PURE AND APPLIED MATHEMATICS

A SERIES OF MONOGRAPHS AND TEXTBOOKS

Dynamic Programming

Foundations and Principles

Second Edition

$$f_n(s) = \underset{x \in D(n,s)}{\text{opt}} \{w_n(s,x) \oplus f_{n+1}(T(n,s,x))\}$$

Moshe Sniedovich

CRC Press
Taylor & Francis Group
A CHAPMAN & HALL BOOK

Dynamic Programming

Foundations and Principles

Second Edition

Dynamic Programming

Foundations and Principles

Second Edition

Moshe Sniedovich
University of Melbourne
Melbourne, Australia

CRC Press
Taylor & Francis Group
Boca Raton London New York

CRC Press is an imprint of the
Taylor & Francis Group, an **informa** business
A CHAPMAN & HALL BOOK

CRC Press
Taylor & Francis Group
6000 Broken Sound Parkway NW, Suite 300
Boca Raton, FL 33487-2742

Printed in the United States of America on acid-free paper
10 9 8 7 6 5 4 3 2 1

International Standard Book Number-13: 978-1-4200-1463-1 (Ebook-PDF)

Preface *(first edition)*

I started entertaining the idea of writing a book on dynamic programming when I first realized that in contrast to, say, linear programming, there seems to be a gentlemen's disagreement as to what exactly dynamic programming is. Obviously, I envisioned a text that would offer a conclusive and incontestable formulation of dynamic programming. But I soon discovered that dynamic programming actually invites a certain amount of disagreement with respect to its definition because, by its very nature, it can be approached from a variety of angles depending on one's approach to modeling, analysis and problem solving. We thus find the intuitionist at one end, the formalist at the other, and the rest somewhere in between. So, in this book I examine the question *what is dynamic programming*? knowing full well that whatever answer one formulates, by necessity it will be subjective in nature.

This book does not aim to portray dynamic programming as a practical tool. It does not examine how dynamic programming solves real-world problems in inventory, scheduling, sport and recreation, expert systems, etc. The book aims to give a portrait of dynamic programming as a *methodology* — to identify its constituent components, explain how it looks at problems, and describe how it proposes to tackle them. It should therefore appeal to readers in academia as well as to practitioners who have an interest in this facet of dynamic programming.

It should be of particular interest to readers who *teach* dynamic programming, as in my view, this book offers a refreshing supplement to texts which present dynamic programming from the conventional standpoint of *Learn/Teach Dynamic Programming by Example.* These readers are advised that, although the book is not designed to serve as a course-text, because it is self-contained it can be used as ancillary reading for graduate and advanced undergraduate courses in dynamic programming.

As for my method of probing the topic; although I fully recognize that certain aspects of dynamic programming are probably best suited for a non-formal treatment, my exposition throughout this book is strictly formal. Still, the mathematical knowledge required for following the presentation is minimal — calculus, set theory and some optimization theory, all at the elementary level. The book does demand, however, mathematical maturity which comes into play more in modelling and formulation than in analysis.

I would like to thank my colleagues and graduate students for their helpful comments and constructive criticism on various drafts of the manuscript. I am particularly grateful to Alleli Domingo, Emmanuel Macalalag and Steven

Ostrover for their comments on the final draft. I thank Ruth Dawe and Henry Boehm from Marcel Dekker Inc. for their encouragement, patience and support. And finally, special thanks are due to my wife, Shoshana, for her major contribution to this book.

Substantial work on this book was done when I was with the National Research Institute for Mathematical Sciences of the CSIR, Pretoria, South Africa.

Moshe Sniedovich
Melbourne
June 1991

Preface *(second edition)*

In the intervening years since the publication of the first edition of this book in 1992, I continued working on various aspects of dynamic programming and the results of this research appeared in a number of publications. So, when I was approached by the publisher with the suggestion to consider a second edition of this book, I decided to take up the offer realizing that a second edition should provide me with an opportunity to enhance certain chapters in the first edition with the material (ideas, examples, etc.) that I had developed in more recent years. I was particularly keen to revisit a topic that I consider crucial for a good understanding of dynamic programming, namely *Modeling.*

And so, I expanded the discussions on *sequential decision models* and the *role of the state variable in modeling*, and I added a new chapter on *forward* dynamic programming models.

