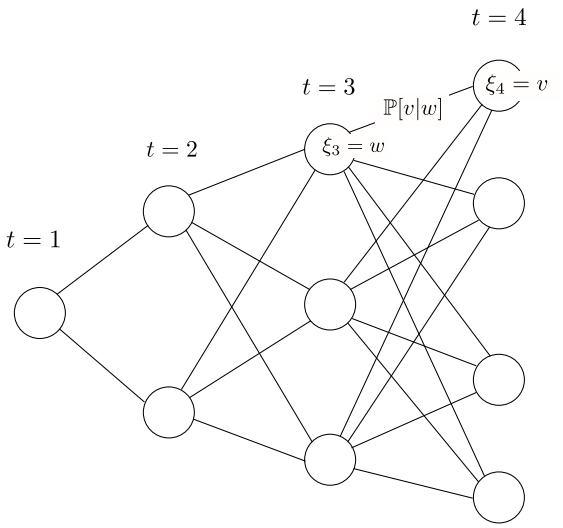
A process ξ_t which obeys serial independence can be represented using a lattice, with each node of the lattice associated with a probability of realization (which is, in general, not the case for lattices without serial independence).

Example 4.4 Consider the scenario trees of figure 4.4. The first scenario tree does not obey serial independence because the value of $\xi_3(\omega)$ depends on whether $\xi_2(\omega) = 1$ or $\xi_2(\omega) = 2$. The second scenario tree does not obey serial independence because the transition probabilities from stage 2 to stage 3 depend on whether $\xi_2(\omega) = 1$ or $\xi_2(\omega) = 2$. The third scenario tree is serially independent.

In order to fully specify a scenario tree or a lattice, it is necessary to describe the values of the process $\{\xi_t\}$, as well as the probability measure of the associated space. This can be done as follows:

- For scenario trees, one specifies:

Figure 4.4 A graphical representation of a lattice. The dashed line box marks the set Ξ_t. Each node $\xi_t \in N$ is associated with a realization of the stochastic input. Each edge $(\xi_t, \xi_{t+1}) \in E$ is associated with a transition probability $\mathbb{P}[\xi_{t+1}|\xi_t]$.

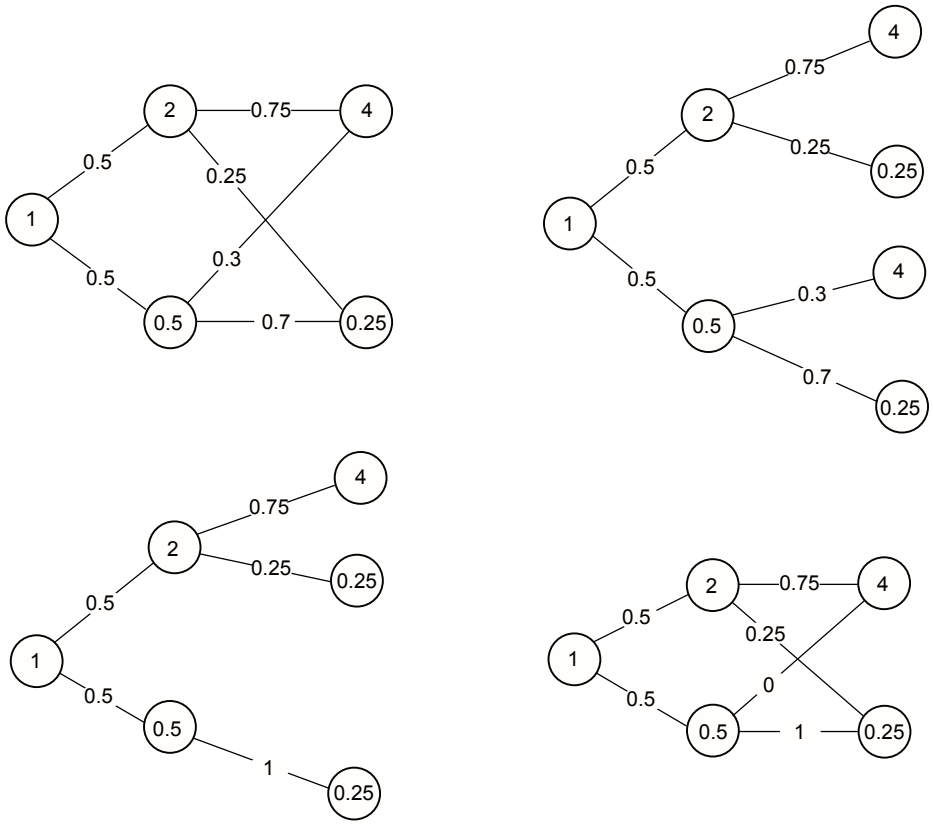


  - The value of ξ_t at each node of the scenario tree. This implies a unique history $\xi_{[t]}$ for each node of the scenario tree.
  - The transition probability for every edge of the scenario tree.
- For lattices, one specifies:
  - The value of ξ_t at each node of the lattice. Note that in contrast to a lattice, this node is not necessarily associated with a unique history leading up to the given node.
  - The transition probability for every edge of the lattice.
- For lattices with stage-wise independence, one specifies:
  - The value of ξ_t at each node of the lattice.
  - The probability of realization of each node of the lattice, which is a well-defined quantity.

4.3 Multi-Stage Stochastic Linear Programs

Multi-stage stochastic linear programs generalize the two-stage linear programs that are introduced in the previous section over multiple periods. Consider a finite set of outcomes Ω, and the probability space $(\Omega, 2^{\Omega}, \mathbb{P})$. Further consider a filtration $\{\mathcal{A}_t\}_{t\in\mathbb{Z}}$. The following is referred to as the **extended formulation**

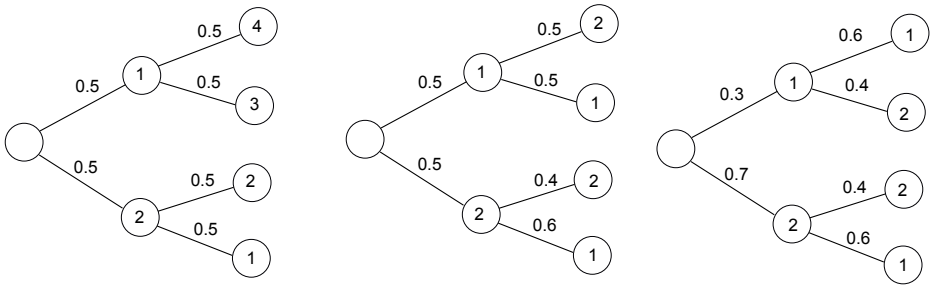
Figure 4.5 An illustration of the equivalence of scenario trees and lattices. The top part of the figure represents the transformation of a lattice to a scenario tree. The bottom part of the figure represents the transformation of a scenario tree to a lattice. Numbers within nodes correspond to realizations of ξ_t. Numbers on edges correspond to transition probabilities.



of a stochastic program:

$$
\begin{aligned}
&(MSLP):\\
&\min c_1^T x_1 + \mathbb{E}[c_2(\omega)^T x_2(\omega) + \cdots + c_H(\omega)^T x_H(\omega)]\\
&\text{s.t. } W_1 x_1 = h_1\\
&T_1(\omega)x_1 + W_2(\omega)x_2(\omega) = h_2(\omega), \omega \in \Omega\\
&\vdots\\
&T_{t-1}(\omega)x_{t-1}(\omega) + W_t(\omega)x_t(\omega) = h_t(\omega), \omega \in \Omega\\
&\vdots\\
&T_{H-1}(\omega)x_{H-1}(\omega) + W_H(\omega)x_H(\omega) = h_H(\omega), \omega \in \Omega\\
&x_1 \geq 0, x_t(\omega) \geq 0, t = 2, \ldots, H
\end{aligned}
$$

Figure 4.6 The scenario trees of example 4.4. The values in nodes represent the realization $\xi_t(\omega)$. The values on the arcs represent transition probabilities.



where the random input[1] $\xi_t^T(\omega) = (c_t^T(\omega), h_t(\omega)^T, vec(T_{t-1}(\omega)), vec(W_t(\omega))]$ and the decision vector $x_t(\omega)$ are adapted to $\mathcal{A}_t$. In this model, $c_t(\omega) \in \mathbb{R}^{n_t}$ is a vector of cost coefficients, $h_t(\omega) \in \mathbb{R}^{m_t}$ is a vector of right-hand-side coefficients, $W_t(\omega) \in \mathbb{R}^{m_t \times n_t}$ corresponds to the coefficients of $x_t(\omega)$ at stage t and $T_{t-1}(\omega) \in \mathbb{R}^{m_t \times n_{t-1}}$ corresponds to the coefficients of $x_{t-1}(\omega)$. Note that $x_t(\omega)$ is a stochastic process, which is being optimized within the model. The requirement that x_t and ξ_t be adapted to $\mathcal{A}_t$ is an implicit way of requiring non-anticipativity, meaning that we cannot react to future outcomes that have not yet been revealed to us.

We now specialize the formulation above to the case of a scenario tree. We will use the notation $\omega_t \in S_t$ and $\omega_{[t]} \in S_1 \times \ldots \times S_t$, therefore ω_t can be interpreted as an index in the set Ξ_t, which is the support of the random vector ξ_t, while $\omega_{[t]}$ can be interpreted as an index in the set $\Xi_{[t]} = \Xi_1 \times \ldots \times \Xi_t$, which is the support for the process corresponding to the history of realizations, up to period t. One can then formulate a multi-stage stochastic linear program on a scenario

[1] The operator $vec(\cdot)$ applied to a matrix expands the columns of the matrix to a vector, i.e. $vec(A) = (A_{\cdot,1}^T, \ldots, A_{\cdot,n}^T)$, where n is the number of columns of A and $A_{\cdot,j}$ is the j-th column of the matrix.

tree as follows:

$$
\begin{aligned}
&(MSLP-ST):\\
&\min c_1^T x_1 + \mathbb{E}[c_2(\omega_{[2]})^T x_2(\omega_{[2]}) + \cdots + c_H(\omega_{[H]})^T x_H(\omega_{[H]})]\\
&\text{s.t. } W_1 x_1 = h_1\\
&T_1(\omega_{[2]})x_1 + W_2(\omega_{[2]})x_2(\omega_{[2]}) = h_2(\omega_{[2]}), \omega_{[2]} \in S_1 \times S_2\\
&\vdots\\
&T_{t-1}(\omega_{[t]})x_{t-1}(\omega_{[t-1]}) + W_t(\omega_{[t]})x_t(\omega_{[t]}) = h_t(\omega_{[t]}), \omega_{[t]} \in S_1 \times \ldots \times S_t\\
&\vdots\\
&T_{H-1}(\omega_{[H]})x_{H-1}(\omega_{[H-1]}) + W_H(\omega_{[H]})x_H(\omega_{[H]}) = h_H(\omega_{[H]}), \omega_{[H]} \in S_1 \times \ldots \times S_H\\
&x_1 \geq 0, x_t(\omega_{[t]}) \geq 0, t = 2, \ldots, H
\end{aligned}
$$

Note that the random parameters of the problem, $\xi_t(\omega_{[t]})$, and decisions, $x_t(\omega_{[t]})$, are a function of the full history of uncertainty up to period t. Effectively, $\omega_{[t]}$ is indexing the nodes of the scenario tree.

When uncertainty is formulated on a lattice, the random vector ξ_t can be indexed over the nodes of the lattice, $\omega_t \in S_t$, instead of the full history $\omega_{[t]} \in S_1 \times \ldots \times S_t$. Although this does not limit the size of the formulation, compared to (MSLP-ST), it does facilitate the description of the dynamic programming algorithm subsequently. The problem is described as follows:

$$
\begin{aligned}
&(MSLP-L):\\
&\min c_1^T x_1 + \mathbb{E}[c_2(\omega_t)^T x_2(\omega_{[2]}) + \cdots + c_H(\omega_H)^T x_H(\omega_{[H]})]\\
&\text{s.t. } W_1 x_1 = h_1\\
&T_1(\omega_2)x_1 + W_2(\omega_2)x_2(\omega_{[2]}) = h_2(\omega_2), \omega_{[2]} \in S_1 \times S_2\\
&\vdots\\
&T_{H-1}(\omega_H)x_{H-1}(\omega_{[H]}) + W_H(\omega_H)x_H(\omega_{[H]}) = h_H(\omega_H), \omega_{[H]} \in S_1 \times \ldots \times S_H\\
&x_1 \geq 0, x_t(\omega_{[t]}) \geq 0, t = 2, \ldots, H
\end{aligned}
$$

In the above formulation, it is understood that $\omega_{[t]} = (\omega_1, \ldots, \omega_t) \in S_1 \times \ldots \times S_t$ implies $\omega_t \in S_t$. Note that the random disturbance ξ_t is now indexed over $\omega_t \in S_t$, while the decision vector x_t is indexed over the history $\omega_{[t]} \in S_1 \times \ldots \times S_t$.

Example 4.5 Consider a 3-stage model of the capacity expansion planning problem where uncertainty can be described by a lattice. The lattice and the corresponding scenario tree are shown in figure 4.7. Two realizations of net load are considered at each stage, a reference realization with probability 10% and a 10x wind realization with a probability 90%. The load duration curve of each realization is repeated in table 4.1, the technology options are repeated in table

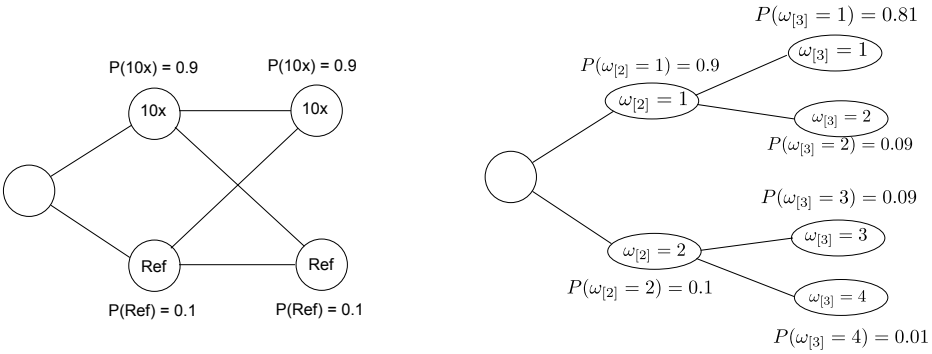


Figure 4.7 Left: the lattice of the multi-stage capacity expansion planning problem, example 4.5. Right: the expansion of the lattice to its corresponding scenario tree, with probabilities of arriving at each node indicated at the respective nodes. For example $\omega_{[2]} = 1$ corresponds to the realization of the 10x wind outcome in period 2 and occurs with a probability of 0.9, whereas $\omega_{[3]} = 2$ corresponds to the realization of the 10x wind outcome in period 2 and the reference wind outcome in period 3, and occurs with probability 0.09.

Table 4.1 Two possible realizations of the load duration curve at each stage, example 4.5.

	Duration (hours)	Level (MW) Reference scenario	Level (MW) 10x wind scenario
Base load	8760	0-7086	0-3919
Medium load	7000	7086-9004	3919-7329
Peak load	1500	9004-11169	7329-10315

4.2. For conciseness of notation, ω is depicted as an index of decision variables

Table 4.2 The set of options for serving demand, example 4.5.

Technology	Fuel cost ($/MWh)	Inv cost ($/MWh)
Coal	25	16
Gas	80	5
Nuclear	6.5	32
Oil	160	2
DR	1000	0

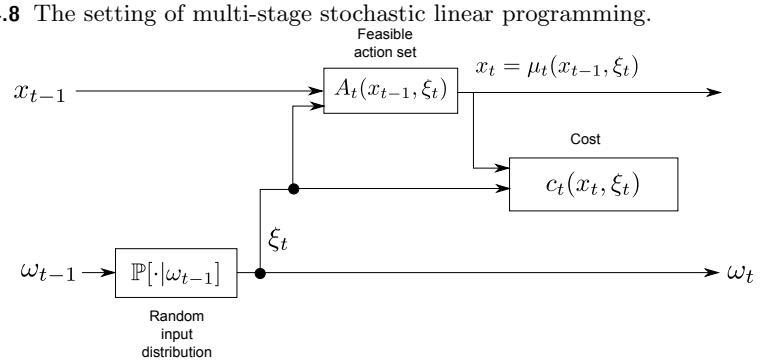
(i.e. $x(\omega)$ is replaced by x_ω).

$$
\begin{aligned}
&\min \sum_{i=1}^{n} I_i \cdot v_{i11} \\
&+ \sum_{\omega_{[2]}=1}^{2} p_{\omega_{[2]}} \Big(\sum_{i=1}^{n} I_i \cdot v_{i2\omega_{[2]}} + \sum_{i=1}^{n}\sum_{j=1}^{m} C_i \cdot T_j \cdot y_{ij2\omega_{[2]}}\Big) \\
&+ \sum_{\omega_{[3]}=1}^{4} p_{\omega_{[3]}} \Big(\sum_{i=1}^{n} I_i \cdot v_{i3\omega_{[3]}} + \sum_{i=1}^{n}\sum_{j=1}^{m} C_i \cdot T_j \cdot y_{ij3\omega_{[3]}}\Big) \\
&\text{s.t. } x_{i11} = v_{i11}, i \in \{1,\dots,n-1\} && (4.1)\\
&\sum_{i=1}^{n} y_{ijt\omega_{[t]}} = D_{jt\omega_{[t]}}, j \in \{1,\dots,m\}, t \in \{2,\dots,3\} \\
&\omega_{[2]} \in \{1,2\}, \omega_{[3]} \in \{1,\dots,4\} && (4.2)\\
&x_{i2\omega_{[2]}} = x_{i11} + v_{i2\omega_{[2]}}, i \in \{1,\dots,n-1\}, \omega_{[2]} \in \{1,2\} \\
&x_{i3\omega_{[3]}} = x_{i21} + v_{i3\omega_{[3]}}, i \in \{1,\dots,n-1\}, \omega_{[3]} \in \{1,2\} \\
&x_{i3\omega_{[3]}} = x_{i22} + v_{i3\omega_{[3]}}, i \in \{1,\dots,n-1\}, \omega_{[3]} \in \{3,4\} \\
&\sum_{j=1}^{m} y_{ij2\omega_{[2]}} \le x_{i11}, i \in \{1,\dots n-1\}, \omega_{[2]} \in \{1,2\} && (4.3)\\
&\sum_{j=1}^{m} y_{ij3\omega_{[3]}} \le x_{i21}, i \in \{1,\dots n-1\}, \omega_{[3]} \in \{1,2\} && (4.4)\\
&\sum_{j=1}^{m} y_{ij3\omega_{[3]}} \le x_{i22}, i \in \{1,\dots n-1\}, \omega_{[3]} \in \{3,4\} && (4.5)\\
&x, v, y \ge 0
\end{aligned}
$$

$v_{it\omega_{[t]}}$ is the amount of capacity of technology i which is constructed in period t for the path corresponding to history $\omega_{[t]}$. Similarly, $x_{it\omega_{[t]}}$ is the *total* amount of capacity of technology i which is available in period t for the path corresponding to history $\omega_{[t]}$. The notation $y_{ijt\omega_{[t]}}$ follows the same logic. Note the sequence of events in a given time stage t: demand $D_{jt\omega_{[t]}}$ is observed but can only be satisfied with the capacity $x_{i,t-1,\omega_{[t-1]}}$ which is available at the *end* of the previous time stage, and after demand has been satisfied it is possible to construct new capacity $v_{it\omega_{[t]}}$. Thus, the first stage only involves an investment decision, and no production decision. For the same reason, the first-stage cost only consists of investment costs and constraints (4.2), (4.3)-(4.5) are only enforced in stages 2 and 3. Note that constraint (4.1) is necessary since without it the construction of capacity would be free in stage 1.

In this example, the production at each stage y does not influence the future state of the system, hence the problem obeys a structured referred to as block separability, which is defined in the next section. The optimal solution of the problem is to construct 2986 MW of coal, 7329 MW of nuclear and 854 MW

Figure 4.8 The setting of multi-stage stochastic linear programming.



of oil in period 1, without adding any more capacity in future periods. Note that since the capacity constructed in period 1 is sufficient to cover the highest possible realization of demand in stage 2, and since the scenarios of stage 3 are identical to those of scenario 2, there is no reason to construct additional capacity in stage 2. It is interesting to compare the solution obtained from this 3-stage model to the solution of the 2-stage model in table 1.5, which constructs 5085 MW of coal, 1311 MW of gas, 3919 MW of nuclear and 854 MW of oil. The 3-stage model could be described equivalently as a deterministic model with 6 demand blocks, and solved using screening curves, as described in section 1.2.

4.4 Applying Dynamic Programming to Stochastic Linear Programs

Multistage stochastic linear programs constitute a special class of the (MP) problem that was introduced in chapter 3. Therefore, the dynamic programming algorithm can be used for resolving this class of problems. The goal of this section is to map the notation used in stochastic dynamic programming to the notation used in stochastic linear programs defined on lattices, which is the typical context for applying SDDP.

The setting of MSLPs on lattices is presented in figure 4.8, and should be compared to figure 3.1. There are a number of differences between the two block diagrams that should be pointed out:

- The decision maker first observes the realization of uncertainty, ξ_t, and then decides x_t, instead of deciding before observing ξ_t.
- The system state which is propagated from one period to the next consists not only of the vector x_t, but also of the node in the lattice, ω_t, in which the decision maker finds itself.
- The feasible action set A_t depends not only on the vector x_{t-1}, but also on the realization of uncertainty in the current stage, ξ_t.

The feasible action set in stage t is encoded by the constraints of the linear program:

$$T_{t-1}(\omega_t)x_{t-1}(\omega_{[t]}) + W_t(\omega_t)x_t(\omega_{[t]}) = h_t(\omega_t), \omega_{[t]} \in S_1 \times \ldots \times S_t. \tag{4.6}$$

Note that there is no explicit distinction of action u_t and state x_t in the setup of figure 4.8, and consequently there are no explicit system dynamical equations either. Such a distinction can be made explicit for MSLPs which obey **block separability**. Block separability occurs when the decision vector can be partitioned into a set of state variables x_t and a set of action variables u_t, and when the constraint matrix of equation (4.6) can be written in the following form:

$$\begin{aligned} T^{xx}_{t-1}(\omega_t)x_{t-1}(\omega_{[t-1]}) + W^{xx}_t(\omega_t)x_t(\omega_{[t]}) &= h^{xx}_t(\omega_t), \omega_{[t]} \in S_1 \times \ldots \times S_t \\ T^{xu}_{t-1}(\omega_t)x_{t-1}(\omega_{[t-1]}) + W^{xu}_t(\omega_t)u_t(\omega_t) &= h^{xu}_t(\omega_t), \omega_{[t]} \in S_1 \times \ldots \times S_t \end{aligned}$$

In this case, the decision variables u_t do not need to be included in the state vector, because, although they influence the current stage cost, they do not influence the evolution of the system.