I also added a new chapter where I discuss the *Push* method — also known as *Reaching* — and I give a dynamic programming perspective on Dijkstra's algorithm for the shortest path problem. My main objective in including this discussion on Dijkstra's algorithm is to make it abundantly clear that — in spite of the impression given by the computer science literature — this algorithm is in fact a dynamic programming algorithm *par excellence.*

Thus, apart from slight modifications in the contents of existing chapters, the major revisions are as follows:

- Significantly extended chapters:
  - Chapter 10: Refinements
  - Chapter 11: The State
- New chapters:
  - Chapter 14: Forward Decomposition
  - Chapter 15: Push!
- New appendix:
  - Appendix E: The Corridor Method

As for the book's new title.

With hindsight, it seems to me that this should have been the book's title in its first edition! The new title, I submit, captures more faithfully the goal that I had set out to accomplish in writing this book, which as I indicated in the preface to the first edition, was to give — dare I say — a more formal treatment to Bellman's presentation of dynamic programming.

The new title thus reflects the fact that while the discussion in this book takes account of recent developments in dynamic programming, its overriding objective is a systematic formal outline of Bellman's approach to dynamic programming.

As a final note, I wish to point out that since the book's publication, I had many occasions to discuss various questions that I raise in the book with colleagues and students. Obviously, their comments on the first edition of the book have been of great benefit to me as they enabled me to formulate more crisply my positions on certain questions that I discuss in the new edition. To provide, however, a full list of all those I am indebted to would prove impossible. I therefore single out, for special thanks, Khurram Kamran for his most valuable comments on several drafts of the second edition.

I thank the Taylor & Francis team: David Grubbs, Bob Stern and Pat Roberson — for their support in this project.

Moshe Sniedovich
Melbourne
July 2010

List of Figures

List of Tables

Contents

1

Introduction

1.1 Welcome to Dynamic Programming!

Almost sixty years have passed since the publication of Bellman's first articles introducing the theory that he entitled *Dynamic Programming.* Since then dynamic programming has become a major discipline in applied mathematics, operations research and computer science, and a standard solution method used in various areas of engineering, economics, commerce, management etc. And yet, the question that I have set out to examine in this book is none other than *What is dynamic programming?* Obviously, an explanation justifying the validity of this position is in order.

My reasons for making this question the fulcrum of this investigation can be summed up as follows. First, I believe that the continued appearance of publications that, even at this stage in the development of dynamic programming, offer new settings, reformulations and even generalizations of the theory, attests to a continued interest in the foundational questions of dynamic programming. I assumes therefore that a study aiming to explain dynamic programming from the standpoint of the most fundamental question that can be asked about it should be deemed a worthwhile effort, at least by dynamic programming scholars whose interest in the theory runs deeper than its application.

Second, having long been involved in the study of dynamic programming, and having followed closely the literature published in this area, I submit that not all is well in dynamic programming. To be sure, almost sixty years of extensive research and application have produced truly excellent books and articles on the various aspects of dynamic programming.

Still, alongside works of the first rank, one often comes across publications that attest to a basic lack of understanding about the type of problems that dynamic programming is most suited for, the precise nature of dynamic programming's plan of attack, and the range of dynamic programming's capabilities.

This somewhat unsettled state of affairs with regard to the fundamentals of dynamic programming is also manifested in the lively debates in dynamic programming sessions at conferences and workshops where one often hears totally conflicting views on these matters.