Figure 4.8 separates the node of the lattice at a given stage, ω_t, from the corresponding realization of uncertainty, ξ_t. This is done in order to emphasize that the information that needs to be propagated forward in the system state is (x_t, ω_t), not (x_t, ξ_t). Therefore, even if the dimensionality of ξ_t may be very large, all information about the influence of ξ_t on future cost can be collapsed into ω_t, the node of the lattice in which the system finds itself in the current time stage. Note that the system state is described by (x_t, ω_t), i.e. the node of the lattice at which the system finds itself is part of the system state.

Consider, now, how the dynamic programming algorithm can be applied in the context of figure 4.8. In the final period of the horizon, one defines the following Q**-function**:

$$\begin{aligned} Q_H(x_{H-1}, \xi_H) = \min_{x_H} \; & c_H(\omega_H)^T x_H \\ \text{s.t. } & T_{H-1}(\omega_H)x_{H-1} + W_H(\omega_H)x_H = h_H(\omega_H) \\ & x_H \geq 0 \end{aligned}$$

This should be compared to the Q function definition in stochastic optimal control (chapter 3), the domain of which is the cross product of state x and action u (as opposed to state x and random input ξ).

The **value function** for the final period is then defined as

$$V_H(x_{H-1}, \omega_{H-1}) = \mathbb{E}_{\xi_H}[Q_H(x_{H-1}, \xi_H)|\omega_{H-1}].$$

Proceeding recursively, at a given time step t the Q-function which represents the cost of being in a given state x_t given that ξ_t will occur in stage t, is computed

as:

$$\begin{aligned} Q_t(x_{t-1}, \xi_t) = \min_{x_t} \; & c_t(\omega_t)^T x_t + V_{t+1}(x_t, \omega_t) \\ & \text{s.t. } T_{t-1}(\omega_t) x_{t-1} + W_t(\omega_t) x_t = h_t(\omega_t) \\ & x_t \geq 0 \end{aligned}$$

The value function of stage t is then obtained as

$$V_t(x_{t-1}, \omega_{t-1}) = \mathbb{E}_{\xi_t}[Q_t(x_{t-1}, \xi_t) | \omega_{t-1}].$$

The solution of the original problem is then obtained by solving the following first-stage problem:

$$\begin{aligned} & \min c_1^T x_1 + V_2(x_1) \\ & \text{s.t. } W_1 x_1 = h_1 \\ & x_1 \geq 0 \end{aligned}$$

Note that this notation is not general, but specifically tailored to MSLPs defined on lattices, which will be resolved by SDDP in later chapters. In particular, the notation $V_t(x_{t-1}, \omega_{t-1})$ indicates that the value function depends on a system state which includes the vector x_t that appears in constraints (4.6) *and* the node of the lattice where the system finds itself in. Instead, the notation $Q_t(x_{t-1}, \xi_t)$ emphasizes the dependence of the Q functions on the actual values of the random vector ξ_t. The SDDP algorithm literally stores value functions as data structures indexed by ω_t, whereas the Q-functions are not stored explicitly by the algorithm, therefore the notation for Q_t is kept in line with the general notation used in stochastic programming, whereas the notation for V_t emphasizes the way that value functions are stored in the lattice of the SDDP algorithm.

To further clarify this notational convention, the state vector x_t of chapter 3 would include all the necessary information for determining the conditional distribution of ξ_t, including the node of the lattice in which the system finds itself (recall that the probability distribution of ξ_t in chapter 3 depends on u_t and x_t, i.e. it is defined as $\mathbb{P}[\cdot | x_t, u_t]$). By contrast, the notation in this section is adapted to MSLPs on lattices, and is therefore intended to represent how data is stored by the SDDP algorithm. This choice of notation is a compromise between the notation used in cutting plane algorithms and the notation used in stochastic control.

Example 4.6 Consider again the hydrothermal scheduling problem of example 3.1. Consider the specific case where there is a demand of 1000 MW in the system at each period. Demand is either served by hydroelectric power, at zero marginal cost, or by thermal generators, whose marginal cost is equal to 25 \$/MWh. Thermal generators have a technical capacity of 500 MW. The cost of not serving demand amounts to 1000 \$/MWh. Rainfall in each period is described by *independent*, identically distributed random variables that follow a uniform

distribution over the interval [0, 1000] MW. Denote $f : \mathbb{R} \to \mathbb{R}$ as the probability distribution function of the rainfall. The Q function for the third period can be computed as follows:

$$\begin{aligned} Q_3(x_2, R_3) = \min\ & 1000 \cdot l + 25 \cdot p \\ \text{s.t. } & l + p + q \geq 1000 \\ & p \leq 500 \\ & q \leq x_2 + R_3(\omega) \\ & l, p, q \geq 0 \end{aligned}$$

where p corresponds to thermal production, q corresponds to hydro production, l corresponds to unserved demand, and x_2 corresponds to the amount of stored hydro energy at the beginning of period 2. Q_3 can be expressed in closed form as follows:

$$Q_3(x_2, R_3) = \begin{cases} 0, x_2 + R_3(\omega) \geq 1000 \\ 25 \cdot (1000 - (x_2 + R_3(\omega))), 500 \leq x_2 + R_3(\omega) < 1000 \\ 500 \cdot 25 + 1000 \cdot (500 - (x_2 + R_3(\omega))), 0 \leq x_2 + R_3(\omega) < 500 \end{cases}$$

The value function of period 3 can then be computed as follows:

$$\begin{aligned} & V_3(x_2) = \mathbb{E}_{R_3}[Q_3(x_2, R_3)] \\ & = \mathbb{P}[R_3(\omega) \geq 1000 - x_2] \cdot 0 \\ & + \int_{r=500-x_2}^{1000-x_2} (25 \cdot (1000 - r - x_2)) f(r) dr \\ & + \int_{r=0}^{500-x_2} (500 \cdot 25 + 1000 \cdot (500 - r - x_2)) f(r) dr \\ & = \begin{cases} 0, & x_2 \geq 1000 \\ 12500 - 25 \cdot x_2 + 0.0125 \cdot x_2^2, & 500 \leq x_2 < 1000 \\ 134375 - 512.5 \cdot x_2 + 0.5 \cdot x_2^2, & 0 \leq x_2 < 500 \end{cases} \end{aligned}$$

Note that V_3 is a convex function of x_2, but *not* a piecewise linear function of x_2. Instead, it will be shown that value functions are piecewise linear when the set of uncertain realizations is finite. The dynamic programming algorithm proceeds by solving the following linear program for deriving Q_2:

$$\begin{aligned} & Q_2(x_1, R_2) = \min 1000 \cdot l + p + V_3(x_2) \\ & \text{s.t. } l + p + q \geq 1000, p \leq 500 \\ & x_2 = x_1 - q + R_2(\omega) \\ & l, p, q, x_2 \geq 0 \end{aligned}$$

Once Q_2 is computed, V_2 can be evaluated and used in the first stage optimization. However, note that the value function expressions become increasingly complex moving backwards, rendering a closed-form solution impossible in most cases.

Problems

4.1 Show that a lattice can be represented by a scenario tree, and vice versa.
4.2 Implement the newsboy problem of example 4.1 in the FAST toolbox with the scenario lattice of example **??**.
4.3 Consider how the model of example 4.3 should be rewritten if x_t denotes the amount of power available in the hydroelectric dam at the beginning of period t (as opposed to the end of period t).
4.4 Implement the model of example 4.5 in a mathematical programming language and verify the correctness of the solution.
4.5 Write out a two-stage stochastic program for modeling the Monty Hall problem.
4.6 Provide a closed-form solution for the newsboy problem with salvage cost.

Bibliography

Chapter 4.3. The notation for multi-stage stochastic linear programming follows (Birge & Louveaux 2010).

5 Cutting Plane Methods

The previous chapters introduce stochastic programming and relate this class of models to a more general framework of optimization under uncertainty that relies on dynamic programming. One evident disadvantage of stochastic programs is that their size grows rapidly as a function of the optimization horizon. From the present chapter forward, the text proceeds to computational methods for tackling the problem.

All algorithms presented in this chapter fall under the general classification of cutting plane methods, which are generally defined as optimization methods which are based on the idea of iteratively refining the objective function or set of feasible constraints of a problem through linear inequalities. The most general variant of a cutting plane method in convex non-differentiable optimization is **Kelley's cutting plane algorithm**, which is used for solving the following optimization problem:

$$\begin{aligned} z^\star = \min\, & c^T x + F(x) \\ \text{s.t. } & x \in X \end{aligned}$$

where X is a compact convex subset of $\mathbb{R}^n$, $F : \mathbb{R}^n \to \mathbb{R}$ is a convex function, and $c \in \mathbb{R}^n$ is a parameter vector. The idea of Kelley's cutting plane algorithm is to build a globally lower bounding function of $F(x)$ at each iteration of the algorithm, which is denoted $L_k : \mathbb{R}^n \to \mathbb{R}$ at the k-th iteration. A record of a lower and upper bound of $z^\star$ is recorded at each iteration of the algorithm. The lower bound at iteration k is denoted L_k, the upper bound is denoted U_k.

DEFINITION 5.1 (Kelley's cutting plane algorithm) Kelley's cutting plane algorithm can be stated as follows:

Step 0: Set $k = 0$, and assume $x_1 \in X$ given. Set $L_0(x) = -\infty$ for all $x \in X$, $U_0 = c^T x_1 + F(x_1)$, and $L_0 = -\infty$.

Step 1: Set $k = k + 1$. Find $a_k \in \mathbb{R}$ and $b_k \in \mathbb{R}^n$ such that

$$\begin{aligned} & F(x_k) = a_k + b_k^T x_k \\ & F(x) \geq a_k + b_k^T x, x \in X \end{aligned}$$

Step 2: Set

$$U_k = \min(U_{k-1}, c^T x_k + F(x_k))$$

and

$$L_k(x) = \max(L_{k-1}(x), a_k + b_k^T x), x \in X$$

Step 3: Compute

$$L_k = \min_{x \in X} c^T x + L_k(x)$$

and denote x_k as the optimal solution of this problem.

Step 4: If $U_k - L_k = 0$, stop. Else, repeat from step 1.

Note that the algorithm involves finding a supporting hyperplane of $F(x)$ in step 1, and solving a linear optimization problem in step 3. The Benders decomposition algorithm which is developed section 5.1 is a specific method for obtaining the supporting hyperplane of step 1 in the case where the function $F(x)$ can be described as the value function of a second-stage linear program. The L-shaped method of section 5.2 is a specific instance of the Benders decomposition algorithm where the second-stage linear program is decomposable into a set of scenarios. The multi-cut L-shaped method of section 5.3 is an alternative to the L-shaped method which generates a tighter under-approximation at step 1 of each iteration.

5.1 Benders Decomposition for Deterministic Linear Programs

Consider a problem of the form

$$\begin{aligned} z^\star = \min\ & c^T x + q^T y \\ & Ax = b \\ & Tx + Wy = h \\ & x, y \geq 0 \end{aligned}$$

where $x \in \mathbb{R}^{n_1}$ and $y \in \mathbb{R}^{n_2}$ represents a set of decision variables. The parameters of the problem are given by $c \in \mathbb{R}^{n_1}$, $b \in \mathbb{R}^{m_1}$, $A \in \mathbb{R}^{m_1 \times n_1}$, $q \in \mathbb{R}^{n_2}$, $h \in \mathbb{R}^{m_2}$, $T \in \mathbb{R}^{m_2 \times n_1}$ and $W \in \mathbb{R}^{m_2 \times n_2}$. We will assume throughout that $z^\star$ is bounded.

Note that the structure has a natural two-stage interpretation. The vector x can be interpreted as first-stage decisions, with $Ax = b$ corresponding to first-stage constraints. The vector y represents second-stage decisions, and $Tx+Wy = h$ represents second-stage constraints that depend on what has been decided in the first stage.

Benders decomposition is an algorithm that is suited for solving this problem whenever (i) the set of second-stage constraints $Tx + Wy = h$ makes the problem excessively difficult to solve, whereas (ii) if the second-stage constraints are ignored then the problem is relatively easy to solve, and (iii) if the first-stage decisions are fixed, then the problem is relatively easy to solve. The idea of the algorithm is to relax the second-stage constraints, and to gradually account for

their impact on the optimization problem by adding terms in the objective function of the problem that reflect the second-stage cost of a given first-stage trial decision.

In order to describe the algorithm, as in the case of dynamic programming, it is helpful to define a **value function** $V : \mathbb{R}^{n_1} \to \mathbb{R}$. The value function maps the first-stage decision x to the best possible performance that can be attained in the second stage, and is described mathematically as follows:

$$\begin{aligned}(S) : V(x) = \min_{y} \; & q^T y \\ & Wy = h - Tx \\ & y \geq 0\end{aligned}$$

It is possible that the first-stage decision results in a second stage problem (S) which is infeasible. In that case, $V(x) = +\infty$. These decisions obviously cannot be optimal, and it will therefore be required that $x \in \text{dom } V$.

The dual of (S) can be expressed as:

$$\begin{aligned}(D) : \max_{\pi} \; & \pi^T (h - Tx) \\ & \pi^T W \leq q^T\end{aligned}$$

The idea of Benders decomposition, and more broadly dynamic programming, is to replace the solution of the original problem with the following equivalent problem:

$$\begin{aligned}& \min c^T x + V(x) \\ & Ax = b \\ & x \in \text{dom } V \\ & x \geq 0\end{aligned}$$

The benefit of focusing on this problem is that the second-stage constraints that complicate the problem have been dropped. However, the challenge with solving this problem is that neither $V(x)$ nor dom V are typically available. A crucial property stemming from the structure of the problem is that $V(x)$ is a piecewise linear convex function and, given a trial decision x, the value function can be approximated exactly at x.

PROPOSITION 5.2 *$V(x)$ is a piecewise linear convex function of x. If π_0 denotes the dual optimal multipliers of (S) given x_0, then*

$$\pi_0^T (h - Tx)$$

is a supporting hyperplane of $V(x)$ at x_0.

Proof From proposition 2.23 it follows that $\pi_0^T (h - Tx)$ is a supporting hyperplane of $V(x)$ at x_0. Since (S) has a finite number of dual optimal multipliers, only a finite number of supporting hyperplanes exist for $V(x)$, therefore

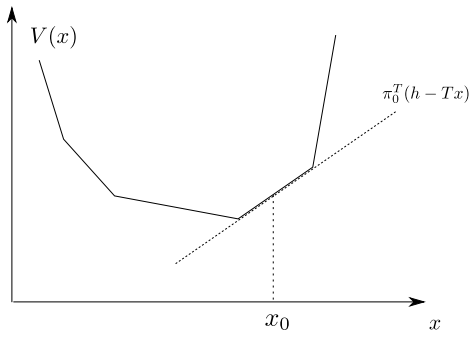


Figure 5.1 A graphical illustration of proposition 5.2.

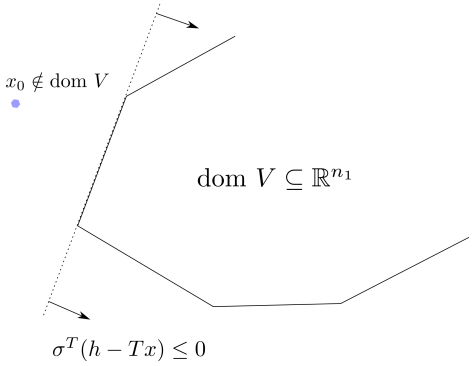


Figure 5.2 A graphical illustration of proposition 5.3.

the function must be piecewise linear convex. The proposition is demonstrated graphically in figure 5.1. □

In addition, the set dom V can be expressed as a set of linear inequalities.

PROPOSITION 5.3 *The domain of V can be expressed equivalently as follows:*

$$dom\ V = \{\sigma^T(h - Tx) \leq 0, \sigma \in R\},$$

where R is the set of extreme rays of the polyhedron $\{\pi : \pi^T W \leq q^T\} \subseteq \mathbb{R}^m$.

Proof Suppose that $x \in \text{dom } V$ and $\sigma^T(h - Tx) > 0$ for some extreme ray σ. Since σ is an extreme ray of $\{\pi : \pi^T W \leq q^T\}$, it follows that $\sigma^T W \leq 0$. Consider any dual feasible vector π_0 (which is known to exist since by assumption $x \in \text{dom } V$ hence the problem (S) is finite-valued for the given choice of x and therefore the dual problem is feasible and finite-valued). Then $\pi_0 + \lambda\sigma$ is feasible for any $\lambda \geq 0$ and since $\sigma^T(h - Tx) > 0$ the dual problem becomes unbounded. This contradicts the assumption that $x \in \text{dom } V$, hence it must be the case that $\sigma^T(h - Tx) \leq 0$ for all $\sigma \in R$. Therefore, dom $V \subseteq \{\sigma^T(h - Tx) \leq 0, \sigma \in R\}$.

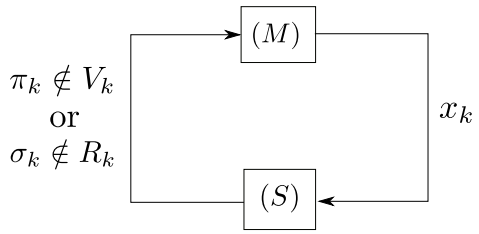


Figure 5.3 The exchange of information between the master and slave problem in Benders decomposition.

Now consider a vector x such that $\sigma^T(h - Tx) \leq 0$ for all $\sigma \in R$. Any ray of the set $\pi^T W \leq q^T$ can be expressed as a convex combination of its extreme rays. Therefore, for any ray γ of $\pi^T W \leq q^T$ it follows that $\gamma^T(h - Tx) \leq 0$ which implies that the dual problem cannot become unbounded. Thus, the vector x must be in dom V. It follows that $\{\sigma^T(h - Tx) \leq 0, \sigma \in R\} \subseteq \text{dom } V$.

□

The preceding proposition is depicted graphically in figure 5.2.

The two previous propositions imply that the original problem can be expressed equivalently as:

$$\begin{aligned}
&\min c^T x + \theta \\
&Ax = b \\
&\sigma^T(h - Tx) \leq 0, \sigma \in R \\
&\theta \geq \pi^T(h - Tx), \pi \in V \\
&x \geq 0
\end{aligned}$$

where θ is a free auxiliary variable, V is the set of vertices of $\{\pi : \pi^T W \leq q^T\}$, and R is the set of extreme rays of $\{\pi : \pi^T W \leq q^T\}$.

This problem can be relaxed by ignoring a large number of the inequalities that define $V(x)$ and dom V. The resulting relaxed problem can be expressed as follows:

$$\begin{aligned}
(M) : z_k = &\min c^T x + \theta \\
&Ax = b \\
&\sigma^T(h - Tx) \leq 0, \sigma \in R_k \subseteq R \\
&\theta \geq \pi^T(h - Tx), \pi \in V_k \subseteq V \\
&x \geq 0
\end{aligned}$$

(M) is referred to as the *master program*. The constraints $\theta \geq \pi^T(h - Tx)$ are referred to as *optimality cuts*, whereas the constraints $\sigma^T(h - Tx)$ are referred to as *feasibility cuts*. The resolution of the master program provides (i) a lower

bound z_k to the optimal objective function value $z^\star$, (ii) a candidate solution x_k that can be fed to (S), which is referred to as the *slave program*, and (iii) an estimate θ_k of $V(x_k)$, such that[1] $\theta_k \leq V(x_k)$. The resolution of (S) with input x_k yields (i) an upper bound $c^T x_k + q^T y_{k+1}$ to the optimal objective function value $z^\star$, (ii) a new vertex π_{k+1} or a new extreme ray σ_{k+1}. The process is depicted in figure 5.3. The index k is as an iteration counter. The master problem (M) depends on the existing set of extreme vertices V_k and rays R_k which is available at iteration k. Over each iteration that is depicted in figure 5.3, one of these sets is augmented by information from the slave problem (S). Thus, $V_k \subseteq V_{k+1}$ and $R_k \subseteq R_{k+1}$. The following proposition demonstrates that either the obtained vertex or extreme ray must add new information to the master problem, or the optimal solution has been found.

PROPOSITION 5.4 *Suppose that x_k is obtained by solving (M) and used as input in (S).*

- *Suppose (S) is feasible and denote π_{k+1} as the optimal vertex that is obtained from (S). If $\pi_{k+1} \in V_k$ then $z^\star = c^T x_k + V(x_k)$ (i.e. x_k is an optimal solution).*
- *Suppose (S) is infeasible and denote σ_{k+1} as the extreme ray that is obtained from (S). Then $\sigma_{k+1} \notin R_k$.*

Proof For any x feasible, $c^T x + V(x) \geq c^T x_k + \theta_k$ because (M) is a relaxation of the original problem. If it can be shown that $\theta_k = V(x_k)$, then it follows that x_k is optimal since then it would be the case that for any x feasible, $c^T x + V(x) \geq c^T x_k + V(x_k)$. From the first inequality of the proof, it is obviously the case that $\theta_k \leq V(x_k)$. It therefore needs to be shown that $\theta_k \geq V(x_k)$. By proposition 5.2, $V(x_k) = \pi_{k+1}^T (h - T x_k)$. Moreover, if $\pi_v \in V_k$ then $\theta_k \geq \pi_v^T (h - T x_k)$ because the constraint $\theta \geq \pi_v^T (h - Tx)$ is enforced in (M) at iteration k. Therefore, if $\pi_{k+1} \in V_k$ then $\theta_k \geq \pi_{k+1}^T (h - T x_k) = V(x_k)$, and the first part of the proposition follows.

If σ_{k+1} is an extreme ray, it must be the case that $\sigma_{k+1}^T (h - T x_k) > 0$. Therefore, it must be the case that $\sigma_{k+1} \notin R_k$, otherwise the point x_k obtained from (M) would have to obey the inequality $\sigma_{k+1}^T (h - T x_k) \leq 0$ by the definition of (M). □

Summarizing all of these observations, the Benders decomposition algorithm is described as follows:

DEFINITION 5.5 Benders decomposition algorithm.
Step 0: Set $k = 0$, $V_0 = R_0 = \emptyset$.
Step 1: Solve (M).

- If (M) is feasible, store x_k.

[1] To see why $\theta_k \leq V(x_k)$, note that if $\theta_k > V(x_k)$ then a contradiction would arise because $V_k \subseteq V$ and $R_k \subseteq R$.