Now, readers who are not at home in dynamic programming may wonder what the fuss is all about. Why should dynamic programming give rise to argument with regard to its very fundamentals when no such argument arises in the case of other optimization methods notably *Linear Programming?*

Indeed, no controversy arises as to *Linear Programming's* mode of operation, its capabilities, or its scope. In a word, no ambiguity surrounds the question *"What is Linear Programming?"* It is commonly accepted that *Linear Programming* is a branch of optimization theory that concerns itself with the modeling, analysis, and solution of linear programming problems.

The reason that this characterization of *Linear Programming* proves satisfactory is that it is universally accepted that a *Linear Programming Problem* is an optimization problem whose objective function and constraints are linear. What is more, an almost equally universally accepted formulation for such problems is also available, namely:

$$\operatorname*{opt}_{x} \; cx \tag{1.1}$$

$$s.t. \quad Ax = b \tag{1.2}$$

$$x \geq 0 \tag{1.3}$$

where $c \in \mathbb{R}^{1\times n}$, $b \in \mathbb{R}^{m\times 1}$, $A \in \mathbb{R}^{m\times n}$, $x \in \mathbb{R}^{n\times 1}$ and $\mathbb{R}$ denotes the real line.

So, why can't we simply draw an analogy with *Linear Programming*, and proceed to define dynamic programming a branch of optimization theory that concerns itself with the modeling, analysis, and solution of *dynamic programming problems?*

The reason that such a characterization of dynamic programming would not be as straightforward as in the case of *Linear Programming* is that in dynamic programming we lack consensus on what constitutes a *dynamic programming problem.*

So, taking this state of affairs as my point of departure, I shall proceed in the opposite direction. That is, I shall begin from the position that a *dynamic programming problem* is any problem that is amenable to the type of treatment that dynamic programming prescribes. Hence, an elucidation of the mode of operation by means of which this treatment is applied should have the effect of elucidating the nature of dynamic programming, namely the question *What is dynamic programming?*

My plan is then to progressively unfold the mechanics of this mode of operation.

As a first step, I shall begin with a rather abstract description of this mode of operation, a sort of *meta-recipe* that I shall argue underlies and drives dynamic programming's plan of attack:

1. *Embed* your problem in a family of related problems.
2. *Derive* a relationship between the solutions to these problems.
3. *Solve* this relationship.
4. *Recover* a solution to your problem from this relationship.

As things stand then, my position is that *any* problem admitting of the kind of treatment outlined by this *meta-recipe* is a *dynamic programming problem.*

It is important to note that no appeal whatsoever is made in this *meta recipe* to *optimization.* This position therefore has the effect of characterizing dynamic programming as a general problem-solving methodology rather than just an optimization method. Furthermore, the implications of this characterization of dynamic programming are that:

· The class of dynamic programming problems is extremely large.
· Members of the public routinely use dynamic programming without even realizing it.

I hasten to add, though, that my focus in this book will be on dynamic programming in its familiar role as an *optimization method.*

To be able to bring out how this *meta recipe* governs dynamic programming's treatment of a problem, I shall have to clarify the following questions:

1. What types of *mathematical models* are suitable for dynamic programming?
2. What problems are amenable to *full treatment* by dynamic programming?
3. What is the function of the *state variable* in dynamic programming?
4. What is the exact role of the *Principle of Optimality* in dynamic programming?

I wish to point out, though, that by targeting these topics for discussion I do not mean to suggest that they are still open to debate, nor that my treatment of them is a revelation. Obviously, these and related topics were dealt with by Bellman in his numerous books and articles on dynamic programming; and they were either touched on or discussed at length by other dynamic programming scholars.

The point is, however, that discussions on these issues are scattered over a wide ranging literature that cuts across a variety of disciplines differing in mathematical idiom, fundamental objectives, and basic positions. So, my aim is to approach these topics in a systematic fashion, from a single, coherent, point of view about dynamic programming.

The perspective from which I shall examine these topics will be what I take to be is Bellman's understanding of dynamic programming. I believe that this approach will make it possible to clear away misconceptions not only about dynamic programming, but also about Bellman's treatment of it. A case in point is the *Principle of Optimality.* I argue here that the *Principle* has often been misconstrued in the literature and that this accounts for much of the misapprehension about dynamic programming and Bellman's conception thereof.