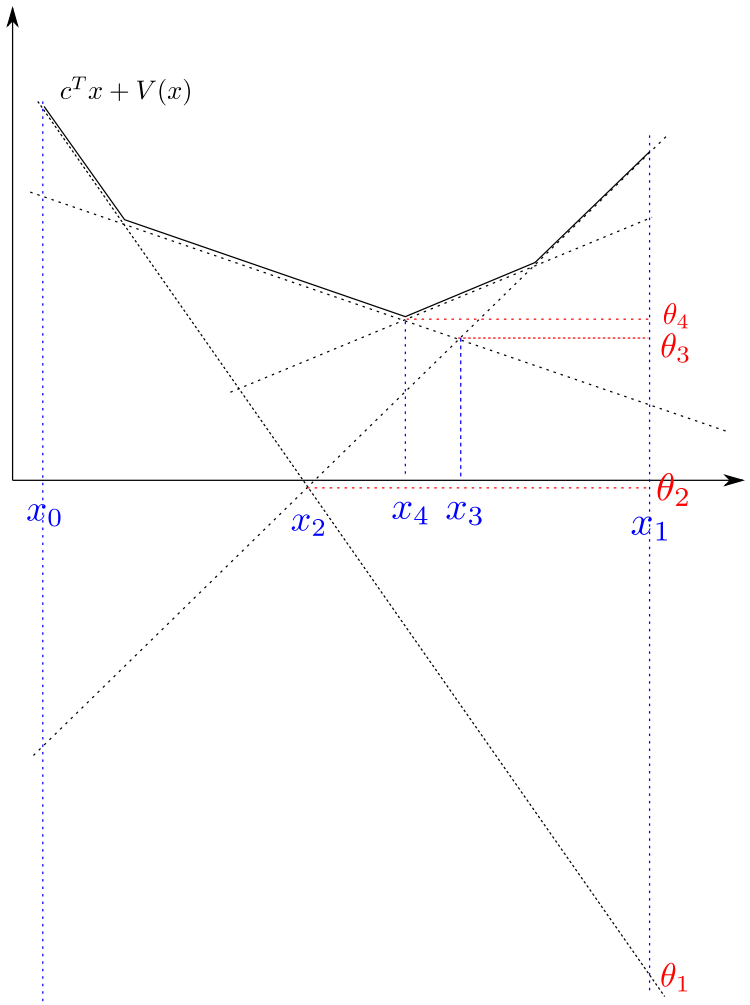


Figure 5.4 A graphical illustration of the evolution of Benders decomposition.

- If (M) is infeasible, exit. The problem is infeasible.

  Step 2: Solve (S) with x_k as input.

- If (S) is infeasible, let $R_{k+1} = R_k \cup \{\sigma_{k+1}\}$. Let $k = k + 1$ and return to step 1.
- If (S) is feasible, let $V_{k+1} = V_k \cup \{\pi_{k+1}\}$.
  - If $V_k = V_{k+1}$ then terminate with (x_k, y_{k+1}) as the optimal solution.
  - Else, let $k = k + 1$ and return to step 1.

The following bounds can be obtained from the algorithm as it progresses: at iteration k, $\underline{z} = c^T x_k + \theta_k$ is a lower bound and $\bar{z} = c^T x_k + V(x_k)$ is an

Table 5.1 The expected load duration curve in example 5.1.

	Duration (hours)	Level (MW)
Base load	8760	0-4235
Medium load	7000	4235-7496
Peak load	1500	7496-10401

upper bound. A graphical illustration of Benders decomposition is demonstrated in figure 5.4.

PROPOSITION 5.6 *The Benders decomposition algorithm converges finitely.*

Proof This follows from the fact that at each iteration, either a new cut is added, or the algorithm terminates (with a certificate of infeasibility or optimality). Since there is a finite number of vertices and extreme rays in $\{\pi : \pi^T W \leq q^T\}$, a finite number of cuts can be added. □

Example 5.1 Recall the capacity expansion planning problem of section 1.2. Consider the case of solving for the expected load duration curve, which is presented in table 5.1. The problem can be expressed as follows:

$$\begin{aligned}
&\min_{x,y\geq 0} \sum_{i=1}^{n}(I_i \cdot x_i + \sum_{j=1}^{m} C_i \cdot T_j \cdot y_{ij}) \\
&\text{s.t. } \sum_{i=1}^{n} y_{ij} = D_j, j = 1, \ldots, m \\
&\sum_{j=1}^{m} y_{ij} \leq x_i, i = 1, \ldots n-1
\end{aligned}$$

Benders decomposition proceeds by defining the following master problem:

$$\begin{aligned}
(M) : &\min_{x\geq 0} \sum_{i=1}^{n} I_i \cdot x_i + \theta \\
&\theta \geq \sum_{j=1}^{m} \lambda_j^v D_j + \sum_{i=1}^{n} \rho_i^v x_i, (\lambda^k, \rho^k) \in V_k \\
&\theta \geq 0
\end{aligned}$$

Table 5.2 The sequence of investment decisions generated by the Benders decomposition algorithm in example 5.1.

Iteration	Coal (MW)	Gas (MW)	Nuclear (MW)	Oil (MW)
1	0	0	0	0
2	0	0	0	8735.6
3	0	0	0	18565.1
4	0	14675.8	0	0
5	10673.3	0	0	0
6	0	0	7337.9	3063.1
7	0	1497.7	7337.9	732.2
8	0	1497.7	7337.9	2033.3
9	0	0	8966	1435
10	2851.8	2187.2	5362	0
11	8321	0	0	2080
12	6989.5	4489.5	56.5	0
13	3261	2905	4235	0

where $\lambda^k = (\lambda_1^k, \ldots, \lambda_{n-1}^k)$, $\rho^k = (\rho_1^k, \ldots, \rho_m^k)$ and λ_j^k, ρ_i^k are dual optimal multipliers obtained from the slave problem:

$$
\begin{aligned}
(S): & \min_{y \geq 0} \sum_{i=1}^{n} \sum_{j=1}^{m} C_i \cdot T_j \cdot y_{ij} \\
(\lambda_j): & \sum_{i=1}^{n} y_{ij} = D_j, j = 1, \ldots, m \\
(\rho_i): & \sum_{j=1}^{m} y_{ij} \leq \bar{x}_i, i = 1, \ldots n-1
\end{aligned}
$$

with $\bar{x}_i$ fixed from the master problem. Note that the constraint $\theta \geq 0$ has been added to the master problem because it is known that the second-stage cost cannot be negative. This constraint is in fact necessary for the algorithm to progress beyond the first iteration since the master problem otherwise becomes unbounded. The sequence of investment decisions generated by the algorithm is presented in table 5.2. Note that the investment decision changes in each iteration, since otherwise the algorithm would terminate. The 'greedy' behavior of the algorithm is evident, as the algorithm first attempts to invest in no technology at all, or technologies with low operating cost, whereas technologies with higher capital cost are added into the investment mix in later iterations due to optimality cuts.

5.2 The L-Shaped Method

The L-shaped method is a specific application of Benders decomposition in the context of two-stage stochastic linear programs. A 2-stage stochastic linear program can be written in *extensive form* by explicitly writing out all the constraints associated to the first and second stage:

$$\begin{aligned}
&\min c^T x + \mathbb{E}[\min q(\omega)^T y(\omega)] \\
&Ax = b \\
&T(\omega)x + W(\omega)y(\omega) = h(\omega), \omega \in \Omega \\
&x \geq 0, y(\omega) \geq 0
\end{aligned}$$

Recalling the notation from section 4.1, first-stage decisions are denoted as $x \in \mathbb{R}^{n_1}$. For a given realization ω, second stage decisions are denoted as $y(\omega) \in \mathbb{R}^{n_2}$. The deterministic parameters of the first stage are $c \in \mathbb{R}^{n_1}$, $b \in \mathbb{R}^{m_1}$, $A \in \mathbb{R}^{m_1 \times n_1}$. Second-stage data are $q(\omega) \in \mathbb{R}^{n_2}$, $h(\omega) \in \mathbb{R}^{m_2}$, $T(\omega) \in \mathbb{R}^{m_2 \times n_1}$ and $W(\omega) \in \mathbb{R}^{m_2 \times n_2}$. As in the previous section, we assume that the problem has a bounded optimal objective function value.

Two-stage stochastic linear programs obviously obey the format of problems described in section 5.1. Denote the finite collection of random outcomes as N, with $\omega = 1, \ldots, N$. The probability of each scenario is denoted as p_ω.

Following the development of section 5.1, value functions are introduced in order to describe the proposed algorithms. The Q-function is defined as follows[2]:

$$\begin{aligned}
(S_\omega) : Q_\omega(x) = \min_y \; & q_\omega^T y \\
& W_\omega y = h_\omega - T_\omega x \\
& y \geq 0.
\end{aligned}$$

and represents the cost of the best possible reaction to a first-stage decision x and a realization ω.

The dual of (S_ω) can be expressed as:

$$\begin{aligned}
(D_\omega) : \max_\pi \; & \pi^T (h_\omega - T_\omega x) \\
& \pi^T W_\omega \leq q_\omega^T
\end{aligned}$$

The **expected value function** is then defined as

$$V(x) = \sum_{\omega=1}^{N} p_\omega Q_\omega(x).$$

Any candidate first-stage decision x must ensure that there exists a feasible second-stage reaction y for *any* realization of uncertainty ω. The set of feasible first-stage decisions is denoted as $K_1 = \{x : Ax = b, x \geq 0\}$. The set of first-stage

[2] Alternatively, following the notation of chapter 4 we can denote the Q function as $Q(x, \xi(\omega))$.

decisions that have a feasible reaction for a given realization of uncertainty in the second stage is denoted as $K_2(\omega) = \{x : \exists y, T_\omega x + W_\omega y = h_\omega, y \geq 0\}$. The set of first-stage decisions that have a feasible reaction for *any* realization of uncertainty is then obtained as $K_2 = \cap_\omega K_2(\omega) = \text{dom } V$. A two-stage stochastic program has **relative complete recourse** if obeying first-stage constraints ensures that feasible second-stage decisions always exist, i.e. $K_1 \subseteq K_2$. A two-stage stochastic program has **complete recourse** if a feasible second-stage decision can be found, regardless of the first-stage decision and the realization of uncertainty.

PROPOSITION 5.7 *Let $\pi_{\omega 0}$ denote the dual optimal multipliers of (S_ω) given x_0.*

1. *$V(x)$ and $Q_\omega(x)$ are piecewise linear convex functions of x.*
2. *$\pi_{\omega 0}^T(h_\omega - T_\omega x)$ is a supporting hyperplane of $Q_\omega(x)$ at x_0.*
3. *$\sum_{\omega=1}^N p_\omega \pi_{\omega 0}^T(h_\omega - T_\omega x)$ is a supporting hyperplane of $V(x)$ at x_0.*

Proof The proof of statements 1 and 2 are identical to the proof of proposition 5.2. Statement 3 then follows as a direct consequence of the definition of $V(x) = \sum_{\omega=1}^N p_\omega Q_\omega(x)$.

□

In order to prove convergence of the L-shaped method, we convert the overall problem into the format of the previous section, i.e. we would like to think of the problem as involving a single first-stage decision vector x, and a second-stage decision vector y which is the concatenation of the scenario-specific decision vectors y_ω. We will further use the notation $\text{diag}(A_1, \ldots, A_n)$, where $A_i, i = 1, \ldots, n$ are matrices, in order to denote the square matrix that is obtained as

$$\text{diag}(A_1, A_2, \ldots, A_n) = \begin{pmatrix} A_1 & 0 & \ldots & 0 \\ 0 & A_2 & \ldots & 0 \\ \vdots & & & \\ 0 & 0 & \ldots & A_n \end{pmatrix}$$

Denote $y^T = [y_1^T, \ldots, y_N^T]$, $h^T = [h_1^T, \ldots, h_N^T]$, $T^T = [T_1^T, \ldots, T_N^T]$, $W = \text{diag } (W_\omega, \omega \in \{1, \ldots N\})$, $T = \text{diag } T_\omega$, and consider the full second-stage optimization:

$$\begin{aligned} (S) : \min_y & \sum_{\omega=1}^N p_\omega q_\omega^T y_\omega \\ & Wy = h - Tx \\ & y \geq 0. \end{aligned}$$

The following propositions establish a relationship between the feasible regions of the duals of (S) and (S_ω).

PROPOSITION 5.8 *Denote V as the set of extreme vertices of $\{\pi : \pi^T W \leq q^T\}$, and V_ω as the set of extreme vertices of $\{\pi : \pi^T W_\omega \leq q_\omega^T\}$ where $q^T = [q_1^T, \ldots, q_N^T]$. Then*

$$V = \{(p_1\pi_1^T, \ldots, p_N\pi_N^T)^T : \pi_1 \in V_1, \ldots, \pi_N \in V_N\}.$$

Proof The result follows from the fact that the constraint $\pi^T W \leq q^T$ is separable by scenario. □

PROPOSITION 5.9 *Denote R as the set of extreme rays of $\{\pi : \pi^T W \leq q^T\}$ and R_ω as the set of extreme rays of $\{\pi : \pi^T W_\omega \leq q_\omega^T\}$. Then*

$$R = \{(0, \ldots, \sigma_\omega^T, \ldots, 0)^T : \sigma_\omega \in R_\omega, \omega = 1, \ldots, N\}.$$

Proof The proof follows from the fact that the constraint $\pi^T W \leq q^T$ can be decomposed by scenario.

□

The preceding propositions imply that the original problem can be expressed equivalently as the following **deterministic equivalent program**:

$$\begin{aligned}
&\min c^T x + \theta \\
&Ax = b \\
&\sigma^T(h - Tx) \leq 0, \sigma \in R \\
&\theta \geq \pi^T(h - Tx), \pi \in V \\
&x \geq 0
\end{aligned}$$

As in the case of Benders decomposition, the L-shaped algorithm relies on the fact that $V(x)$ and dom V are defined by a set of inequalities. The algorithm is initialized by ignoring these inequalities and proceeds by adding one new inequality at each iteration to the following relaxation of the full optimization problem:

$$\begin{aligned}
(M) : z_k = &\min c^T x + \theta \\
&Ax = b \\
&\sigma^T(h - Tx) \leq 0, \sigma \in R_k \subseteq R \\
&\theta \geq \pi^T(h - Tx), \pi \in V_k \subseteq V \\
&x \geq 0
\end{aligned}$$

The computation of the inequalities that define $V(x)$ is prescribed by proposition 5.7, while the computation of the inequalities that define dom V is guided by proposition 5.9.

Let k be an iteration counter. As in the case of Benders decomposition, the resolution of the master program provides (i) a lower bound z_k to the optimal objective function value $z^\star$, (ii) a candidate solution x_k that can be fed

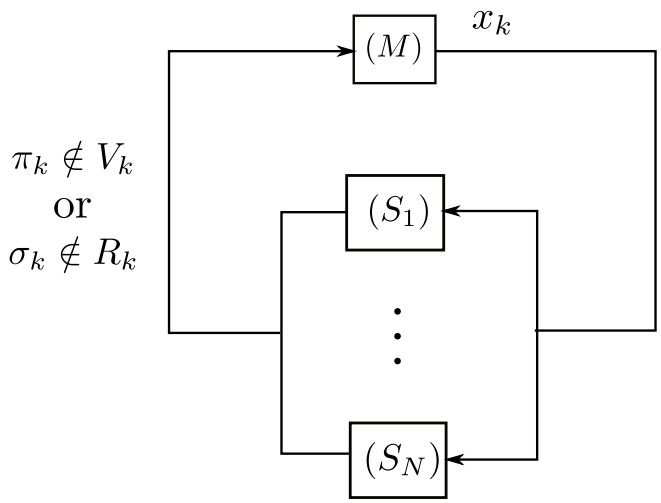


Figure 5.5 The exchange of information between the master and slave problem in the L-shaped algorithm.

to each slave program (S_ω), and (iii) an under-estimator of the value function, $\theta_k \leq V(x_k)$. The resolution of *all* (S_ω) with input x_k yields (i) an upper bound $c^T x_k + \sum_{\omega=1}^{N} p_\omega q_\omega^T y_{\omega,k+1}$ to the optimal objective function value $z^\star$, (ii) a new vertex $\pi_{k+1} = (p_1 \pi_{1,k+1}^T, \ldots, p_N \pi_{N,k+1}^T)^T$ (if all (S_ω) are feasible), or (iii) a new extreme ray $\sigma_{k+1} = (0, \ldots, \sigma_\omega^T, \ldots, 0)^T$ (if any (S_ω) is infeasible). The process is depicted in figure 5.5. Following the same reasoning as in the case of Benders decomposition, it can be seen that either the obtained extreme vertex or extreme ray adds information to the master program, or the algorithm terminates. The L-shaped algorithm applied to two-stage stochastic linear programs can then be stated as follows:

DEFINITION 5.10 L-shaped algorithm.
Step 0: Set $k = 0$, $V_0 = R_0 = \emptyset$.
Step 1: Solve (M).

- If (M) is feasible, store x_k.
- If (M) is infeasible, exit. The problem is infeasible.

Step 2: For $\omega = 1, \ldots, N$, solve (S_ω) with x_k as input.

- If (S_ω) is infeasible, let $S_{k+1} = S_k \cup \{\sigma_{k+1}\}$, where σ_{k+1} is an extreme ray of (S_ω). Let $k = k + 1$ and return to step 1.
- If (S_ω) is feasible, store $\pi_{\omega,k+1}$.

Step 3: Let $V_{k+1} = V_k \cup \{(p_1 \pi_{1,k+1}, \ldots, p_N \pi_{N,k+1})\}$.

- If $V_k = V_{k+1}$ then terminate with (x_k, y_{k+1}) as the optimal solution.
- Else, let $k = k + 1$ and return to step 1.

It should be noted that the algorithm of figure 5.5 can be accelerated through parallel computing, since each slave sub-problem can be solved independently.

Example 5.2 Consider the stochastic version of the capacity expansion planning problem. The stochastic programming formulation of the problem is provided in example 4.2. The master problem of the L-shaped algorithm is identical to that of the Benders algorithm, shown in example 5.1. The slave problem can be expressed as follows:

$$
\begin{aligned}
(S_\omega): \quad & \min_{y \geq 0} \sum_{i=1}^{n} \sum_{j=1}^{m} C_i \cdot T_j \cdot y_{ij} \\
(\lambda_j(\omega)): & \sum_{i=1}^{n} y_{ij} = D_j(\omega), j = 1, \ldots, m \\
(\rho_i(\omega)): & \sum_{j=1}^{m} y_{ij} \leq \bar{x}_i, i = 1, \ldots n-1
\end{aligned}
$$

where $\bar{x}$ has been fixed from the master problem. Recall that the height $D_j(\omega)$ of each load block is random, whereas the duration T_j is not. The sequence of investment decisions generated by the algorithm is presented in table 5.3. Note, again, that the the investment candidate proposed in each iteration is necessarily different from all past investment candidates. Once again the 'greedy' behavior of the algorithm can be observed with low capital cost technologies attempted in early iterations. The evolution of θ_k is presented in table 5.4. Note that θ_k need not be increasing, and may even be decreasing. For example, in iteration 12 the algorithm attempts a trial solution which does not rely on nuclear, and incurs an unnaturally high second stage cost of 186788 \$/MWh. The final iterations (12-15) oscillate around a near-optimal mix, thereby resulting in small changes of θ_k. Indeed, one motivation of the decomposition methods presented in this chapter is to 'hone in' on a neighborhood of the optimal solution and solve the problem by generating cutting planes of $V(x)$ within this neighborhood, without ever having to discover, or include in the master program, any of the cuts that support $V(x)$ far from the optimal neighborhood.

5.3 The Multi-Cut L-Shaped Method

The L-shaped method relies on dual multipliers of the second-stage subproblems for generating supporting hyperplanes of $V(x)$. Given a trial decision x_0, part 2 of proposition 5.7 prescribes that a supporting hyperplane of $V(x)$ at x_0 can be computed as $\sum_{\omega=1}^{N} p_\omega \pi_{\omega 0}^T (h_\omega - T_\omega x)$, where $\pi_{\omega 0}$ is the dual optimal multiplier

Table 5.3 The sequence of investment decisions generated by the L-shaped decomposition algorithm in example 5.2.

Iteration	Coal (MW)	Gas (MW)	Nuclear (MW)	Oil (MW)
1	0	0	0	0
2	0	0	0	8736
3	0	0	0	15999.6
4	0	14675.5	0	0
5	10673.8	0	0	0
6	10673.8	0	0	13331.8
7	0	163.8	7174.5	3830.8
8	0	3300.6	7868.4	0
9	0	5143.4	7303.9	1679.4
10	3123.9	1948.1	4953.7	1143.3
11	1680	4322.4	6625	0
12	8747.6	1652.8	0	768.6
13	5701.9	464.9	4233.6	768.6
14	4935.9	1405	3994.7	0
15	6552.6	386.3	3173.7	882.9
16	5085	1311	3919	854

Table 5.4 The sequence of value function approximations generated by the L-shaped and multi-cut L-shaped decomposition algorithm.

Iteration	L-shaped θ_k	Multicut $\sum_{\omega=1}^{N} p_\omega \theta_{\omega k}$
1	0	0
2	0	0
3	0	14674
4	0	61181
5	0	61181
6	0	28444
7	59736	83545
8	40998	186865
9	50222	108401
10	96290	171767
11	61593	125272
12	186788	
13	107349	
14	124788	
15	130041	
16	125272	

of the second-stage subproblem for scenario ω. Instead, one could attempt to approximate each $Q_\omega(x)$ at a trial point x_0. This idea is the basis of the **multi-cut L-shaped algorithm**.

The qualitative difference of the L-shaped method and the multi-cut L-shaped method can be illustrated in figure 5.6. The figure presents, in bold, the value

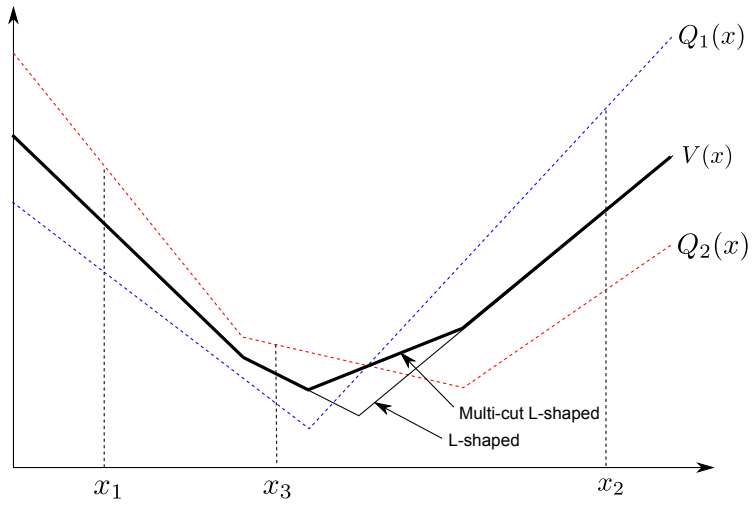


Figure 5.6 A graphical illustration of the difference between the L-shaped and the multi-cut L-shaped method. The bold line indicates $V(x)$, which is the weighted average of $Q_1(x)$ (blue line) and $Q_2(x)$ (red line). The multi-cut L-shaped method applied to trial points x_1, x_2 and x_3 results in an exact representation of $V(x)$, whereas the L-shaped method results in the approximation which is indicated in the light black line.

function $V(x)$, which is the weighted average of $Q_1(x)$ (blue line) and $Q_2(x)$ (red line). Note that since there are three trial points, the L-shaped method generates an approximation of $V(x)$ which necessarily consists of no more than three points, and is shown in light black in the figure. In fact, there is a region of $V(x)$ for which the L-shaped method produces an approximation which is strictly less than $V(x)$. Instead, for the three trial points considered in the figure the multi-cut L-shaped method exactly approximates $V(x)$. The difference between the methods lies in the fact that, given the three trial points, each of the linear regions of both $Q_1(x)$ and $Q_2(x)$ is computed exactly, but whereas the L-shaped method averages the slopes to approximate $V(x)$ the multi-cut L-shaped method stores the slopes of $Q_1(x)$ and $Q_2(x)$ separately.

The key difference between the L-shaped and the multi-cut L-shaped method is the master program. The notation follows that of the previous section. The new element is a set of auxiliary variables θ_ω, one for each scenario, which are used for approximating $Q_\omega(x)$. The master program of the multi-cut L-shaped

method is then expressed as follows:

$$
\begin{aligned}
(M): \min\, & c^T x + \sum_{\omega=1}^{N} p_\omega \theta_\omega \\
& Ax = b \\
& \sigma^T (h_\omega - T_\omega x) \leq 0, \sigma \in R_{\omega k} \subseteq R_\omega \\
& \theta_\omega \geq \pi^T (h_\omega - T_\omega x), \pi \in V_{\omega k} \subseteq V_\omega \\
& x \geq 0
\end{aligned}
$$

It is evident that the multi-cut L-shaped method makes better use of the information generated from the resolution of second-stage subproblems. The drawback of the multi-cut L-shaped method is the amount of cuts that are stored in the master program, which makes the master program grow more rapidly than in the case of the L-shaped method.

The termination of the algorithm is

DEFINITION 5.11 Multi-cut L-shaped algorithm.

Step 0: Set $k = 0$, $V_{\omega 0} = R_{\omega 0} = \emptyset$ for all ω.

Step 1: Solve (M).

- If (M) is feasible, store x_k.
- If (M) is infeasible, exit. The problem is infeasible.

Step 2: For $\omega = 1, \ldots, N$, solve (S_ω) with x_k as input.

- If (S_ω) is infeasible, let $S_{\omega,k+1} = S_{\omega k} \cup \{\sigma_{\omega,k+1}\}$. Let $k = k+1$ and return to step 1.
- If (S_ω) is feasible, store $\pi_{\omega,k+1}$.

Step 3: For $\omega = 1, \ldots, N$, let $V_{\omega,k+1} = V_{\omega k} \cup \{\pi_{\omega,k+1}\}$.

- If $V_{\omega k} = V_{\omega,k+1}$ for all ω then terminate with (x_k, y_{k+1}) as the optimal solution.
- Else, let $k = k+1$ and return to step 1.

Example 5.3 Consider again the stochastic version of the capacity expansion planning problem, presented in example 5.2. The master problem of the multi-cut L-shaped method can be described as follows:

$$
\begin{aligned}
(M): \min_{x \geq 0}\, & \sum_{i=1}^{n} I_i \cdot x_i + \sum_{\omega=1}^{N} p_\omega \theta_\omega \\
& \theta_\omega \geq \sum_{j=1}^{m} \lambda_{\omega j}^{v} D_j + \sum_{i=1}^{n} \rho_{\omega i}^{v} x_i, v \in V_{\omega k} \\
& \theta_\omega \geq 0
\end{aligned}
$$

Table 5.5 The sequence of investment decisions generated by the multi-cut L-shaped method in example 5.3.

Iteration	Coal (MW)	Gas (MW)	Nuclear (MW)	Oil (MW)
1	0	0	0	0
2	0	0	0	10701.3
3	0	14309.6	0	0
4	10407	0	0	0
5	0	0	7154.8	5034.6
6	0	2329	7154.8	1410
7	0	1280.2	8518.8	1647.6
8	2102.1	3310.9	5756	0
9	8767.3	236.7	0	2291.4
10	6396	0	3919	1168.8
11	8230.5	2165	773.5	0
12	5085	1311	3919	854

The sequence of investment decisions generated by the algorithm is presented in table 5.5. The algorithm requires 12 iterations to converge, compared to 16 iterations required by the L-shaped algorithm. Although the multi-cut L-shaped method may require more iterations in order to converge than the L-shaped method (see exercise 5.4), it commonly converges in fewer iterations. The convergence of the value function approximation, $\sum_{\omega=1}^{N} p_\omega \theta_\omega$, is presented in table 5.4. Note that the multi-cut L-shaped method incurs a non-zero second stage cost already in iteration 3, whereas the L-shaped method requires 7 iterations. Also note that the value function approximations in iterations 5 and 6 are identical, yet the algorithm has not converged. This can happen if the same optimality cuts are active from one iteration to the next, and does not imply convergence.

Problems

5.1 Implement the Benders decomposition algorithm and replicate the results of example 5.1.

5.2 Implement the L-shaped method and replicate the results of example 5.2.

5.3 Implement the multi-cut L-shaped method and replicate the results of example 5.3.

5.4 Provide an example where the multi-cut L-shaped method requires more iterations than the L-shaped method.

6 Nested Decomposition

The algorithms that are presented in chapter 5 are based on the principle of decomposing a two-stage optimization problem into two separate stages of optimization. The key ingredient for achieving this decomposition is the value function that maps actions in the first stage to expected second-stage cost. In this chapter, this idea is extended to a multi-stage setting. Section 6.1 recalls the application of dynamic programming on multi-stage stochastic linear programs, and discusses various properties of the dynamic programming value functions which are exploited computationally. Section **??** defines the nested L-shaped decomposition subproblem, which is the building block of extending the L-shaped algorithm to a multi-period setting. The nested decomposition is presented in section 6.2, and generalizes the previously introduced L-shaped algorithm.

6.1 Value Functions of Multi-Stage Stochastic Linear Programs

In this section we recall the definition of multi-stage stochastic linear programs (MSLPs), we recall the application of dynamic programming in the context of MSLPs, and we prove that the dynamic programming value functions are piecewise affine convex functions with polyhedral domains. This generalizes the result of the previous chapter to multiple stages, and sets the foundation for generalizing the L-shaped method to a multi-stage version, called the nested decomposition algorithm.

Recall the definition of a multi-stage stochastic linear program on a scenario tree:

$$
\begin{aligned}
&(MSLP - ST):\\
&\min c_1^T x_1 + \mathbb{E}[c_2(\omega_{[2]})^T x_2(\omega_{[2]}) + \cdots + c_H(\omega_H)^T x_H(\omega_{[H]})]\\
&\text{s.t. } W_1 x_1 = h_1\\
&T_1(\omega_{[2]})x_1 + W_2(\omega_{[2]})x_2(\omega_{[2]}) = h_2(\omega_{[2]}), \omega_{[2]} \in S_1 \times S_2\\
&\vdots\\
&T_{t-1}(\omega_{[t]})x_{t-1}(\omega_{[t-1]}) + W_t(\omega_{[t]})x_t(\omega_{[t]}) = h_t(\omega_{[t]}), \omega_{[t]} \in S_1 \times \ldots \times S_t\\
&\vdots\\
&T_{H-1}(\omega_{[H]})x_{H-1}(\omega_{[H-1]}) + W_H(\omega_{[H]})x_H(\omega_{[H]}) = h_H(\omega_{[H]}),\\
&\omega_{[H]} \in S_1 \times \ldots \times S_H\\
&x_1 \geq 0, x_t(\omega_{[t]}) \geq 0, t = 2, \ldots, H, \omega_{[t]} \in S_1 \times, \ldots, S_t
\end{aligned}
$$

Recall that Ξ_t corresponds to the support of the random vector ξ_t, and $\Xi_{[t]} = \Xi_1 \times \ldots \times \Xi_t$ correponds to the support of the random process $\xi_{[t]} = (\xi_1, \ldots, \xi_t)$. In what follows, we will refer interchangeably to Ξ_t and S_t as the set of lattice nodes in stage t, and we will refer interchangeably to $S_1 \times \ldots \times S_t$ and $\Xi_{[t]}$ as the set of histories up to stage t. Furthermore, the notation $c_t(\omega_{[t]})$ and $c_{t,\omega_{[t]}}$ is used interchangeably for random variables, random vectors, and random matrices corresponding to node $\omega_{[t]}$ of a lattice.

The dynamic programming algorithm, applied to multi-stage stochastic linear programming on a scenario tree[1], proceeds as follows. For $\omega_{[H]} \in S_1 \times \ldots \times S_H$ in the final period of the horizon we have:

$$
\begin{aligned}
Q_H(x_{H-1}, \xi_H) = \min_{x_H} \; & c_H(\omega_{[H]})^T x_H\\
\text{s.t. } & T_{H-1}(\omega_{[H]})x_{H-1} + W_H(\omega_{[H]})x_H = h_H(\omega_{[H]})\\
& x_H \geq 0
\end{aligned}
$$

The value function for the final period is then defined as

$$V_H(x_{H-1}, \omega_{[H-1]}) = \mathbb{E}_{\xi_H}[Q_H(x_{H-1}, \xi_H)|\omega_{[H-1]}].$$

Proceeding recursively, at a given time step t, the Q-function is computed as:

$$
\begin{aligned}
Q_t(x_{t-1}, \xi_t) = \min_{x_t} \; & c_t(\omega_{[t]})^T x_t + V_{t+1}(x_t, \omega_{[t]})\\
\text{s.t. } & T_{t-1}(\omega_{[t]})x_{t-1} + W_t(\omega_{[t]})x_t = h_t(\omega_{[t]})\\
& x_t \geq 0
\end{aligned}
$$

The value function of stage t is then obtained as

$$V_t(x_{t-1}, \omega_{[t-1]}) = \mathbb{E}_{\xi_t}[Q_t(x_{t-1}, \xi_t)|\omega_{[t-1]}]. \tag{6.1}$$

[1] Note that in chapter 4 the same algorithm is presented with notation applied to a lattice model of uncertainty.

The solution of the original problem is then obtained by solving the following first-stage problem:

$$\begin{aligned} &\min c_1^T x_1 + V_2(x_1) \\ &\text{s.t. } W_1 x_1 = h_1 \\ &x_1 \geq 0 \end{aligned}$$

It has been shown in chapter 5 that the value function in two-stage stochastic linear programming is piecewise linear convex, and the domain of the function (the set of first-stage decisions that result in finite second-stage cost) is polyhedral. The following proposition establishes a similar result for multi-stage stochastic linear programs.

PROPOSITION 6.1 *Consider a multi-stage stochastic linear program defined on a scenario tree, and denote $S_1 \times \ldots \times S_t$ as the set of nodes in stage t. For finite $S_t, t = 1, \ldots, H$, the functions $V_{t+1,\omega_{[t]}}(x_t)$ and $Q_{t+1}(x_t, \xi_{t+1})$ are piecewise linear convex. Moreover, dom $V_{t+1,\omega_{[t]}}$ and dom Q_{t+1} are polyhedral functions of x_t.*

Coming soon: notation needs to be aligned with section 6.3

Proof For node k stage H, the following Q function can be defined:

$$\begin{aligned} Q_H(x_{H-1}, \xi_H) = \min_x \; & c_{H,k}^T x \\ (\pi) : \; & W_{H,k} x = h_{H,k} - T_{H-1,k} x_{H-1} \\ & x \geq 0 \end{aligned}$$

Denote the dual of this linear program as

$$\begin{aligned} \max_\pi \; & \pi^T (h_{H,k} - T_{H-1,k} x_{H-1}) \\ \text{s.t.} \; & \pi^T W_{H,k} \leq c_{H,k}^T. \end{aligned}$$

The extreme rays of the feasible region of this dual are denoted as $R_{H,k}$. The extreme points are denoted as $P_{H,k}$. Both sets are finite and *independent of* x_{H-1}. The Q function can then be written equivalently

$$Q_H(x_{H-1}, \xi_H) = \max_{\pi \in P_{H,k}} \pi^T (h_{H,k} - T_{H-1,k} x_{H-1})$$

if $x_{H-1} \in \{\sigma^T (h_{H,k} - T_{H-1,k} x_{H-1}) \leq 0, \sigma \in R_{H,k}\}$, and becomes $+\infty$ otherwise. Since the sets $P_{H,k}$ and $R_{H,k}$ are finite, the resulting Q function is a piecewise affine convex of x_{H-1} with a polyhedral domain.

The value function $V_{H,\omega_{[H-1]}}(x_{H-1}), \omega_{[H-1]} \in S_1 \times \ldots \times S_{H-1}$, is also piecewise affine convex, since it is an average of piecewise affine convex Q-functions, and its domain is polyhedral, since it is the intersection of a finite number of polyhedra (the domains of the Q-functions of stage H). Therefore, the proposition is true for the end of the horizon. We proceed by induction, with the inductive assumption that the statement of the proposition is true for stage $t+1$.

Using the inductive assumption and the definition of Q_t, we have:

$$
\begin{aligned}
Q_t(x_{t-1},\xi_t) = \min_{x_t,\theta}\ & c_{t,k}^T x_t + \theta \\
(\pi):\ & W_{t,k}x_t = h_{t,k} - T_{t-1,k}x_{t-1} \\
(\rho_i):\ & \theta \geq e_{t+1,k,i} - E_{t+1,k,i}x_t, i = 1,\ldots,n_{t,k} \\
(\sigma_i):\ & D_{t+1,k,i}x_t - d_{t+1,k,i} \geq 0, i = 1,\ldots,m_{t,k} \\
& x_t \geq 0
\end{aligned}
$$

where $(e_{t+1,k,i}, E_{t+1,k,i}), i = 1,\ldots,n_{t,k}$ describe the cutting planes that define the piecewise affine convex function $V_{t+1,k}$, $(d_{t+1,k,i}, D_{t+1,k,i}), i = 1,\ldots,m_{t,k}$ describe the polyhedral domain of $V_{t+1,k}$, $n_{t,k}$ denotes the number of affine inequalities needed to describe the function $V_{t+1,k}$, and $m_{t,k}$ denotes the number of affine inequalities needed to describe its domain. The dual of the above linear program is obtained as

$$
\begin{aligned}
& \max_{\pi,\rho_i,\sigma_i} \pi^T(h_{t,k} - T_{t-1,k}x_{t-1}) + \sum_{i=1}^{n_{t,k}} \rho_i^T e_{t+1,k,i} - \sum_{i=1}^{m_{t,k}} \sigma_i^T d_{t+1,k,i} \\
& \text{s.t. } \pi^T W_{t,k} + \sum_{i=1}^{n_{t,k}} \rho_i^T E_{t+1,k,i} - \sum_{i=1}^{m_{t,k}} \sigma_i^T D_{t+1,k,i} \leq c_{t,k}^T \\
& \sum_{i=1}^{n_{t,k}} 1^T \rho_i = 1 \\
& \rho_i \geq 0, i = 1,\ldots,n_{t,k} \\
& \sigma_i \geq 0, i = 1,\ldots,m_{t,k}
\end{aligned}
$$

Note that the feasible region of this problem is a polyhedron with a finite set of vertices $P_{t,k}$ and extreme rays $R_{t,k}$. Hence, Q_t can be rewritten as

$$
\begin{aligned}
& Q_t(x_{t-1},\xi_t) = \max_{(\pi,\rho_1,\ldots,\rho_{n_{t,k}},\sigma_1,\ldots,\sigma_{m_{t,k}})\in P_{t,k}} \pi^T(h_{t,k} - T_{t-1,k}x_{t-1}) + \\
& \sum_{i=1}^{n_{t,k}} \rho_i^T e_{t+1,k,i} - \sum_{i=1}^{m_{t,k}} \sigma_i^T d_{t+1,k,i},
\end{aligned}
$$

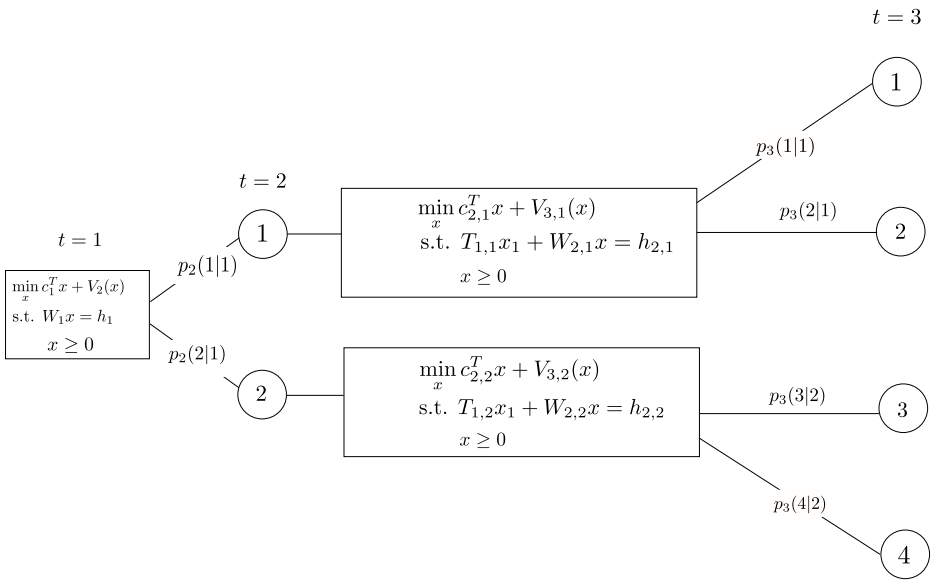
provided x_{t-1} belongs to the following set:

$$
\begin{aligned}
& \pi^T(h_{t,k} - T_{t-1,k}x_{t-1}) + \sum_{i=1}^{n_{t,k}} \rho_i^T e_{t+1,k,i} - \sum_{i=1}^{m_{t,k}} \sigma_i^T d_{t+1,k,i} \leq 0 \\
& (\pi,\rho_1,\ldots,\rho_{n_{t,k}},\sigma_1,\ldots,\sigma_{m_{t,k}}) \in R_{t,k}
\end{aligned}
$$

Otherwise, the Q function is $+\infty$. Since the sets $P_{t,k}$ and $R_{t,k}$ are finite, the Q function is a piecewise affine convex function of x_{t-1}, and its domain is polyhedral.

Since $V_{t,\omega_{t-1}}, \omega_{t-1} \in S_{t-1}$ is given by a finite non-negative sum of such Q-functions with polyhedral domains, it is also a piecewise affine convex function

Figure 6.1 A scenario tree for a multistage stochastic linear program. The function $V_{t+1,k}$ is the value function of stage $t+1$, given realization k in stage t.



with a polyhedral domain. Since the inductive assumption has been shown to be true, the conclusion of the proposition follows.

□

The preceding result is exploited in the next section in order to develop nested decomposition, which is an extension of the L-shaped method to multi-stage programs.

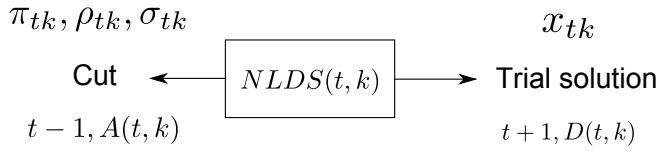
6.2 The Nested Decomposition Algorithm

Proposition 6.1, which is the main result of the previous section, implies that we can insert value functions as linear programs in the dynamic programming algorithm. This idea is illustrated in figure 6.1.

The boxes in figure 6.1 contain the building block of nested decomposition. This building block is the **nested L-shaped decomposition subproblem (NLDS)**, which is the linear subproblem solved at each node k of the scenario tree of the problem for a given time stage t. As such, the $NLDS$ is indexed by the time stage t and the node k of a given time stage, $NLDS(t,k)$.

Proposition 6.1 implies that the dynamic programming recursion for multi-stage stochastic linear programs, equation (6.1), can be cast as a linear program. According to figure 6.1, one linear program is associated with each time stage

Figure 6.2 $NLDS(t,k)$ communicates primal information with future time periods and dual information with previous time periods.

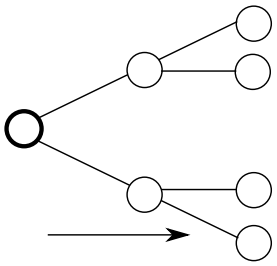


and node of the scenario tree. As in the case of the L-shaped method, the nested decomposition algorithm approximates the value function and its domain at each stage and scenario with a relaxation.

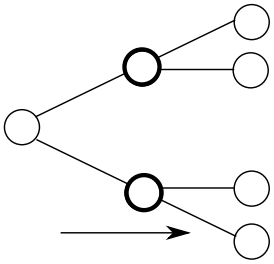
In order to clarify how the L-shaped algorithm generalizes, in multiple stages, to the nested decomposition algorithm, consider a node of the scenario tree that is identified uniquely by (t,k), i.e. its time index and $k \in \Xi_{[t]}$. Each $NLDS$ originates from an ancestor in the previous stage of the scenario tree, which is denoted as $A(t,k)$, and is associated with a set of descendants in the following time stage, which is denoted as $D(t,k)$. The algorithm is separated into a forward and a backward pass. In the forward pass the algorithm generates primal trial solutions (primal information) that are communicated to descendants, in the backward pass the algorithm generates cuts (dual information) that are communicated to the ancestor. The communication of information is shown in figure 6.2. Nested decomposition amounts to a repeated application of the L-shaped method, with variants of the algorithm depending on how the scenario tree is traversed. Figure 6.3 presents an example of how nested decomposition could proceed on a three-stage scenario tree, whereby all nodes of a given time stage are processed before proceeding to the next time stage (in the forward pass), and similarly all nodes of a given time stage are processed before proceeding to the previous time stage (in the backward pass).