So, to a certain extent, my treatment of these issues and hence of dynamic programming as a whole, comes close to treatments of the method that one would be likely to encounter elsewhere in the literature. However, in this important respect that, as indicated, my fundamental approach to dynamic programming faithfully reflects Bellman's conception, it sharply diverges from prevailing approaches to, and explanations of, dynamic programming. The irony of this is that, when it is set off against "standard" treatments of dynamic programming, my interpretation presents dynamic programming in a new light.

Turning now to the considerations that influenced and ultimately determined my method and style of exposition.

First, the model that I have chosen to employ as a medium of discourse. In principle I could have used any one of the models that dynamic programming provides. However, because my overriding objective in this investigation is to expound the basics of dynamic programming and its foundational issues, the ideal medium for such a discussion is a *simple* model that does not require a constant accounting for and explanation of technical considerations. Such a framework, I submit, is provided by a *deterministic multistage decision model.*

So except for touching in *Appendix D* on the interpretation of the *Principle of Optimality* in the *stochastic* case, I shall not extend the discussion to stochastic decision processes. This in spite of dynamic programming's well-earned reputation of being particularly effective for stochastic problems.

It is my view that a detailed analysis of the deterministic model promises to provide a complete account of dynamic programming's *essentials* with the added bonus of avoiding digressions into areas such as probability theory and statistics that are not central to the discussion. It should be noted, however, that the treatment of the deterministic model outlined in this book can be readily extended to stochastic models. To show how smoothly this can be done I shall take up a few specific stochastic problems to demonstrate how they are handled by dynamic programming.

For similar reasons I do not go into *multiobjective* or *fuzzy* processes and I devote no more than a short discussion in *Chapters 11 and 14* to the no doubt important class of *non-serial* dynamic programming models.

Secondly, the style of exposition that I decided to adopt in this book. In this case as well, I could have opted for a number of styles. For instance, I could have followed Bellman's example and chosen to explain dynamic programming by means of a narrative exposition. However, being mindful that such a style may be construed as lacking in rigor, I decided to adopt a formal mathematical style. I believe, however, that I manage to maintain a proper balance by not engaging in technical analyses where these are not essential. For example, for the most part I shall not go into the question of the existence of optimal solutions, but rather shall assume this to be the case.

As indicated in the preface, this book is intended primarily for readers with some background in dynamic programming. And yet, my exposition works its way from what are no doubt first principles, that is, material that may well be described as elementary or self-evident. My major consideration in adopting this line is not so much to render the book self-contained and accessible also to beginners, as to make a point about dynamic programming itself. My objective is to emphasize that dynamic programming consists of a core of profoundly simple concepts and techniques. However, a good grasp of it requires that one be able to *identify* these rudimentary elements and appreciate how they hang together.

As for the the general structure of this book. I divided it into three parts. The discussion in the first part, which will be referred to broadly as *Science,* will focus on dynamic programming's mathematical idiom and techniques. The goals here are to describe dynamic programming's terminology and notation, explain how it expresses a problem in terms of a mathematical model and how it then boils this model down to a functional equation, outline and analyze the techniques used to solve the equation and discuss the difficulties encountered in its solution. A major concern in this part of the inquiry will also be to identify the type of problems that are amenable to dynamic programming.

The investigation in the second part will be referred to broadly as *Art.* Here the predominant aim will be to concentrate on those aspects that have gained dynamic programming its reputation, or perhaps notoriety, of being an "approach" more than a "method", "art" more than "science". Namely, the accent will be on *modeling* and *formulation.* The object will be to indicate an approach to modeling and formulation by clarifying the role of the *state variable* in this effort. Modeling and formulation will be further elaborated upon in the chapter entitled *Parametric Schemes,* where the idea is to illustrate ways of tackling difficult problems by capitalizing on their structural features and making effective use of them. Appropriately, this part will include a discussion on the *Principle of Optimality* and Bellman's conception of dynamic programming.