Example 6.1 Consider the newsboy problem of example 4.1 in a two-stage setting. The unit cost of newspapers is denoted as C, the sales price of newspapers in the second stage is denoted as S. x denotes the amount of newspapers procured in the first stage, s is the amount of papers sold in the second stage. The $NLDS$ for the first time stage, before any cuts are added, can be expressed as follows:

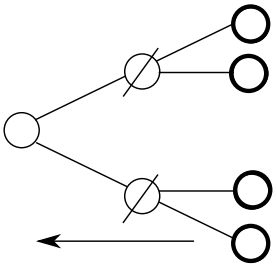
$$\begin{aligned} &NLDS(1) : \min_{x} C \cdot x \\ &\text{s.t. } x \geq 0 \end{aligned}$$



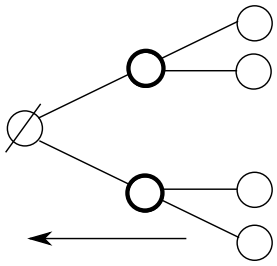
- Node: $(t = 1, k = 1)$
- Direction: forward
- Output: $x_{1,1}$



- Nodes: $(t = 2, k), k \in \{1, 2\}$
- Direction: forward
- Output: $x_{2k}, k \in \{1, 2\}$



- Nodes: $(t = 3, k), k \in \{1, 2, 3, 4\}$
- Direction: backward
- Output: $\pi_{3k}, \rho_{3k}, \sigma_{3k}, k \in \{1, 2, 3, 4\}$



- Nodes: $(t = 2, k), k \in \{1, 2\}$
- Direction: backward
- Output: $\pi_{2k}, \rho_{2k}, \sigma_{2k}, k \in \{1, 2\}$

Figure 6.3 An example trajectory of the nested decomposition algorithm on a three-stage scenario tree.

The $NLDS$ for stage 2 and outcome k is given by

$$
\begin{aligned}
&NLDS(2,k) : \min_{s} -P \cdot s \\
&\text{s.t. } s \leq D_{2,k} \\
&s \leq x \\
&s \geq 0
\end{aligned}
$$

Example 6.2 Consider the hydrothermal scheduling problem of example 3.1 in a multi-stage setting. Recalling notation, C represents the marginal cost of thermal units, E corresponds to the reservoir capacity of the hydroelectric dam, R is a random variable corresponding to rainfall, D_t represents the demand at a

given time stage, and the decision variables correspond to the amount of hydro power stored in the dam x, the amount of hydro power production q, and the amount of thermal production p. The $NLDS$ at an arbitrary stage t for a given outcome k can be stated as follows:

$$
\begin{aligned}
& NLDS(t,k) : \min_{x,q,p} C \cdot p \\
& \text{s.t. } x \leq E \\
& x \leq x_{t-1} + R_{t,k} - q \\
& p + q \geq D_t \\
& x, q, p \geq 0
\end{aligned}
$$

As in the case of the L-Shaped algorithm, the idea of nested decomposition if to produce outer approximations of the value functions and their domain by employing optimality cuts and feasibility cuts respectively. Concretely, suppose that we have been able to obtain $r_{t,k}$ optimality cuts and $s_{t,k}$ feasibility cuts (we will explain how to obtain these cuts later). Then, the NLDS can be described as follows:

$$
\begin{aligned}
NLDS(t,k) : & \min_{x,\theta} (c_{t,k})^T x + \theta \\
(\pi) : \quad & W_{t,k} x = h_{t,k} - T_{t-1,k} x_{t-1,A(t,k)} \\
(\rho_j) : \quad & E_{t,k,j} x + \theta \geq e_{t,k,j}, j = 1, \ldots, r_{t,k} \qquad (6.2) \\
(\sigma_j) : \quad & D_{t,k,j} x \geq d_{t,k,j}, j = 1, \ldots, s_{t,k} \qquad (6.3) \\
& x \geq 0
\end{aligned}
$$

where $A(t,k)$ is the ancestor of scenario k at stage $t-1$, and $x_{t-1,A(t,k)}$ is the trial solution that has been communicated from $A(t,k)$ to $NLDS(t,k)$. Constraints (6.2) are optimality cuts that approximate $V_{t+1,k}$, and constraints (6.3) are feasibility cuts that approximate dom $V_{t+1,k}$.

For the first stage of the problem, $t = 1$, the right-hand side of the first constraint is replaced by a constant vector b that corresponds to initial conditions. For the last stage of the problem, the variable θ and the optimality and feasibility cuts are removed from $NLDS$.

Let us now explain how we can obtain the optimality and feasibility cuts of equations (6.2) and (6.3). Suppose that we have passed a candidate primal solution $x_{t-1,A(t,k)}$ to $NLDS(t,k)$ from $NLDS(t-1, A(t,k))$, and it turns out that $NLDS(t,k)$ is infeasible for this specific choice of candidate primal solution $x_{t-1,A(t,k)}$. We are interested in using the dual information from the resolution of $NLDS(t,k)$ in order to generate a feasibility cut for $NLDS(t-1, A(t,k))$. To

do this, we start by expressing the dual of $NLDS(t,k)$ as follows:

$$\max_{\pi,\rho,\sigma} \pi^T(h_{t,k} - T_{t-1,k}x_{t-1,A(t,k)}) + \sum_{j=1}^{r_{t,k}} \rho_j^T e_{t,k} + \sum_{j=1}^{s_{t,k}} \sigma_j^T d_{t,k,j}$$

$$\text{s.t. } \pi^T W_{t,k} + \sum_{j=1}^{r_{t,k}} \rho_j^T E_{t,k,j} + \sum_{j=1}^{s_{t,k}} \sigma_j^T D_{t,k,j} \leq c_{t,k}^T$$

$$\sum_{j=1}^{r_{t,k}} 1^T \rho_j = 1$$

$$\rho_1, \ldots, \rho_{r_{t,k}} \geq 0$$

$$\sigma_1, \ldots, \sigma_{s_{t,k}} \geq 0$$

A certificate of infeasibility of $NLDS(t,k)$ is equivalent to a certificate of unboundedness of the dual, i.e. a vector $(\pi, \sigma_1, \ldots, \sigma_{s_{t,k}})$ with $\sigma_j \geq 0, j = 1, \ldots, s_{t,k}$ such that *(i)* $\pi^T(h_{t,k} - T_{t-1,k}x_{t-1,A(t,k)}) + \sum_{j=1}^{s_{t,k}} \sigma_j^T d_{t,k,j} > 0$, and *(ii)* $\pi^T W_{t,k} + \sum_{j=1}^{s_{t,k}} \sigma_j^T D_{t,k,j} \leq 0$. The following cut is generated from a vector $(\pi, \sigma_1, \ldots, \sigma_{s_{t,k}})$ that verifies the infeasibility of $NLDS(t,k)$:

$$(FC) : D_{t-1,A(t,k)}x \leq d_{t-1,A(t,k)} \tag{6.4}$$

where

$$D_{t-1,A(t,k)} = \pi^T T_{t-1,k}$$

$$d_{t-1,A(t,k)} = \pi^T h_{tk} + \sum_{j=1}^{s_{t,k}} \sigma_j^T d_{t,k,j}$$

This is a valid feasibility cut for $NLDS(t-1, A(t,k))$, in the sense that it is not satisfied by $x_{t-1,A(t,k)}$, but the inequality is satisfied for all $x \in \text{dom } \tilde{V}_{t-1,k}$, where $\tilde{V}$ is the outer approximation of the true value function which is available given the existing collection of optimality and feasibility cuts. Since the proposed feasibility cut excludes the vector $x_{t-1,A(t,k)}$ from future consideration in $NLDS(t-1, A(t,k))$ but does not exclude any vectors $x \in \text{dom } V_{t-1,k} \subseteq \text{dom } \tilde{V}_{t-1,k}$, it is indeed helping us make progress in solving the problem.

Similarly, optimality cuts for $NLDS(t-1,j)$ are generated by solving $NLDS(t,k)$ for all $k \in D(t-1,j)$ and storing the dual optimal multipliers π_{tk}. Then, the following cut can be computed:

$$E_{t-1,j}x + \theta \geq e_{t-1,j} \tag{6.5}$$

where

$$E_{t-1,j} = \sum_{k \in D(t-1,j)} p_t(k|j) \cdot (\pi_{t,k})^T T_{t-1,k}$$

$$e_{t-1,j} = \sum_{k \in D(t-1,j)} p_t(k|j) \cdot ((\pi_{t,k})^T h_{t,k} + \sum_{i=1}^{r_{t,k}} \rho_{t,k,i}^T e_{t,k,i} + \sum_{i=1}^{s_{t,k}} \sigma_{t,k,i}^T d_{t,k,i})$$

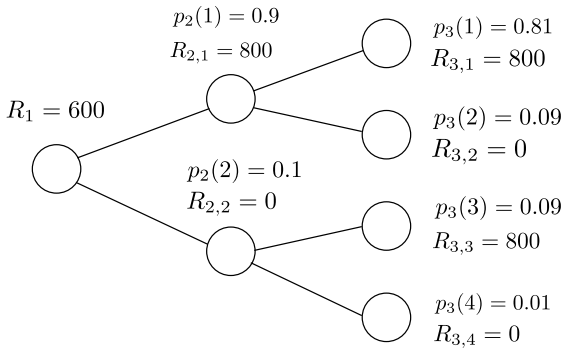


Figure 6.4 The three-stage scenario tree of example 6.3.

This is a valid optimality cut for $NLDS(t-1, j)$, in the sense that $V_{t-1,j}(x) \geq e_{t-1,j} - E_{t-1,j}x$ for any x. To see why this is the case, note first (this follows from weak duality, when considering the dual of $NLDS(t, k)$) that, by the way that we have constructed the optimality cut, we have $\tilde{V}_{t-1,j}(x) \geq e_{t-1,j} - E_{t-1,j}x$, where $\tilde{V}$ is the outer approximation of the true value function given the current collection of optimality and feasibility cuts. And since $\tilde{V}_{t-1,j}(x)$ is an outer approximation of $V_{t-1,j}(x)$, we have that $V_{t-1,j}(x) \geq \tilde{V}_{t-1,j}(x) \geq e_{t-1,j} - E_{t-1,j}x$.

Having clarified how to generate optimality and feasibility cuts for the refinement of outer approximation of the value functions and their domains, what remains to be clarified is how exactly we traverse the scenario tree. There is no unique choice for deciding how to do this, and in what follows we will describe one among various options.

The nested decomposition algorithm is described in table 6.1. The algorithm consists of forward and backward passes. During forward passes, the algorithm generates trial solutions for future time stages. During backward passes the algorithm generates optimality cuts for previous time stages. The scenario tree can be traversed in various ways. The algorithm presented in table 6.1 changes from forward to backward pass whenever it encounters an infeasible subproblem, and changes from backward to forward pass whenever it reaches $t = 1$. For example, when the algorithm is in a forward pass in a time stage $1 < t < H$ and encounters an infeasible $NLDS$, it adds a feasibility cut to the ancestor subproblem of the previous time stage, steps back to the ancestor $NLDS$, and reverts to a backward pass (instead of, for example, retaining a forward pass).

PROPOSITION 6.2 *If all Ξ_t are finite sets and all x have finite upper bounds, then the nested L-shaped method converges finitely to an optimal solution.*

Proof See (Birge & Louveaux 2010), page 268.

□

Table 6.1 The nested decomposition algorithm.

Pass	t	k	Result	Action
F	1		Feasible	$t \leftarrow 2$, $k \leftarrow 1$, Store θ_1, x_1 Send x to $NLDS(2, j)$, $j \in D(1)$
F	1		Infeasible	Infeasible, exit
F	$1 < t \leq H-1$	$k < \lvert\Xi_{[t]}\rvert$	Feasible	$k \leftarrow k+1$, Send x to $NLDS(t+1, j), j \in D(t,k)$
F	$1 < t \leq H-1$	$k < \lvert\Xi_{[t]}\rvert$	Infeasible	$k \leftarrow k+1$ Add FC to $NLDS(t-1, A(t,k))$
F	$1 < t \leq H-1$	$\lvert\Xi_{[t]}\rvert$	Feasible	$t \leftarrow t+1$, $k \leftarrow 1$ Send x to $NLDS(t+1, j), j \in D(t,k)$ If $t = H-1$ then Pass $\leftarrow$ B
F	$1 < t \leq H-1$	$\lvert\Xi_{[t]}\rvert$	Infeasible	$t \leftarrow t+1$, $k \leftarrow 1$ Add FC to $NLDS(t-1, A(t,k))$ If $t = H-1$ then Pass $\leftarrow$ B
B	$t \geq 2$	$k < \lvert\Xi_{[t]}\rvert$	Feasible	$k \leftarrow k+1$, Store (π, ρ, σ)
B	$t \geq 2$	$k < \lvert\Xi_{[t]}\rvert$	Infeasible	$k \leftarrow k+1$ Add FC to $NLDS(t-1, A(t,k))$
B	2	$\lvert\Xi_{[t]}\rvert$	Feasible	Pass $\leftarrow$ F, $t \leftarrow 1$ Add OC to $NLDS(1)$ If $\theta_1 \geq e - Ex_1$: Optimal, exit
B	2	$\lvert\Xi_{[t]}\rvert$	Infeasible	Pass $\leftarrow$ F, $t \leftarrow 1$ Add FC to $NLDS(1)$
B	$t > 2$	$\lvert\Xi_{[t]}\rvert$	Feasible	$t \leftarrow t-1$, $k \leftarrow 1$ Add OC to $NLDS(t-1, A(t,k))$
B	$t > 2$	$\lvert\Xi_{[t]}\rvert$	Infeasible	$t \leftarrow t-1$, $k \leftarrow 1$ Add FC to $NLDS(t-1, A(t,k))$

Example 6.3 This example revisits the hydrothermal scheduling problem. The demand for each period amounts to 1000 MW. Hydro dams can hold up to 750 MWh of energy. Thermal production is available at a marginal cost of 25 \$/MWh. Thermal generators have a technical capacity of 500 MW. The cost of not serving demand amounts to 1000 \$/MWh. Rainfall in each period is described by the scenario tree of example 6.4. The nested decomposition commences with the following $NLDS$ in stage 1:

$$
\begin{aligned}
&NLDS(1) : \min 25 \cdot p + 1000 \cdot l \\
&\text{s.t. } x \leq 750 \\
&x \leq 600 - q \\
&p + q + l \geq 1000 \\
&p \leq 500 \\
&x, p, q, l \geq 0
\end{aligned}
$$

The sequence of decisions and cuts that are generated by the algorithm is presented in figure 6.5. The algorithm commences with a greedy policy, whereby no water is stored in the reservoir. This leads to highly costly load shedding in stage 2, node 2, as well as stage 3, nodes 2 and 4 (since there is zero rainfall in these periods). In the first backward pass, the cuts generated in stage 2 are identical since the outcomes of stage 3 are identical, as seen from both nodes of stage 2[2]. The second forward pass utilizes hydro more conservatively in the first stage in order to protect against the eventuality where there is a shortage later. One new cut is generated in node 1 of stage 2, and one new cut in stage 1, no cut is added in node 2 of stage 2. The algorithm performs a third forward pass which produces identical primal decisions, thus resulting in no new optimality cuts in the backward pass. The optimality criterion is finally satisfied when the algorithm completes its third backward pass and verifies that no new cuts have been added to $NLDS(1)$. The optimal policy is unable to avoid load shedding in cases of low water supply (nodes 2 and 4 of stage 3), nevertheless the optimal policy is able to optimally manage the water resource by preventing spillage in periods of abundance (node 1 of stage 3).

The nested decomposition method is similar to the L-shaped method. There is, however, one important difference between the two algorithms. Whereas the L-shaped method is guaranteed to generate optimality cuts that support the value function, this is not the case for the nested L-shaped method. As shown in figure 6.6, the nested L-shaped method may produce optimality cuts that lie strictly below the value function. This can be understood intuitively by the fact that, at stage t, the value function approximation of V_{t+1,ω_t} is itself an under-estimate of the true value function. By contrast, the L-shaped method only extends over two time periods, therefore the under-estimation cannot back-propagate. For an example of this effect, the reader is referred to (Birge & Louveaux 2010), page 274.

Coming soon: one-dimensional example of nested decomposition.

Problems

6.1 Suppose that in proposition 6.1 the sets Ω_t are not necessarily finite. Show that $V_{t+1,\omega}(x_t)$ and dom $V_{t+1,\omega}(x_t)$ are convex.

6.2 Prove that equation (6.4) is a valid feasibility cut for $NLDS(t-1, A(t,k))$, meaning that the inequality is satisfied for all $x \in$ dom $V_{t-1,k}$.

6.3 Prove that equation (6.5) is a valid optimality cut for $NLDS(t-1, j)$, meaning that $V_{t-1,j}(x) \geq e_{t-1,j} - E_{t-1,j}x$ for any x.

6.4 Develop a table like table 6.1 for

- a fast-forward-fast-backward nested decomposition algorithm. A fast-forward-fast-backward algorithm moves as far as possible in the current direction.

[2] Note that the scenario tree of example 6.3 obeys serial independence. In chapter 7 it is shown how this can be exploited computationally in order to avoid redundant computation.

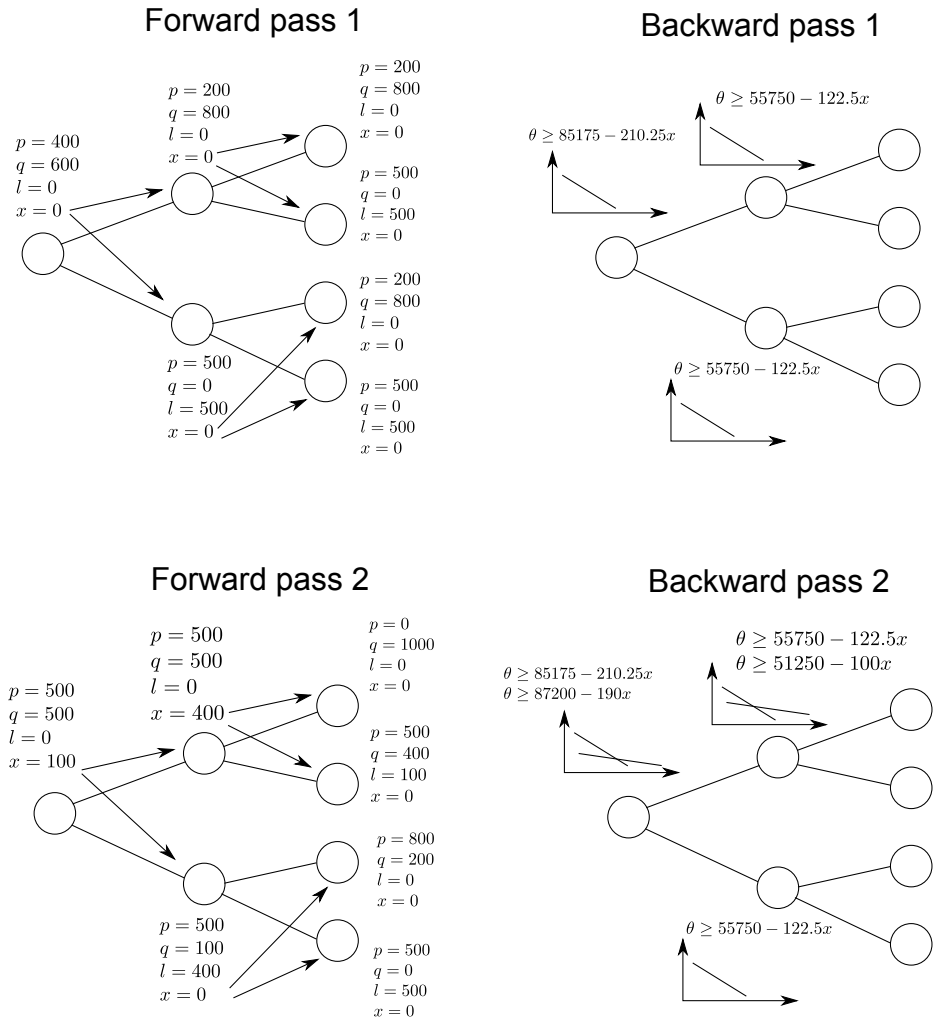


Figure 6.5 The progress of the nested decomposition algorithm of example 6.3.

- a fast-backward nested decomposition algorithm. A fast-forward algorithm moves forward whenever possible.
- a fast-backward nested decomposition algorithm. A fast-backward algorithm moves forward whenever possible.

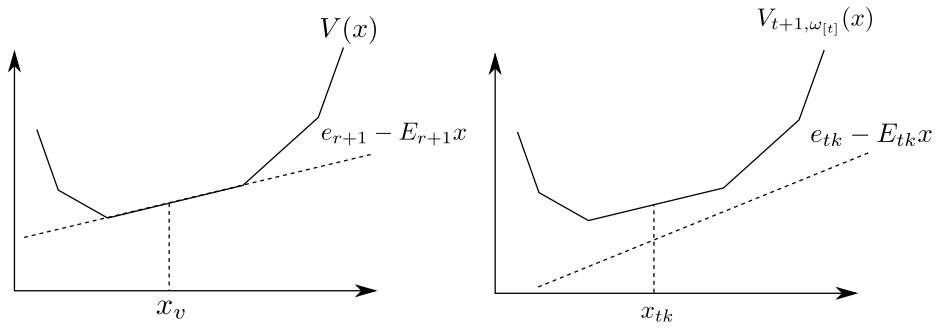


Figure 6.6 Whereas the L-shaped method is guaranteed to produce optimality cuts that support the value function (left figure), the nested L-shaped method may produce cuts that are strictly below the value function (right figure).

7 SDDP

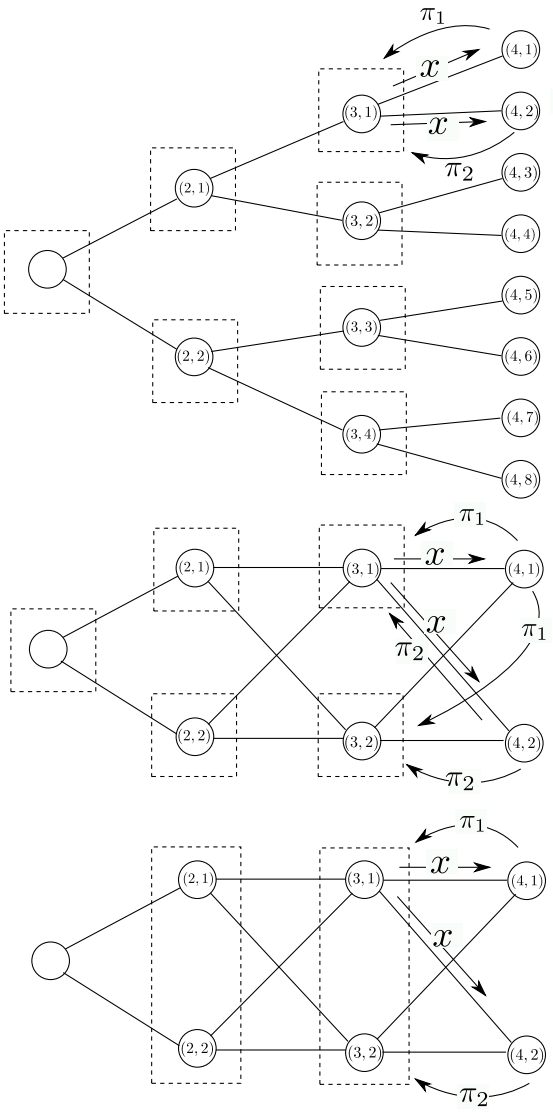
The previous chapter has built on the idea of the L-shaped algorithm and extended it to a multi-stage setting by developing the nested decomposition algorithm. Section 7.1 discusses the scalability of the nested decomposition algorithm, and some computational savings that can be achieved in the case of serial independence. Despite these computational savings, nested decomposition through an exhaustive search of scenario tree paths remains a non-scalable solution to multi-stage stochastic linear programming. Section 7.2 presents the stochastic dual dynamic programming algorithm, which overcomes the non-scalability of nested decomposition by resorting to Monte Carlo sampling. Monte Carlo sampling mitigates the issue of non-scalability, however it presents a challenge in terms of deciding on convergence. Section 7.3 discusses the termination criteria of SDDP. Finally, section 7.4 applies the SDDP algorithm to a hydrothermal scheduling example.

7.1 Drawbacks of Nested Decomposition

A major drawback of nested decomposition is the amount of linear programs that need to be solved in order for the algorithm to converge. The nested decomposition algorithm is not particularly efficient at sharing information between time stages. This is partly due to the model of uncertainty that the algorithm is designed for, which is a general multi-stage scenario tree. Denote as $|S_t|$ the number of realizations that can occur in stage t. Then nested decomposition will perform a forward pass that requires the resolution of $|S_1| + |S_1| \cdot |S_2| + \ldots = \sum_{t=1}^{H-1} \Pi_{j=1}^{t} |S_t|$ linear programs. In the backward pass, the algorithm will solve $\sum_{t=2}^{H} \Pi_{j=1}^{t} |S_j|$ linear programs.

This process is clearly non-scalable since the number of subproblems that need to be solved at each pass grows rapidly as the horizon of the problem increases. Moreover, a value function needs to be stored at each node of the scenario tree. The storage of value functions and the communication of information among sub-problems is depicted in figure 7.1. The dashed boxes indicate the storage of a value function. One value function is stored at each node of the scenario tree. The arrows indicate the communication of information. In order to generate a single cut in node 1 of stage 3, it is necessary to solve linear programs at nodes

Figure 7.1 Value function storage and cut sharing for different models of uncertainty: (i) a scenario tree, (ii) a lattice without serial independence, and (iii) a lattice with serial independence. Nodes are indexed by (t, k), where t is the time stage and $k = 1, \ldots, |\Xi_t|$ indexes the realization of ξ_t in stage t.



1 and 2 of stage 4. Even if nodes 3 and 4 of stage 4 correspond to *identical* outcomes as nodes 1 and 2 of stage 4, the nested decomposition algorithm needs to re-solve a linear program at nodes 3 and 4 of stage 4 in order to generate a cut for node 2 of stage 3.

A first step towards managing information more efficiently in multi-stage prob-

lems is to exploit the structure of a *lattice*, whenever possible, as shown in the middle of figure 7.1. Note the economy in computation that is achieved by the introduction of a lattice. The dual multipliers generated by the solution of nodes $(4, 1)$ and nodes $(4, 2)$ can now be used to generate *two* optimality cuts: one cut at node (3, 1), and one cut at node (3, 2).

Further economy in the storage of information can be achieved when the lattice satisfies the assumption of serial independence, i.e. the property whereby the distribution of uncertainty at a certain time stage is independent of past outcomes. If this is the case, then the value function at each time stage $V_{t+1,k}(x_t)$ becomes independent of the node k at stage t, and can be indexed only by time, $V_t(x_t)$. This is due to the fact that the expected future cost, which depends on future outcomes, does not depend on the node k that has actually materialized at the current time stage t. The consequence of this property is demonstrated in the last panel of figure 7.1. A single value function is now stored at each time stage, and the generation of optimality cuts at stage t requires solving $|S_{t+1}|$ linear programs in stage $t + 1$.

The **stochastic dual dynamic programming algorithm (SDDP)** is designed in order to overcome the bottlenecks associated with the forward and backward pass of the nested decomposition algorithm. The weaknesses of the nested decomposition algorithm are that *(i)* in the forward pass, all possible combinations of outcomes need to be enumerated, and *(ii)* in the backward pass, there is a loss of information in the generation of cuts, as noted in figure 7.1.

SDDP relies on two ideas for overcoming the drawbacks of nested decomposition: *(i)* The forward pass relies on a Monte Carlo simulation, instead of an enumeration of the possible outcomes. *(ii)* In the backward pass, dual multiplier information is shared in order to avoid solving subproblems repeatedly. The drawback of simulating outcomes in the forward pass, instead of enumerating them, is that a probabilistic upper bound is obtained, and this complicates the termination of the algorithm. This issue is discussed further in section 7.3. Serial independence delivers further economy in computation, however it more restrictive than a general lattice or scenario tree model.

7.2 The SDDP Algorithm

The following section presents the SDDP algorithm, as implemented on a lattice which does not necessarily satisfy serial independence.

As in the case of the nested decomposition algorithm, the SDDP algorithm consists of forward and backward passes. The algorithm is characterized by the number of Monte Carlo simulations that are performed in a *single* forward pass, which is denoted as K. During each forward pass, K Monte Carlo samples of the random process $(\xi_{1,i}, \ldots, \xi_{H,i})$, $i = 1, \ldots, K$, are generated. These samples dictate which node of the lattice will be visited during the forward pass. Each node of the lattice is associated with the following $NLDS$ (feasibility cuts are

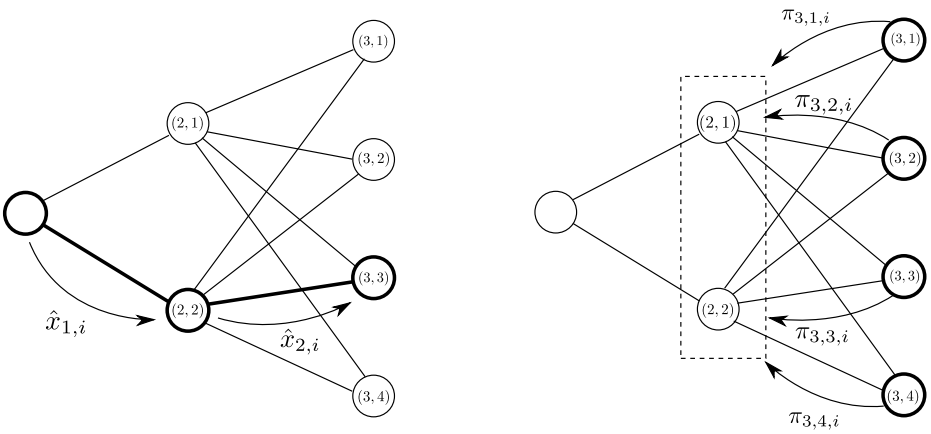


Figure 7.2 Forward (left panel) and backward (right panel) pass in SDDP. The forward pass of Monte Carlo sample i generates trial decisions $\hat{x}_{ti}$. The backward pass generates dual multipliers (π_{tki}, ρ_{tki}) that can be used for generating cuts for the previous time stage.

ignored):

$$\begin{aligned} & \min c_{t,k}^T x + \theta \\ (\pi) : & T_{t-1,k}\hat{x}_{t-1,i} + W_{t,k}x = h_{t,k} \\ (\rho) : & E_{t,k}x + \theta \cdot 1 \geq e_{t,k} \\ & x \geq 0 \end{aligned}$$

The vector $\hat{x}_{t-1,i}$ is a trial decision that is generated from the resolution of the $NLDS$ of the previous time period during Monte Carlo pass i. In this way, each forward pass of the algorithm generates K samples of trial decisions, $(\hat{x}_{1,i}, \ldots, \hat{x}_{H,i}), i = 1, \ldots, K$. The process is depicted graphically in figure 7.2.

At each Monte Carlo sample of a forward pass of the algorithm, the linear problem at stage $t = 1$ and another $H - 2$ linear problems are solved. Given K Monte Carlo simulations for each forward pass, this amounts to solving the root problem once, and another $H - 2$ linear problems K times, totaling $1 + K \cdot (H - 2)$ linear programs that need to be solved at each forward pass. For a given sequence of trial decisions, $(\hat{x}_{1,i}, \ldots, \hat{x}_{H,i})$, the backward pass requires the resolution of $\sum_{t=2}^{H} |\Xi_t|$ linear programs. Over K Monte Carlo trials, the backward pass amounts to the resolution of $K \cdot \sum_{t=2}^{H} |\Xi_t|$ linear programs. Thus, the computational effort of the algorithm scales linearly with respect to the horizon of the problem.

The SDDP algorithm can be described formally as follows:

DEFINITION 7.1 SDDP algorithm.

Forward pass:

- Solve $NLDS(1)$. Let x_1 be the optimal solution. Initialize $\hat{x}_{1,i} = x_1$ for $i = 1, \ldots, K$
- Repeat for $t = 2, \ldots, H-1$, $i = 1, \ldots, K$
  - Sample an outcome ξ_{ti} from the set Ξ_t
  - Solve $NLDS(t, i)$ with trial decision $\hat{x}_{t-1,i}$
  - Store the optimal solution as $\hat{x}_{t,i}$

Backward pass:

- Repeat for $t = H, H-1, \ldots, 2$
  - Repeat for $i = 1, \ldots, K$
    - Repeat for $k = 1, \ldots, |\Xi_t|$
      Solve $NLDS(t, k)$ with trial decision $\hat{x}_{t-1,i}$
    - For all $j = 1, \ldots, |\Xi_{t-1}|$, compute

$$E_{t-1,j,i} = \sum_{k=1}^{|\Xi_t|} p_t(k|j) \cdot \pi_{t,k,i}^T T_{t-1,k},$$

$$e_{t-1,i,j} = \sum_{k=1}^{|\Xi_t|} p_t(k|j) \cdot (\pi_{t,k,i}^T h_{t,k} + \rho_{t,k,i}^T e_{t,k})$$

    - Add the optimality cut

$$E_{t-1,j,i} x + \theta \geq e_{t-1,j,i}$$

to *every* $NLDS(t-1, j)$, $j = 1, \ldots, |\Xi_{t-1}|$

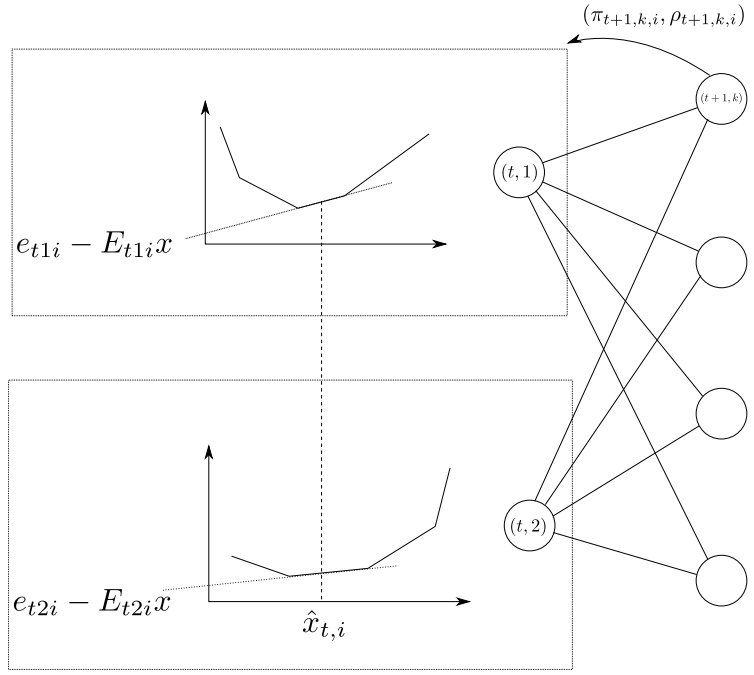
There is a learning attribute to the SDDP algorithm. The idea of the algorithm is to learn the value function of each time stage over iterations, and to eventually 'hone in' on the neighborhood of near-optimal decisions. Thus, the choice of the Monte Carlo simulation number K is an important feature of the design of the algorithm. The trade-off involved in the choice of K is that a larger value implies accelerated learning of the value function, at the cost of *(i)* increasing the number of linear programs that need to be solved at each forward-backward pass, and *(ii)* increasing the size of the $NLDS$ subproblems more rapidly[1]. The role of the forward pass in the SDDP algorithm is to determine the point $\hat{x}_{t,i}$ at which the value function should be explored, as shown in figure 7.3.

The parameter K can change over iterations. For example, the parameter may increase over iterations, since early iterations will anyways lead to short-sighted trial decisions $\hat{x}_{t,i}$, and there is little gain from exploring the value function in the neighborhood of such poor decisions.

Serial independence implies a common value function for all nodes of a given time stage. Regardless of serial independence, the dual multipliers of stage $t+1$ can be used for generating cuts for stage t as long as multiplier $\pi_{t+1,k,j}$ is weighted according to $p_{t+1}(k|j)$ when generating the optimality cut for node j of stage t.

[1] In practice, cuts can be dropped from the $NLDS$ subproblems based on various criteria (e.g. if a cut has not been active for more than a certain number of iterations).

Figure 7.3 The value function approximation at node j of stage t is explored at $\hat{x}_{t,i}$, the trial point generated in Monte Carlo sample i. The backward pass of Monte Carlo sample i provides dual multipliers $(\pi_{t+1,k,i}, \rho_{t+1,k,i})$ that are used for generating cut $e_{t,j,i} - E_{t,j,i}x$ at each node $j = 1, \ldots, |\Xi_t|$.



The SDDP algorithm is particularly well suited for handling uncertainty that appears in the right-hand side parameters h in the form of an auto-regressive process:

$$h_t = c + \sum_{j=1}^{p} \phi_j \cdot h_{t-j} + \epsilon_t,$$

where ϵ_t are independent, identically distributed variables. In particular, the value of the process in previous time stages can be incorporated into the state vector x_t while preserving the structure of the $NLDS$. This idea is demonstrated in section 7.4.

7.3 Termination

A challenge that is raised by the introduction of Monte Carlo simulation for the forward pass of the SDDP algorithm is the convergence criterion: it is not clear when the algorithm should be terminated. As SDDP is a learning-based algorithm, it will eventually build an approximation of the value functions at each

stage in the neighborhood of 'interesting', or near-optimal, decisions. As a first step towards reasoning about termination, it is useful to consider how the SDDP algorithm would behave if value functions had been approximated *exactly* at each $NLDS$ (i.e. if all cutting planes that define the value functions were present). In such a situation, each Monte Carlo sample of a forward pass provides a sequence $(\hat{x}_{1,i}, \ldots, \hat{x}_{H,i})$ of optimal reactions to the realized uncertainty. Then, the total cost of the forward pass:

$$z_i = \sum_{t=1}^{H} c_{t,i}^T \hat{x}_{t,i}$$

provides a sample of the average cost of the optimal policy. The forward pass consists of K *independent* sample paths of uncertainty, and therefore K independent samples of the cost of the optimal policy. It is natural to average these independent samples in order to estimate the true average cost of the optimal policy:

$$\bar{z} = \frac{1}{K} \sum_{i=1}^{K} z_i$$

Indeed, as K increases, the average cost estimate $\bar{z}$ converges to the true average cost of the optimal policy. In fact, a stronger statement can be made about the behavior of the sample average. The estimate $\bar{z}$, which is itself a random variable (since it depends on the K Monte Carlo samples that are selected at random), behaves as a Gaussian random variable centered around the true average cost. As intuition suggests, the variance of this Gaussian distribution decreases as the number of Monte Carlo samples K increases:

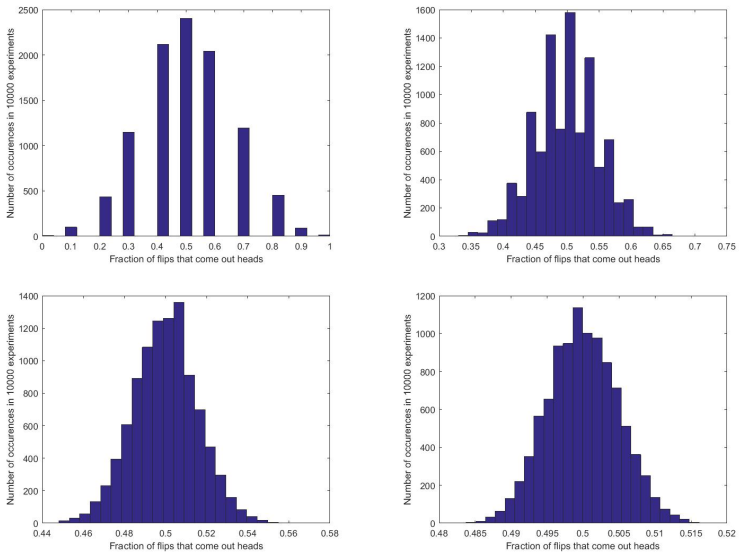
THEOREM 7.2 *Central Limit Theorem. Suppose* $\{X_1, X_2, ...\}$ *is a sequence of independent, identically distributed random variables with* $\mathbb{E}[X_i] = \mu$ *and* $Var[X_i] = \sigma^2 < \infty$. *Then*

$$\sqrt{n}\left(\left(\frac{1}{n}\sum_{i=1}^{n} X_i\right) - \mu\right) \xrightarrow{d} N(0,\ \sigma^2).$$

where $N(\mu, \sigma^2)$ *denotes a normal distribution with mean* μ *and variance* σ^2.

Example 7.1 Consider conducting the following experiment: flip a coin K times and count the fraction of times that the result is heads. The central limit theorem predicts that as K increases, the distribution of the resulting fraction will converge towards a Gaussian distribution centered around the probability of getting heads, which is 0.5 for a fair coin. Figure 7.4 presents the histogram of this estimate for various values of K.

Figure 7.4 Illustration of the central limit theorem in example 7.1. The histogram (over 10000 experiments) of the fraction of 'heads' obtained over K coin flips for $K = 10$ (upper left), $K = 100$ (upper right), $K = 1000$ (lower left) and $K = 10000$ (lower right).



The argument presented above is developed in the case where the value functions are exactly represented in all $NLDS$ of all time stages, but it applies equally well in the case where the $NLDS$ consist of only a subset of the cuts that define the true value functions. If the value functions are not represented exactly, then the resulting reactions $(\bar{x}_1, \ldots, \bar{x}_H)$ are feasible, but not necessarily optimal. In this case, the estimate $\bar{z}$ provides an estimate of the average cost of a feasible, not necessarily optimal, policy. Thus, $\bar{z}$ is a probabilistic upper bound.

Recall that the optimality cuts of $NLDS(1)$ determine an under-approximation of $V_2(x)$, and the feasibility cuts define a superset of dom V_2. Thus, $NLDS(1)$ is a relaxation of the actual multi-stage stochastic program. The SDDP therefore also provides a lower bound $\underline{z}$ during the forward pass, which is obtained as the objective function of $NLDS(1)$.

Given enough Monte Carlo samples K for the forward pass, it is possible to compute a confidence interval $(\eta_{a/2}, \eta_{100-a/2})$ for the upper bound of the problem. It could then be reasonable to terminate the algorithm if the lower bound z_{LB} is within this confidence interval, since in this case the lower and upper bound are close with high confidence. This inspires the termination criterion originally proposed by Pereira and Pinto: terminate the algorithm when

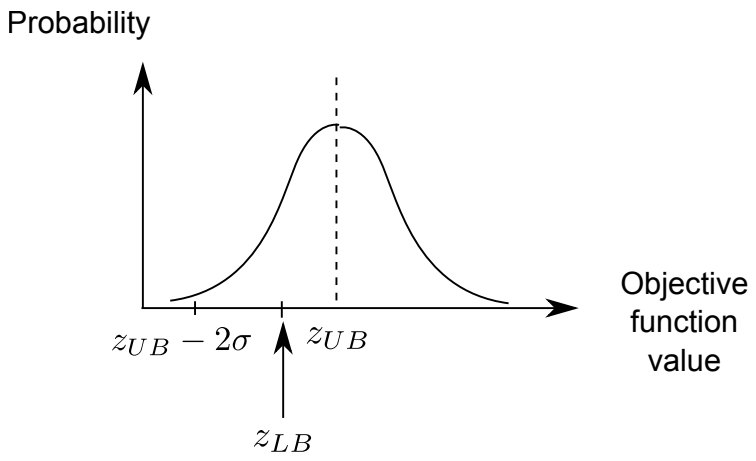


Figure 7.5 A graphical illustration of the termination criterion proposed by Pereira and Pinto. The SDDP algorithm is terminated when $z_{LB} \in (\bar{z} - 2\sigma, \bar{z} + 2\sigma)$, where $(\bar{z} - 2\sigma, \bar{z} + 2\sigma)$ is the 95.4% confidence interval of $\bar{z}$.

$z_{LB} \in (\bar{z} - 2\sigma, \bar{z} + 2\sigma)$, where

$$\sigma = \sqrt{\frac{1}{K^2} \sum_{k=1}^{K} (z_k - \bar{z})^2}$$

is the sample standard deviation of the average cost. The interval $(\bar{z} - 2\sigma, \bar{z} + 2\sigma)$ is the 95.4% confidence interval of $\bar{z}$. The termination criterion is illustrated graphically in figure 7.5.

The criterion of Pereira and Pinto may not perform well in practice due to a number of reasons: *(i)* The value of K needed to satisfy the criterion may be excessively large. *(ii)* The standard deviation σ may be excessively high in early stages of the algorithm due to the fact that certain outcomes lead to very high costs. These high costs are a result of the fact that the algorithm reaches poor decisions when the value function approximations are inaccurate at early stages. The above termination criterion would then terminate the algorithm, whereas in fact the algorithm has not converged. These drawbacks are illustrated in the example of section 7.4.