The last two chapters in this part of the book will take up two important topics that in my judgment do not receive the attention they merit. These are: *Forward Decomposition* and the *Push* method for solving dynamic programming functional equations (also known as *Reaching*).

In the third part, the main findings of the first two parts will be pulled together to provide what I believe will prove a conclusive answer to the central question pursued in this book, which, as I stated, is: *What is dynamic programming?*

1.2 How to Read This Book

I suggest that readers who are new to dynamic programming, or have only a fleeting knowledge of it, would do best to follow the evolution of my argument in the order in which it is unfolded, which means of course reading the book in sequence. Those with no particular interest in the more technical aspects of the successive approximation method can make do with the general outline of the method delineated in *Chapter 5,* thus skipping *Chapter 6* altogether. For similar reasons one can also skip those sections in *Chapter 7* that deal with non-truncated problems. As for *Chapter 12,* given its reliance on a close familiarity with fractional programming and c-programming, readers who are not familiar with these methods would perhaps do best to skip it on first reading, returning to it at a later stage after they had read *Appendix B* and *Appendix C.*

Readers who are conversant with dynamic programming can of course read the book in any order they see fit. Still, I submit that reading it in sequence is conducive to a better appreciation of my approach to dynamic programming.

Part I

Science

2

Fundamentals

2.1 Introduction

The conceptual dimension of dynamic programming consists of a cluster of ideas whose profound simplicity presents the dynamic programming scholar setting out to elucidate them with a difficult task. This is so because to be able to do them justice, one would need to explicate them in terms of the techniques that make them work, which would entail interlacing a detailed analysis of the techniques themselves with the discussion. But such a strategy stands to accomplish the precise opposite of what is sought. Combining an exposition of the techniques with the discussion of the ideas is more likely to bury the latter in a mass of technical detail than to bring out their full meaning.

To avoid this pitfall, I shall begin my inquiry into the question *What is dynamic programming?* by first turning my attention to the techniques that dynamic programming deploys.

This will not only pave the way for an unobstructed examination of the main ideas in later chapters, but it will enable working out the full sense of the techniques. Furthermore, it will equip us with a body of terminology and notation on which we shall be able to draw throughout the remainder of the book.

Now, ever since the inception of dynamic programming, it has been accepted practice to conduct the presentation of the method in the context of a *sequential* or *multistage decision-making* model, so as to avoid becoming embroiled in a discussion of peripheral technical matters.

Obviously, I shall adhere to this convention as well and in the next chapter I shall outline the *multistage decision model* that from then onwards will serve as our medium of discourse.

In this chapter, however, I shall depart from this practice and conduct the discussion in the framework of a far less hospitable setting than that of the *multistage decision model*. That is, in this chapter our framework of discussion will be devoid of all reference to a formal sequential decision process. Or, in other words, it will be devoid of all the elements that are commonly associated with dynamic programming. The advantage of adopting this setting as a medium of discourse is twofold.

First, it will enable me to bring into sharp focus Bellman's ingenious idea to cast an optimization problem in terms of a sequential decision process and to formulate dynamic programming along these lines. Of course today, that this idea has taken root, it is being treated as the obvious way to go. But, unless one is aware of its background, it is difficult to truly appreciate its great merit.

Second, by using a so to speak "dynamic-programming-free-zone", as a framework of presentation, I shall be able to uncover the rudimentary concepts and techniques out of which dynamic programming evolves. And what is more, I shall be able to demonstrate this "evolution process" in action.

In short, my main thrust in this chapter will be to identify the basic concepts and techniques that, as we shall progressively see, constitute the fundamental building blocks of the dynamic programming "infrastructure".

In the next chapter these basic ingredients will be used in the construction of a formal *multistage decision model* that will provide the framework of discourse in all subsequent chapters. That said, a word of caution is in order.

Some readers may find the discussion in this chapter somewhat annoying because of its laborious analysis of techniques and procedures that in the end are found lacking and hence in need of considerable improvement.