The formal explanation of why Pereira's termination criterion may not work is explained in detail by Shapiro (Shapiro 2011) in remark 1. *The meaning of the confidence interval* $[\bar{z} - z_{\alpha/2} s/\sqrt{K}, \bar{z} + z_{\alpha/2} s/\sqrt{K}]$ *is that with (approximate) probability* $1 - \alpha$ *some number inside this interval gives an upper bound for the optimal value of the [...] problem. However, this upper bound could be any point of that interval and this will be too optimistic to assume that the lower end of the confidence interval gives such an upper bound. According to that criterion the larger the variance* s^2 *is, the sooner we stop. Moreover, increasing the confidence* $1 - \alpha$ *makes the critical value* $z_{\alpha/2}$ *bigger, and the lower end of the confidence in-*

terval can be made arbitrary small for sufficiently large confidence $1-\alpha$*. That is, by increasing the confidence* $1-\alpha$ *the procedure could be stopped at any iteration by this stopping criterion; this does not make sense. A more meaningful criterion would be to stop the procedure if the difference between the upper confidence bound* $\bar{z} + z_{\alpha/2}s/\sqrt{K}$ *and the lower bound* $\underline{z}$ *is less than a prescribed accuracy level* $\epsilon > 0$*. This will suggest, with confidence of about* $1-\alpha$*, that the [...] problem is solved with accuracy* ϵ*.*

Example 7.2 Suppose that one wishes to achieve a 1% optimality gap with 95.4% confidence. It is necessary to choose K such that $2\sigma = 0.01 \cdot \bar{z}$. In order to determine the number of Monte Carlo samples K that achieve the desired gap, it is necessary to know (or be able to estimate) the mean $\bar{z}$ and standard deviation s of the cost of the optimal policy.

$$s = \sqrt{\frac{1}{K}\sum_{k=1}^{K}(z_k - \bar{z})^2} \Rightarrow \sigma = \frac{1}{\sqrt{K}}s$$

The required number of samples K that achieve the desired optimality gap with the desired confidence is then obtained as follows:

$$K = (\frac{2 \cdot s}{0.01 \cdot \bar{z}})^2.$$

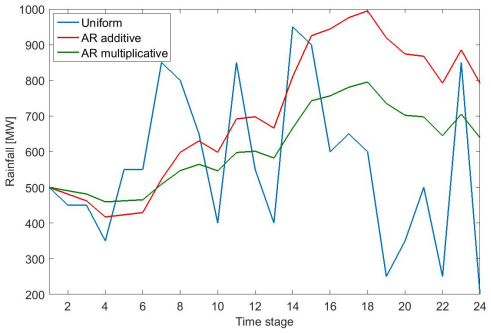
Using this formula to decide on the necessary value of K for running the algorithm often leads to a value which is too large for running the algorithm in a reasonable amount of time.

Alternative convergence criteria may be envisioned, based on the stabilization of the mean $\bar{z}$ standard deviation σ. A stabilization of the mean cost and its standard deviation indicates that the algorithm has 'learned' the value functions and reaches the same decisions over the same samples $(\xi_{1i}, \ldots, \xi_{Hi})$.

7.4 Example: Application of SDDP to Hydrothermal Scheduling

This section applies the SDDP algorithm to the hydrothermal scheduling problem. Suppose that there is a demand of 1000 MW in the system at each period. Thermal production is available at a marginal cost of 25 \$/MWh. Thermal capacity is equal to 500 MW. The cost of not serving demand amounts to 1000 \$/MWh. A hydroelectric dam can store up to 1000 MWh of energy per hour. The planning is performed over a horizon of 24 months with a time step of 1 month.

Figure 7.6 Three sample trajectories of rainfall, according to the models of section 7.4.1. All three processes start from an initial rainfall value of 500 MW. The blue trajectory corresponds to a uniform serially independent model. The red trajectory corresponds to an additive autoregressive model. The green trajectory corresponds to a multiplicative autoregressive model. All three models use the same underlying lattice of 24 stages, with 20 nodes per stage. Whereas all three samples correspond to the same underlying trajectory on the lattice, the actual rainfall outcomes are quite different.



The initial level of the hydro reservoir, normalized to hourly energy availability[2], is equal to 700 MWh.

The problem is solved for three models of rainfall uncertainty: *(i)* independent uniformly distributed rainfall between time stages in section 7.4.2, *(ii)* an additive autoregressive model of rainfall in section 7.4.3, and *(iii)* a multiplicative autoregressive model of rainfall in section 7.4.3. These models are presented in the following section.

7.4.1 Rainfall Models

Consider a model of rainfall over 24 stages, where each stage represents a month. The three alternative models of rainfall (uniform, autoregressive additive, and autoregressive multiplicative) are developed by using a lattice which obeys serial independence. This lattice is built by defining a random variable w_t which is uniformly distributed in the set $\{1, 2, \ldots, 20\}$.

Suppose that the rainfall follows a uniform distribution over the interval [0, 1000] MW. Then this model of rainfall can be approximated by assigning to each (t, k) of the lattice the following rainfall value:

$$R_{t,k} = 50 \cdot w_{t,k}.$$

Now consider the alternative case where rainfall is assumed to follow an addi-

[2] This implies that the dam carries a total energy of $700 \cdot 24 \cdot 30$ MWh at the beginning of the horizon. For the sake of notational convenience all numerical values are normalized by hour.

tive autoregressive process, according to the following model:

$$R_t = \phi \cdot R_{t-1} + \sigma \cdot \epsilon_t,$$

where ϕ is an autoregressive parameter, ϵ_t are independent identically distributed random variables that obey a standard normal distribution, and σ corresponds to the standard deviation of the random input.

Consider the following choice of parameters for the additive autoregressive model: $c = 0$, $\phi = 1$, $\sigma = 100$ MW, and an initial rainfall level of 500 MW. Then the 24-stage, 20-node lattice can be used for approximating additive autoregressive rainfall, according to the following assignment of rainfall values in each node (t, k) of the lattice:

$$R_{t,k} = R_{t-1} + 100 \cdot \Phi^{-1}(\frac{w_{t,k}}{20} - 0.025),$$

where Φ is the cumulative distribution function of a standard normal variable and w_t is uniformly distributed in the interval $\{1, 2, \ldots, 20\}$.

Finally, consider using the same lattice for approximating multiplicative autoregressive rainfall of the following form:

$$R_t = (c + \phi \cdot R_{t-1}) \cdot \eta_t,$$

where ϕ is an autoregressive parameter, and η_t are independent identically distributed random variables that obey a normal distribution of mean 1 and standard deviation σ.

Consider the following choice of parameters: for the multiplicative autoregressive model: $c = 0$, $\phi = 1$, $\sigma = 0.1$ MW, and an initial rainfall level of 500 MW. In order to approximate the multiplicative autoregressive model on the 24-stage, 20-node lattice, the following assignment of rainfall values in each node (t, k) of the lattice can be used:

$$R_{t,k} = R_{t-1} \cdot (1 + 0.01 \cdot \Phi^{-1}(\frac{w_{t,k}}{20} - 0.025)).$$

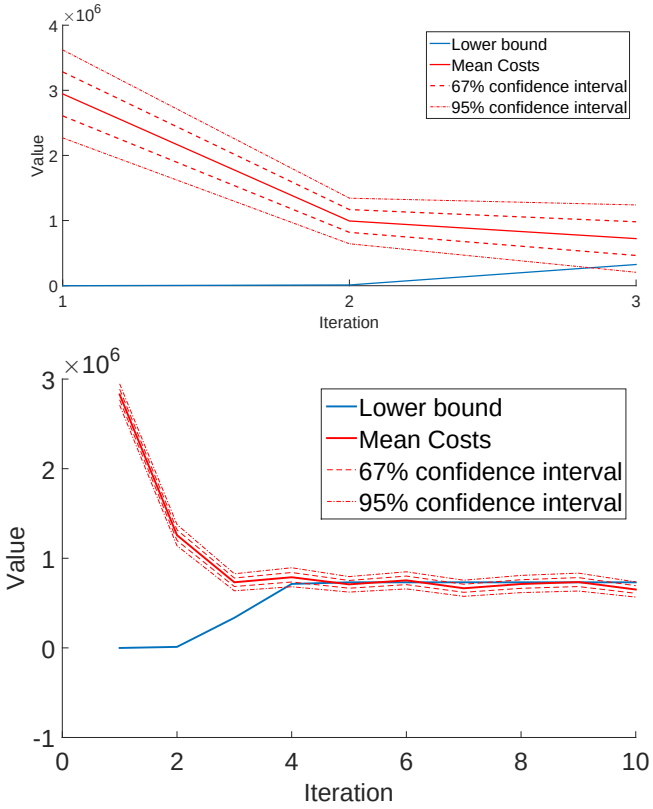
A sample outcome for each of the three rainfall models is shown in figure 7.6. Note that the underlying lattice is *identical* for all three models. However, the rainfall trajectory is quite different. Figure 7.6 compares the trajectory of rainfall for each of the models, given an *identical* sample trajectory of the underlying noise, $w_{[t]}$.

7.4.2 Hydrothermal Scheduling with Serially Independent Rainfall

In this section it is assumed that rainfall in each period is described by independent, identically distributed random variables that follow a uniform distribution over the interval [0, 1000] MW. The model is discretized using 20 nodes per stage, as shown in the previous section.

Denote p as the thermal production, q as the hydro production, l as the unserved demand, and x_t as the amount of stored hydro energy at the *beginning* of

Figure 7.7 The upper and lower bounds of the SDDP algorithm for the model of section 7.4.2 for $K = 10$ samples in the forward pass (upper panel) and $K = 150$ samples in the forward pass (lower panel).



period t. Rainfall is a random variable, denoted as $R_{t,k}$ for the node of stage t, given realization k. The $NLDS$ for a given stage and outcome can be expressed as follows:

$$\begin{aligned} NLDS(t,k) = \min\ & 1000 \cdot l + 25 \cdot p \\ \text{s.t. } & l + p + q \geq 1000 \\ & p \leq 500 \\ & q \leq x_{t-1} + R_{t,k} \\ & x = x_{t-1} + R_{t,k} - q \\ & x \leq 1000 \\ & l, p, q, x \geq 0 \end{aligned}$$

In order to illustrate the risk of premature convergence using Pereira's proba-

bilistic criterion, we illustrate a case where the algorithm terminates at a point where the upper bound has not actually stabilized.

Consider running the algorithm with $K = 10$ samples in the forward pass and a confidence interval of $(\bar{z} - 2\sigma, \bar{z} + 2\sigma)$. The algorithm converges after three iterations, as shown in figure 7.7. Upon convergence, the lower bound is reported as being equal to 324,594 \$. The upper bound is reported as being equal to 722,350 \$ and the standard deviation is estimated to be equal to $s = 817{,}923$ \$.

However, upon performing a forward pass with 200 samples *after* the algorithm has converged, the upper bound is estimated to be $\bar{z} = 770{,}440$ \$. A new and more accurate lower bound is estimated, which is equal to $\underline{z} = 657{,}697$ \$. The estimated standard deviation amounts to $s = 602{,}680$ \$. Note that the lower bound estimated with 200 samples is notably different from what is estimated with 10 samples. In fact, the confidence interval estimated in the forward pass $(\bar{z} - 2\sigma, \bar{z} + 2\sigma)$ with the values estimated from the 200-sample forward pass is (685,208 \$, 855,672 \$). The lower bound estimated in the previous paragraph is far from being close to this confidence interval.

We now use the results from the run of the algorithm with $K = 150$ samples ($s = 602,680$ \$ and $\bar{z} = 770{,}440$ \$) in order to estimate the number of iterations that are needed in order to achieve a certain optimality gap with a certain level of confidence. From example 7.2 it is possible to use the following formula in order to obtain the required number of Monte Carlo samples K so as to achieve a 15% optimality criterion with 95% confidence:

$$K = \left(\frac{2 \cdot s}{0.15 \cdot \bar{z}}\right)^2 = \left(\frac{2 \cdot 602,680}{0.15 \cdot 770,440}\right)^2 \simeq 109$$

Using this estimate of forward pass samples, SDDP is run again with $K = 150$ samples for the forward pass. The algorithm satisfies the probabilistic convergence criterion that $\underline{z} \in (\bar{z} - 2\sigma, \bar{z} + 2\sigma)$ within 4 iterations. The convergence is presented in figure 7.7. Although the confidence interval of $\bar{z}$ narrows significantly compared to the run with $K = 10$ samples in the forward pass, the run time of the algorithm increases by an order of magnitude due to the higher number of samples K in the forward pass. The evolution of the algorithm is presented for 10 iterations, indeed one verifies that the algorithm has converged after 4 iterations (as opposed to the three iterations that were required when K was set equal to 10 in figure **??**).

The value function approximations for $t = 1$ and $t = 12$ are plotted in figure 7.8. As expected, the value function of earlier time periods is greater than that of later periods for the same reservoir level. Note that whereas the slope of V_1 remains constant, this is not the case for V_{12}.

Figure 7.9 presents the optimal dispatch policy for a given sample rain realization. The optimal policy attempts to minimize the amount of water that is wasted. Therefore, in periods of excessive rainfall, the hydro contribution to the total production is increased, the thermal production is suppressed, and the

Figure 7.8 The value functions for the model of section 7.4.2 with $K = 150$ samples in the forward pass for $t = 1$ and $t = 12$.

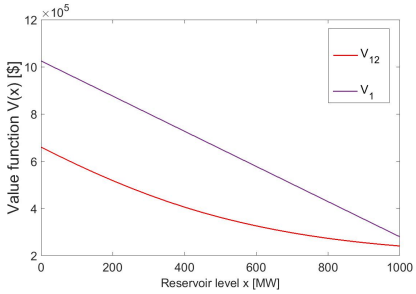
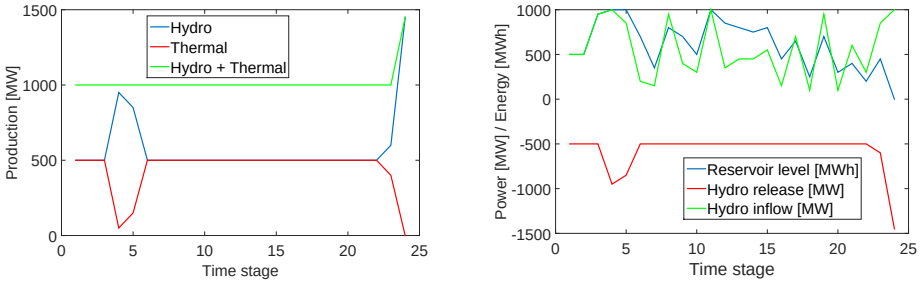


Figure 7.9 The dispatch of hydro and thermal power (left panel) and the management of the reservoir level (right panel) for the model of section 7.4.2.

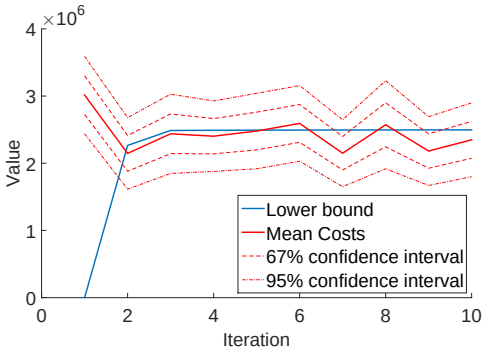


reservoir is filled to its max capacity. On the other hand, certain samples result in load shedding. Note that this is somewhat infrequent, since for this to occur it is necessary for the initial level of water to be depleted through a sequence of unfavorable rainfall outcomes. Although the values used in the example are not representative of a typical system (since the risk of not serving demand is excessively high in the present example), they are intended to highlight the trade-offs that the hydro management policy faces by assigning a non-negligible risk to the possibility of not serving demand.

7.4.3 Hydrothermal Scheduling with Additive Autoregressive Rainfall

In order to solve the hydrothermal scheduling problem with an autoregressive model for rainfall, it is necessary to add a new state variable r_t which denotes the rainfall that occurs in the *beginning* of period t. The remaining notation is identical to the previous section. The $NLDS$ for a given stage and outcome can

Figure 7.10 The convergence of the SDDP algorithm for the model of section 7.4.3, with $K = 150$ and a 95% confidence interval.



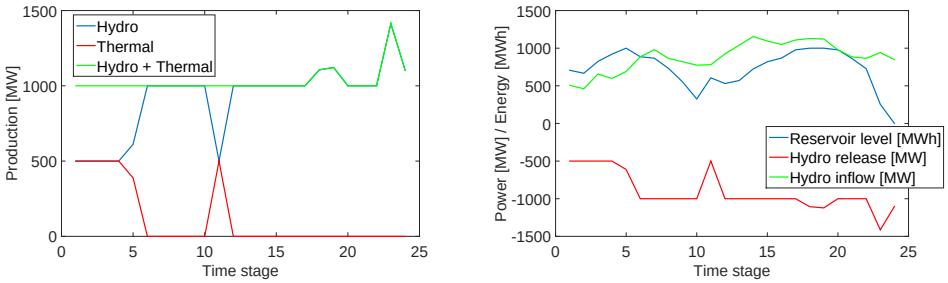
be expressed as follows:

$$\begin{aligned} NLDS(t,k) = \min\ & 1000 \cdot l + 25 \cdot p \\ \text{s.t. } & l + p + q \geq 1000 \\ & p \leq 500 \\ & q \leq x_{t-1} + r \\ & x = x_{t-1} + r - q \\ & x \leq 1000 \\ & r \leq r_{t-1} + h_{t,k} \\ & l, p \geq 0 \end{aligned}$$

where w_{tk} is distributed uniformly on $\{1, \ldots, 20\}$ and $h_{t,k} = 100 \cdot \Phi^{-1}(\frac{w_{t,k}}{20} - 0.025)$, as in the additive autoregressive model of section 7.4.1. Note that the non-negativity of x, q, r has been lifted in order to ensure that the $NLDS$ is feasible. A refined rendering of the $NLDS$ in order to accurately account for the fact that the rainfall can become negative due to the autoregressive model of rain is presented in section 7.4.4. The SDDP algorithm with $K = 150$ and a confidence interval of $(\bar{z} - 2\sigma, \bar{z} + 2\sigma)$ converges within 2 iterations, as shown in figure 7.10.

Figure 7.11 presents the optimal dispatch policy for a given sample rain realization. Notice that in this favorable rain outcome the optimal policy is attempting to balance the minimization of the use of the thermal generator with the risk of depleting the reservoir capacity. Also notice the autoregressive behavior of the rainfall, which results in a noticeably different profile of rainfall compared to the previous section. This also affects the optimal dispatch policy, since the optimal policy will tend to use the reservoir more aggressively in periods of high rainfall, since these periods are likely to be succeeded by further periods of high rainfall. One weakness of this model, however, is the fact that under unfavorable rainfall

Figure 7.11 The dispatch of hydro and thermal power (left panel) and the management of the reservoir level (right panel) for the case of autoregressive rainfall.



outcomes it is possible that the hydro production q and the rainfall r become negative. An alternative to overcome this difficulty is to develop a multiplicative autoregressive model, as discussed in the following section.

7.4.4 Hydrothermal Scheduling with Multiplicative Autoregressive Rainfall

In order to overcome the drawback of the model of the previous section, which can result in negative rainfall, one can develop a multiplicative autoregressive model. Note that the multiplicative autoregressive model results in non-negative rainfall provided an appropriate calibration of the model parameters.

The $NLDS$ for a given stage and outcome can be expressed as follows[3]:

$$\begin{aligned} NLDS(t,k) = \min\ & 1000 \cdot l + 25 \cdot p \\ \text{s.t. } & l + p + q \geq 1000 \\ & p \leq 500 \\ & q \leq x_{t-1} + r \\ & x = x_{t-1} + r - q \\ & x \leq 1000 \\ & r = r_{t-1} \cdot h_{t,k} \\ & l, p, q, r, x \geq 0 \end{aligned}$$

where w_{tk} is distributed uniformly on $\{1, \ldots, 20\}$ and $h_{t,k} = 1 + 0.1 \cdot \Phi^{-1}(\frac{w_{t,k}}{20} - 0.025)$, as in the multiplicative autoregressive model of section 7.4.1. Note that x, q, r can now be imposed to be non-negative, since the values of the uncertainty on the lattice ensure that the rainfall never becomes negative. The SDDP algorithm with $K = 150$ and a confidence interval of $(\bar{z} - 2\sigma, \bar{z} + 2\sigma)$ converges within

[3] For the results presented in this section, the initial rainfall is set to 700 MW.

Figure 7.12 The convergence of the SDDP algorithm for the model of section 7.4.4, with $K = 150$ and a 95% confidence interval.

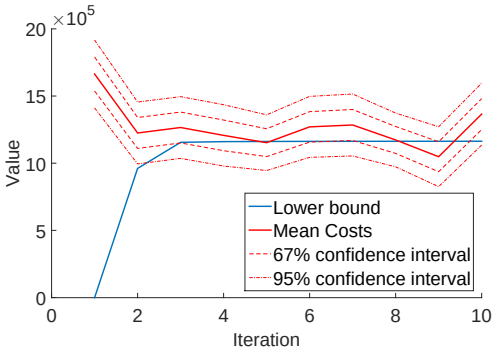
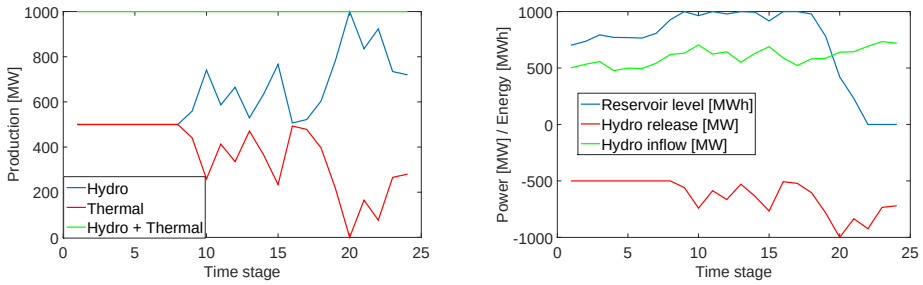


Figure 7.13 The dispatch of hydro and thermal power (left panel) and the management of the reservoir level (right panel) for the case of multiplicative autoregressive rainfall.



3 iterations, as shown in figure 7.12. Figure 7.13 presents the optimal dispatch policy for a given sample rain realization.

Bibliography

Chapter 7.2. The SDDP algorithm was introduced by (Pereira & Pinto 1991).
Chapter 7.3. The termination criterion of the SDDP algorithm is analyzed by (Shapiro 2011).
Chapter 7.4. The inclusion of autoregressive models of uncertainty in the SDDP framework is analyzed by (Infanger & Morton 1996). Multiplicative autoregressive models were proposed by (Cabral 2016).

8 Performance of Stochastic Programming Solutions

Since the development of algorithms that can handle stochastic programs is a non-trivial task, it is useful to know what the benefits are of adopting a more sophisticated model of uncertainty such as stochastic programming. Two natural alternatives that could be checked are: (i) what performance could be attained if the future were known in advance, and (ii) what performance could be attained by a simpler policy (e.g. the policy that emerges if uncertain parameters are replaced by their expected value)?

This chapter presents results regarding the performance of alternatives to stochastic programming. These performance differences are useful to estimate in any analysis of an optimization problem under uncertainty, whether stochastic programming is employed or not. Sections 8.1 and 8.2 introduce metrics of performance in a two-stage setting, and are focused respectively on the expected value of perfect information and the value of the stochastic solution. Section 8.3 derives results that rank these metrics of performance. In multiple periods, the generalization of these metrics of performance and their ranking is not straightforward. Even the numerical estimation of performance becomes challenging in a multi-period setting. The focus of section 8.4 is on the estimation of performance in a multi-period setting through simulation.