These readers, especially if their interest in these technical analyses is marginal, may wish to skip this chapter altogether and proceed directly to the more hospitable environment of *Chapter 3.*

Still, I urge all readers to persevere, as the material in this chapter furnishes the basis for a good appreciation of the issues discussed in the sequel.

And before turning to the investigation itself, I remind the reader that although one of the objectives of this book is to show that dynamic programming is more than just an *optimization method*, for the most part, my discussion will examine it in this familiar role.

2.2 Meta-Recipe Revisited

You will recall that our point of departure is that underlying and driving dynamic programming's mode of operation is the following Meta-recipe:

1. *Embed* your problem in a family of related problems.
2. *Derive* a relationship between the solutions to these problems.
3. *Solve* this relationship.
4. *Recover* a solution to your problem from this relationship.

To be able to demonstrate this meta-recipe in action, my first task is to identify a *concrete* problem formulation that will be representative of the kind of problems that admit of this treatment.

2.3 Problem Formulation

Keeping in mind then that the main thrust of this investigation is to examine dynamic programming in its familiar role as an *optimization method,* the problem definition that will suit our purposes best will have to give apt expression to the various types of optimization problems that will concern us from now on. And, without assuming an unduly complex format, it will have to be broad enough to encompass the largest variety of problems possible. Suppose then that we consider using the following formulation to this end:

$$\textit{Problem } P^\circ\text{:} \quad p^\star := \operatorname*{opt}_{x \in X} q(x) \tag{2.1}$$

where q is a real-valued function on some set X and $:=$ denotes *definition.* We shall refer to q as the *objective function,* to X as the *solution set,* or *decision space,* and to x as the *decision variable*. Note that throughout the book "optimization" means *global optimization.*

It is immediately clear that its constituent elements — the solution set X and the objective function q — being undefined, the problem is too abstract for any meaningful analysis or treatment. The implication therefore is that to give it structure we need to ascribe properties to X and q. And furthermore, given our subject of interest, these properties must be defined so as to render *Problem P° a dynamic programming problem.* But this implies in turn that our quest for properties for X and q would have to be guided by a preexisting conception of what a dynamic programming problem is.

So, although it may seem that I am anticipating things, I shall pause here briefly in order to explain in very broad terms my position on this issue.

The question *What is a dynamic programming problem?* will be addressed at various stages in the discussion in an ongoing attempt to illuminate it from various angles. Roughly, the thesis that will be advanced in this book is that a dynamic programming problem is *any* optimization problem that lends itself to a formulation that yields an equation known as the *dynamic programming functional equation*, also the *dynamic programming optimality equation* and *Bellman's equation.*

As indicated already, this conception is an outgrowth of my view of dynamic programming's mode of operation. Dynamic programming, as we shall progressively see, can be best characterized as a *two-tier method.* By this I mean that any optimization problem proving amenable to a dynamic programming analysis will undergo a treatment consisting of two distinct steps. The first will involve a recasting of the problem into a newly formulated problem which eventually will be expressed in terms of the *dynamic programming optimality equation* mentioned above. The second will involve the solution of this equation by *whatever means are at hand.*

It follows therefore that dynamic programming *per se* does not necessarily concern itself with the techniques used to solve the optimality equation. The task of solving this equation can be taken up by *any* optimization method possessing the appropriate means to tackle the equation. The following example illustrates this important point.

2.3.1 Example

Consider the following nonlinear optimization problem:

$$p^* := \max_{(x_1,x_2,\dots)} \sum_{n=1}^{\infty} q_n(x_n) \tag{2.2}$$

$$\sum_{n=1}^{\infty} x_n \leq r \ , \ r > 0 \tag{2.3}$$

$$x_n \geq 0 \ , \ n = 1,2,3,\dots \tag{2.4}$$

This problem will be studied at length in the ensuing chapters. The sole purpose of bringing it up at this early stage is to point out the following.

To obtain a dynamic programming formulation for this problem we simply define

$$f_n(s) := \max_{(x_n,x_{n+1},\dots)} \sum_{k=n}^{\infty} q_k(x_k) \ , \ n \geq 1, s \in S := [0,r] \tag{2.5}$$

$$\sum_{k=n}^{\infty} x_k \leq s \ , \ k = n, n+1, n+2, \dots \tag{2.6}$$

$$x_k \geq 0 \ , \ k = n, n+1, n+2, \dots \tag{2.7}$$

whereupon, using standard dynamic programming arguments, the following *dynamic programming optimality equation* would be derived:

$$f_n(s) = \max_{0 \leq x \leq s} \{q_n(x) + f_{n+1}(s-x)\} \ , \ s \in S, n = 1,2,3,\dots \tag{2.8}$$

This is the *first tier* alluded to above which consists of translating the formulation of the problem under consideration, namely (2.2)-(2.4), into a dynamic programming functional equation, (2.8).

Solving this equation, however, is a separate matter altogether. This is so because an answer to the question whether the equation is tractable at all, *and/or* what techniques will be required to solve it, will depend entirely on the properties of the functions $\{q_n : n \geq 1\}$. $\square$

And to go back to *Problem* P°, depending on the properties of X and q in a given case, some solution procedures for the associated dynamic programming optimality equation would be purely analytic, others purely numeric, still others a composite of both, and in certain cases, needless to say, the dynamic programming optimality equation may elude solution altogether.

So, the conclusion to be drawn is that by raising the issue of what properties should be attributed to X and q to render *Problem* P° a dynamic programming problem, I am actually posing the following question:

> What properties should be ascribed to X and q to enable *Problem* P° yield a dynamic programming optimality equation?

Happily, the answer to this does not involve a protracted search, as it is rather simple and straightforward:

> *Problem* P° will yield a dynamic programming optimality equation if the solution set X is a subset of the Cartesian product of two or more sets.

To be able to explain things as simply as possible, it would be best to begin our inquiry with an examination of the case where the solution set X is a subset of the Cartesian product of just *two* sets.

Consider then the following class of optimization problems:

$$p^* := \operatorname*{opt}_{(y,z)\in X} q(y,z) \ , \ X \subset \widetilde{Y} \times \widetilde{Z} \tag{2.9}$$

The point to note here is that although this format consists of no more than two decision variables, namely y and z, it nevertheless spans a wide variety of problems. This is so because no conditions are imposed on the sets $\widetilde{Y}$ and $\widetilde{Z}$.

Continuing with *Example 2.3.1,* to comply with the above format the following can be set:

$$\widetilde{Y} = [0,r] \tag{2.10}$$

$$\widetilde{Z} = [0,r]^\infty \tag{2.11}$$

$$X = \left\{(y,z) : y \in \widetilde{Y}, z \in \widetilde{Z}, \left(y + \sum_{n=1}^{\infty} z_n\right) \leq r\right\} \tag{2.12}$$

$$q(y,z) = q_1(y) + \sum_{n=1}^{\infty} q_{n+1}(z_n) \ , \ y \in \widetilde{Y}, z \in \widetilde{Z} \tag{2.13}$$

where z_n denotes the n-th element of the vector $z \in \widetilde{Z}$.

To sum up then, I shall proceed on the thesis that any problem admitting a formulation such as that specified by (2.9) is a dynamic programming problem.

Let us now turn to the techniques that dynamic programming employs to derive an optimality equation for such problems.