8.1 Expected Value of Perfect Information

Recall the two-stage stochastic linear program:

$$\begin{aligned}
&\min c(\omega)^T x + \mathbb{E}[q(\omega)^T y(\omega)] \\
&\text{s.t. } Ax = b \\
&T(\omega)x + W(\omega)y(\omega) = h(\omega), \omega \in \Omega \\
&x \geq 0, y(\omega) \geq 0, \omega \in \Omega
\end{aligned}$$

where $\omega \in \Omega$ with Ω a set of outcomes. Although the set Ω is typically treated as being finite in these notes, this assumption is not necessary for deriving the results presented in this chapter. Recall that $x \in \mathbb{R}^{n_1}$ is the vector of first-stage decisions, and $y(\omega) \in \mathbb{R}^{n_2}, \omega \in \Omega$ is the vector of second-stage decisions. The random vector $\xi(\omega)$ is used to denote all random parameters in the problem:

$\xi^T = (c^T, q^T, T_1^T, \ldots, T_{n_1}^T, W_1^T, \ldots, W_{n_2}^T, h^T)$, where W_i denotes the i-th column of W (likewise for T_i).

For two-stage stochastic programs, the following sets can be defined: $K_1 = \{x|Ax = b, x \geq 0\}$ is the set of first-stage decisions that respect first-stage constraints, and $K_2(\omega) = \{x|\exists y : W(\omega)y = h(\omega) - T(\omega)x\}$ is the set of first-stage decisions that have feasible second-stage reactions for a certain outcome $\omega \in \Omega$.

In what follows, it is useful to define the following function $z(x, \xi)$:

$$z(x, \xi) = c^T x + Q(x, \xi) + \delta(x|K_1),$$

where $\delta(x|K_1)$ is the indicator function[1] for set K_1 and

$$Q(x, \xi) = \min_y \{q(\omega)^T y | W(\omega)y = h(\omega) - T(\omega)x\}.$$

The function $z(x, \xi)$ is the cost incurred if one were to optimize only for scenario ω, where this cost is parametrized by x. Note the difference between $z(x, \xi)$ and $Q(x, \xi)$, since z accounts for first-stage cost. It is possible that $z(x, \xi) = +\infty$ (if $x \notin K_1 \cap K_2(\omega)$), or that $z(x, \xi) = -\infty$ (if z is unbounded below).

The **wait-and-see value** is the expected value of reacting with perfect foresight $x^\star(\xi)$ to any outcome ω:

$$\begin{aligned} WS &= \mathbb{E}[\min_x z(x, \xi)] \\ &= \mathbb{E}[z(x^\star(\xi), \xi)]. \end{aligned}$$

The **here-and-now value** is the expected value of the two-stage stochastic program:

$$SP = \min_x \mathbb{E}[z(x, \xi)].$$

Note that both WS and SP have been defined in terms of the function $z(x, \xi)$, and that their only difference is that the minimization and expectation operators have been swapped. In terms of computation, it should be noted that the computation of SP requires the solution of a stochastic program, whereas the computation of WS requires the solution of $|\Omega|$ linear programs, where $|\Omega|$ is the number of second-stage outcomes. Therefore, computing SP is generally more difficult than computing WS.

The **expected value of perfect information** is the difference between the stochastic programming solution and the wait-and-see value:

$$EVPI = SP - WS.$$

$EVPI$ can be interpreted as the additional benefit that could be obtained, relative to stochastic programming, if a perfect forecast of the future were possible.

[1] The indicator function is defined as $\delta(x|K) = 0$ if $x \in K$ and $\delta(x|K) = +\infty$ otherwise.

Table 8.1 The optimal investments (in MW) for SP, WS and EV in examples 8.1 and 8.2

Technology	SP solution	Reference	10x wind	EV solution
Coal	5085	1918	3410	4235
Gas	1311	2165	2986	3261
Nuclear	3919	7086	3919	2905
Oil	854	0	0	0

Example 8.1 Consider the capacity expansion planning problem of section 1.2. Recall that the goal is to determine an optimal mix of generation, however the load that will materialize is uncertain. The investment and operating cost of the generation technologies is available in table 1.1. The two possible realizations of the load duration curve are presented in table 1.2, where the reference scenario occurs with probability 10% and the 10x wind scenario occurs with probability 90%. It is shown in section 1.2 that $SP = 340316$ \$/h. In order to determine $EVPI$, it is necessary to determine WS. Solving the capacity expansion planning problem for the reference and 10x scenario results in a cost of 382288 \$/h and 329383 \$/h respectively, resulting in an average cost of $WS = 334673$ \$/h. As expected, the cost in the 10x wind scenario is lower since demand is reduced. The $EVPI$ amounts to 5643 \$/h, which is 1.7% of the SP solution. The stochastic programming solution performance is therefore quite close to the performance that could be attained if the future could be forecast perfectly. The investment decisions in the case of the stochastic programming and wait-and-see solution are shown in table 8.1. Note how the wait-and-see model adapts the first-stage choice to the realization of demand, and never chooses oil.

8.2 The Value of the Stochastic Solution

Another natural benchmark for comparing the performance of stochastic programming solutions are simpler decision-making rules. One such decision rule is to solve the **expected (or mean) value problem**:

$$EV = \min_x z(x, \bar{\xi}),$$

where $\bar{\xi} = \mathbb{E}[\xi]$ is the expected value of the random parameters.

Denote $x^\star(\bar{\xi})$ as the expected value solution, which is the optimal solution of the expected value problem. The derivation of $x^\star(\bar{\xi})$ amounts to solving a linear program with a single 'scenario', the average scenario. It may well be the case that $\bar{\xi} \notin \cup_{\omega\in\Omega}\{\xi(\omega)\}$. This linear program is smaller than the full stochastic program, and is therefore more attractive computationally. However, there is

no reason to believe that the average performance of such a first-stage decision should be good, unless there is an independence between the realizations of the random parameters $\xi(\omega)$ and the optimal first-stage decision contingent on those realizations, $x^\star(\xi)$ (see exercise 8.1).

The **expected value of using the EV solution** measures the performance of $x^\star(\bar{\xi})$, where performance is measured by reacting optimally in the second stage *given* first-stage decision $x^\star(\bar{\xi})$:

$$EEV = \mathbb{E}[z(x^\star(\bar{\xi}), \xi)]$$

The **value of the stochastic solution** is then defined as the benefit of the stochastic programming solution relative to the simpler decision rule $x^\star(\bar{\xi})$:

$$VSS = EEV - SP$$

In terms of computational effort, it has already been mentioned that SP is more challenging to compute than WS. On the other hand, EEV is less challenging to compute than WS: in order to compute EEV it is necessary to solve, for every $\omega \in \Omega$, a linear program over second-stage variables, whereas for WS it is necessary to solve a linear program over first *and* second-stage variables.

Example 8.2 Returning to the investment problem of the previous example, the EV solution is presented in table 8.1. The EEV is derived in section 1.2 and is equal to 12739 \$/h. The benefit of the stochastic programming solution stems from variable costs. The total investment cost and variable cost of the two solutions is presented in tables 1.5 and 1.6. The total investment cost of the EV solution is in fact lower than that of the SP solution. The main difference between the two policies is driven by the fact that the EV investment is not adequate for serving the peak demand that could occur in the reference scenario with a low probability, and therefore resorts to costly load shedding. Note that VSS is more than double the value of $EVPI$.

8.3 Comparing Performance

Under certain assumptions, the relative performance of SP, WS and EV can be determined a priori, without knowledge of the specific data for a given problem. The following inequality is general and intuitive:

PROPOSITION 8.1 *Wait-and-see outperforms the stochastic programming solution:* $WS \leq SP$.

Proof For every ω, $z(x^\star(\xi), \xi) \leq z(x^\star, \xi)$, where $x^\star$ is the optimal solution to the recourse problem. Taking expectations on both sides yields $WS \leq SP$. □

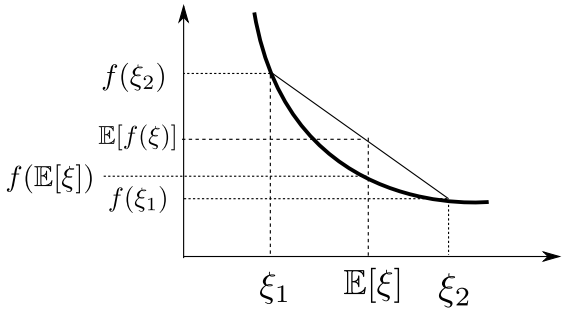


Figure 8.1 A graphical illustration of Jensen's inequality.

The interpretation of this result is that better performance can be attained if uncertainty is known in advance. The next result establishes that it is valuable to account for uncertainty explicitly, when optimizing decisions.

PROPOSITION 8.2 *The stochastic programming solution outperforms the EV solution: $SP \leq EEV$.*

Proof This proof exploits the fact that EEV and SP are both expressed as functions of $z(x, \xi)$. The vector $x^\star$ is the optimal solution of

$$\min_x \mathbb{E}[z(x, \xi)]\}.$$

The vector $x^\star(\bar{\xi})$ is a solution for this same minimization problem, therefore $\min_x \mathbb{E}[z(x, \xi)] = SP \leq EEV = \mathbb{E}[z(x^\star(\bar{\xi}), \xi)]$. □

The following result is useful for deriving a relationship between EEV and SP.

PROPOSITION 8.3 (Jensen's Inequality) *Suppose f is convex and ξ is a random variable, then $f(\mathbb{E}[\xi]) \leq \mathbb{E}[f(\xi)]$.*

Jensen's inequality is demonstrated graphically in figure 8.1. The proof of the inequality is left as an exercise.

PROPOSITION 8.4 *Suppose c, q, W, T are not random (i.e., $\xi = h$): then $z(x, h)$ is jointly convex in (x, h).*

Proof Consider x_1, x_2 and $\lambda \in (0, 1)$. Without loss of generality, assume that $Ax_1 = b$, $Ax_2 = b$, $x_1, x_2 \geq 0$ (since otherwise the convexity inequality holds trivially with both sides being equal to infinity). Denote $z(x_i, h_i) = c^T x_i + q^T y_i$, where $y_i = \min\{q^T y | Wy = h_i - Tx_i, y \geq 0\}, i = \{1, 2\}$. By definition of z,

$$z(\lambda x_1 + (1-\lambda)x_2, \lambda h_1 + (1-\lambda)h_2) = c^T(\lambda x_1 + (1-\lambda)x_2) + q^T y_\lambda,$$

where

$$y_\lambda = \min\{q^T y | Wy = \lambda h_1 + (1-\lambda)h_2 - T(\lambda x_1 + (1-\lambda)x_2), y \geq 0\}$$

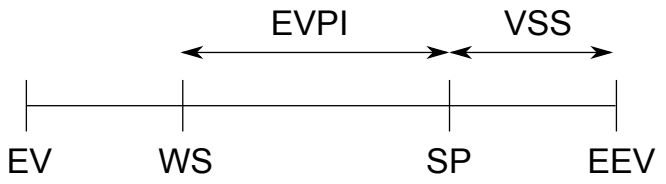


Figure 8.2 A summary of the relationships between EV, WS, SP and EEV when $\xi = h$.

Since $\lambda y_1 + (1 - \lambda)y_2$ is a feasible solution for the optimization problem $\min\{q^T y | Wy = \lambda h_1 + (1 - \lambda)h_2 - T(\lambda x_1 + (1 - \lambda)x_2), y \geq 0\}$, it follows that $q^T y_\lambda \leq \lambda q^T y_1 + (1 - \lambda)q^T y_2$. This implies that

$$z(\lambda x_1 + (1 - \lambda)x_2, \lambda h_1 + (1 - \lambda)h_2) \leq \lambda z(x_1, h_1) + (1 - \lambda)z(x_2, h_2),$$

which establishes the result.

□

PROPOSITION 8.5 *Suppose c, q, W, T are not random (i.e. , $\xi = h$): then $EV \leq WS$.*

Proof From proposition 2.23 it follows that $f(h) = \min_x z(x, h)$ is convex in h. Applying Jensen's inequality, one concludes that $\mathbb{E}[f(h)] \geq f(\mathbb{E}[h])$. The result follows. □

This result has an interesting interpretation: EV is a biased estimate of the true expected cost. The bias is, in fact, quite strong because WS is already an optimistic estimate of expected cost, which cannot be attained unless the decision maker has access to a perfect forecast of uncertainty.

Example 8.3 Returning to the capacity expansion planning problem of the previous examples, the problem satisfies the assumptions of proposition 8.5. It is therefore the case that $EV \leq WS$. WS is derived in example 8.1, with $WS = 334674$ \$/h. Solving the expected value problem yields an objective function value of $EV = 334674$ \$/h. It is left as an exercise to show that this is not a coincidence, i.e. $EV = WS$ holds in general for the two-stage stochastic capacity expansion problem with demand uncertainty.

The results that are derived in this section are summarized in figure 8.2.

8.4 Estimating Performance

As mentioned in section 8.1, the computation of EV amounts to solving a single linear program, which is generally tractable. The computation of SP has been the subject of the previous chapters. Nothing has been said about the computation, or estimation, of EEV and WS.

It is important to note that generalizing the definition of EEV in a multi-stage setting is not obvious, since it is not clear which decisions should be fixed. By contrast, it is fairly obvious how to generalize the definition of WS in a multi-stage setting: one optimizes against every possible realization $(\xi_1, \ldots, \xi_H)$, and computes WS as the weighted average.

The exact computation of EEV and WS can only be expected whenever there is a limited number of uncertain parameters. Even in a two-stage stochastic programming context, n random parameters that are discretized at d values require the resolution of d^n linear programs, which is intractable for large values of n. It is therefore necessary to resort to simulation in order to estimate WS and EEV.

The estimation of WS and EEV through sample mean approximation proceeds as follows:

- For $i = 1, \ldots, K$
  - Sample $\xi_i = \xi(\omega_i)$
  - Compute $x^\star(\bar{\xi})$
  - Compute $WS_i = z(x^\star(\xi_i), \xi_i)$ and $EEV_i = c^T x^\star(\bar{\xi}) + Q(x^\star(\bar{\xi}), \xi_i)$
- Estimate $\bar{WS} = \frac{1}{K}\sum_{i=1}^{K} WS_i$ and $\bar{EEV} = \frac{1}{K}\sum_{i=1}^{K} EEV_i$

The estimates $\bar{WS}$ and $\bar{EEV}$ converge to the true value of WS and EEV as K grows. Convergence behavior is dictated by the central limit theorem (theorem 7.2). Although this method of estimation is often adequate, it suffers from an important weakness: rare outcomes that strongly influence the estimate require an excessively large number of samples in order to be observed. This weakness is demonstrated in the following example.

Example 8.4 Suppose that the cost C of operating a facility is
- $C(N) = 1$ under normal operations, with $\mathbb{P}[N] = 0.9$, and
- $C(E) = 100$ under emergency operations, with $\mathbb{P}[E] = 0.1$,

where $\Omega = \{N, E\}$ is the set of outcomes. Then

$$\mu = \mathbb{E}[C] = 0.1 \cdot 100 + 0.9 \cdot 1 = 10.9$$
$$\sigma = \sqrt{var(C)} = \sqrt{0.9 \cdot (1 - 10.9)^2 + 0.1 \cdot (100 - 10.9)^2} = 29.7$$

Note that a rare outcome (that occurs in 1 out of 10 samples) influences expected value calculation since it contributes by $\frac{0.1 \cdot 100}{10.9} = 91.7\%$ to the expected value that is being estimated. From the central limit theorem, it can be concluded

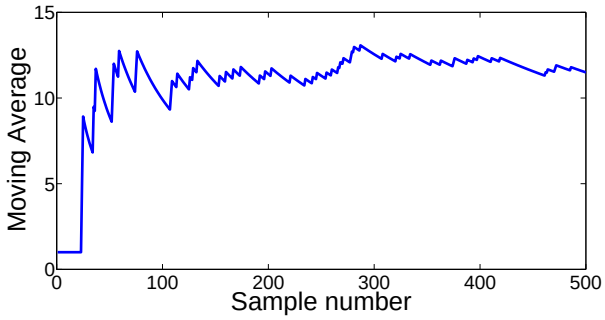


Figure 8.3 A sample of the evolution of the moving average $\frac{1}{n}\sum_{i=1}^{n} C(\omega_i)$ where ω_i denotes the outcome of sample i.

that in order to obtain an estimate of $\mathbb{E}[C]$ with 5% accuracy with a confidence of 95.4% , it is necessary to conduct $2\frac{\sigma}{\sqrt{n}} = 0.05\mu$, from which it is concluded that $n = 11879$ samples are required. A sample of the evolution of the moving average is presented in figure 8.3. Note the sensitivity of the estimate on the rare emergency outcome, even after a few hundred iterations.

Importance sampling is an alternative method of simulation that remedies the high variance of the sample mean estimate. The idea of importance sampling is to place greater weight on rare outcomes when sampling (so that these rare outcomes are observed more frequently in a simulation), and adjust the estimation of the population mean in order to cancel the effect of this biased selection of samples. In order to clarify the idea, suppose that the goal is to estimate $\mathbb{E}[C(\omega)]$, where ω is distributed according to $f(\omega)$. Sample mean estimation obtains samples ω_i according to the distribution $f(\omega)$ and estimates $\mathbb{E}[C(\omega)]$ using the sample mean, $\sum_{i=1}^{K} \frac{1}{K} C(\omega_i)$. By contrast, importance sampling uses a different distribution $g(\omega)$ for sampling, which places greater weight on outcomes that have a greater impact on the expected value. More concretely, in importance sampling the samples ω_i are obtained according to distribution $g(\omega)$ and $\mathbb{E}[\xi]$ is estimated as follows:

$$\bar{C} = \sum_{i=1}^{N} \frac{1}{N} \frac{f(\omega_i) \cdot C(\omega_i)}{g(\omega_i)}$$

The random variable $\frac{C(\omega) \cdot f(\omega)}{g(\omega)}$, which is distributed according to $g(\omega)$, also has expectation $\mathbb{E}[C]$. Ideally, the measure g that is used in importance sampling places a probability on ω which is equal to the contribution of outcome ω_i on the expected cost:

PROPOSITION 8.6 *If* $g(\omega) = \frac{f(\omega) \cdot C(\omega)}{\mathbb{E}[C]}$ *then importance sampling yields* $\bar{C} = \mathbb{E}[C]$.

Proof Note that $\mathbb{E}[C(\omega)] = \int_\Omega C(\omega) \cdot df(\omega) = \int_\Omega \frac{C(\omega) \cdot f(\omega)}{g(\omega)} dg(\omega) = \bar{C}$ □

The conclusion of the previous proposition is that *any* sample evaluates to $\mathbb{E}[C]$! Of course, the use of $g(\omega) = \frac{f(\omega) \cdot C(\omega)}{\mathbb{E}[C]}$ is not possible because computing $g(\omega)$ requires knowledge of $\mathbb{E}[C]$, which is what is being estimated in the first place. Nevertheless, importance sampling provides useful intuition: sampling the uncertain parameters according to their contribution to expected value can accelerate the estimation of $\mathbb{E}[C]$. Even if $\mathbb{E}[C]$ in proposition 8.6 is not known exactly, it could be *approximated* relatively accurately in certain cases.

Example 8.5 Returning to example 8.4, the drawback of sample mean estimation is that rare emergency outcomes have the greatest influence on expected value. The underlying distribution can be redefined so that emergency outcomes are observed earlier. In particular, if $\mathbb{E}[C]$ is known at the outset, the following distribution $g(\omega)$ could be obtained:

$$g(\omega_1) = \frac{f(\omega_1) \cdot C(\omega_1)}{\mathbb{E}[C]} = \frac{0.9 \cdot 1}{10.9} = \frac{0.9}{10.9}$$
$$g(\omega_2) = \frac{f(\omega_2) \cdot C(\omega_2)}{\mathbb{E}[C]} = \frac{0.1 \cdot 100}{10.9} = \frac{10}{10.9}$$

Given this underlying distribution, importance sampling estimates for either outcome ω_1 or ω_2 are constant and equal to $\mathbb{E}[C]$:

$$C(\omega_1) \cdot \frac{f(\omega_1)}{g(\omega_1)} = 1 \cdot \frac{0.9}{\frac{0.9}{10.9}} = 10.9$$
$$C(\omega_2) \cdot \frac{f(\omega_2)}{g(\omega_2)} = 100 \cdot \frac{0.1}{\frac{10}{10.9}} = 10.9$$

Problems

8.1 What can be said about VSS if $x^\star(\xi)$ is independent of ξ?
8.2 Prove Jensen's inequality.
8.3 Provide an example where $EV > WS$.
8.4 Prove that for the two-stage capacity expansion planning problem (see the formulation in example 4.2) the following equality holds: $WS = EV$.

Bibliography

Chapter 8.1. The exposition of this material follows (Birge & Louveaux 2010).
Chapter 8.2. The exposition of this material follows (Birge & Louveaux 2010).
Chapter 8.3. The exposition of this material follows (Birge & Louveaux 2010).

References

Bertsekas, D. P. (1995), *Dynamic Programming and Optimal Control*, Athena Scientific.

Bertsimas, D. & Tsitsiklis, J. N., eds (1997), *Introduction to Linear Optimization*, Athena Scientific.

Birge, J. R. & Louveaux, F. (2010), *Introduction to Stochastic Programming*, Springer Series in Operations Research and Financial Engineering, Springer.

Boyd, S. P. & Vandenberghe, L. (2008), *Convex Optimization*, Cambridge University Press.

Cabral, F. G. (2016), A proposal for a multiplicative autoregressive multivariate periodic model for the generation of inflow scenarios applicable to the planning model of the Brazilian electricity sector, PhD thesis, Federal University of Rio de Janeiro.

Dantzig, G. B. & Glynn, P. W. (1990), 'Parallel processors for planning under uncertainty', *Annals of Operations Research* **22**, 1–21.

Infanger, G. & Morton, D. P. (1996), 'Cut sharing for multistage stochastic linear programs with interstage dependency', *Mathematical Programming* **75**, 241–256.

Louveaux, F. V. & Smeers, Y. (1988), *Numerical Techniques for Stochastic Optimization*, Springer-Verlag, chapter Stochastic Optimization for the Introduction of a New Energy Technology, pp. 33–64.

Luenberger, D. (1998), *Investment Science*, Oxford University Press.

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

Puterman, M. L. (2014), *Markov decision processes: discrete stochastic dynamic programming*, John Wiley and Sons.

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

Author index

Subject index