2.4 Decomposition of the Solution Set

However unequivocal (2.9) is in its definition of the optimization problem in question, when this problem is viewed from a dynamic programming standpoint, it is construed in slightly different terms. It undergoes a reformulation, which is based on the following:

Theorem 2.4.1 *In relation to the problem defined by (2.9), there exist a set $Y \subseteq \widetilde{Y}$ and a collection of sets $\{Z(y) : y \in Y\}$ such that $Z(y) \subseteq \widetilde{Z}$, $\forall y \in Y$ and*

$$X = \{(y, z) : y \in Y, z \in Z(y)\} \tag{2.14}$$

Proof. Let Y denote the projection of X on $\widetilde{Y}$, namely define

$$Y := \left\{y : y \in \widetilde{Y}, (y, z) \in X,\ z \in \widetilde{Z}\right\} \tag{2.15}$$

Also, for every $y \in Y$, define

$$Z(y) := \left\{z \in \widetilde{Z} : (y, z) \in X\right\} \tag{2.16}$$

so that, by construction, (2.6) holds. □

Continuing with *Example 2.3.1,* it follows from (2.10)-(2.12) that the projection of X on $\widetilde{Y}$ is the set $Y = [0, r]$. Hence, granted (2.14)-(2.16), we can set

$$Z(y) := \left\{ z \in \widetilde{Z} : \sum_{n=1}^{\infty} z_n \leq r - y \right\}\ ,\ y \in Y \tag{2.17}$$

Theorem 2.4.1 implies then that the problem specified by (2.9) can be stated as follows:

$$\textit{Problem } P : \qquad p^{\star} := \operatorname*{opt}_{\substack{y \in Y \\ z \in Z(y)}} \ q(y, z) \tag{2.18}$$

This new format gives explicit expression to what was latent in (2.9), namely that the range of feasible values of z may depend on the value of y.

2.5 Principle of Conditional Optimization

The process of deriving the dynamic programming optimality equation for *Problem P* is governed by what I shall henceforth refer to as the *Principle of Conditional Optimization.* Its formal definition is familiar and it reads as follows:

Theorem 2.5.1 *Principle of Conditional Optimization*

$$\operatorname*{opt}_{\substack{y \in Y \\ z \in Z(y)}} \ q(y, z) = \operatorname*{opt}_{y \in Y} \left\{ \operatorname*{opt}_{z \in Z(y)} \ q(y, z) \right\} \tag{2.19}$$

PROOF. Assume that opt = max. Then, by definition,

$$\max_{\substack{y\in Y\\ z\in Z(y)}} q(y,z) \geq q(y,z) \ , \ \forall y \in Y, z \in Z(y) \tag{2.20}$$

Since for each $y' \in Y$ we have

$$\max_{\substack{y\in Y\\ z\in Z(y)}} q(y,z) \geq \max_{z\in Z(y')} q(y',z) \tag{2.21}$$

it follows that

$$\max_{\substack{y\in Y\\ z\in Z(y)}} q(y,z) \geq \max_{y\in Y} \left\{ \max_{z\in Z(y)} q(y,z) \right\} \tag{2.22}$$

On the other hand, since by definition,

$$\max_{z\in Z(y)} q(y,z) \geq q(y,z) \ , \ \forall z \in Z(y) \tag{2.23}$$

is true for every $y \in Y$, it follows that

$$\max_{y\in Y} \left\{ \max_{z\in Z(y)} q(y,z) \right\} \geq q(y,z) \ , \ \forall y \in Y, z \in Z(y) \tag{2.24}$$

and therefore we conclude that

$$\max_{y\in Y} \left\{ \max_{z\in Z(y)} q(y,z) \right\} \geq \max_{\substack{z\in Z(y)\\ y\in Y}} q(y,z) \tag{2.25}$$

Consequently, (2.22) in conjunction with (2.25) provide that the principle holds for opt = max. For the case where opt = min, replace max by min and reverse the direction of the inequalities. □

In short, the *Principle of Conditional Optimization* asserts that a problem consisting of two decision variables can be decomposed into two *embedded* problems, each consisting of a single decision variable.

2.6 Conditional Problems

The variable y on the right-hand side of (2.19) induces a family of parametric optimization problems that can be stated formally as follows:

$$\textit{Problem } P(y), y \in Y: \qquad p(y) := \operatorname*{opt}_{z\in Z(y)} \ q(y,z) \tag{2.26}$$

I shall refer to *Problem $P(y)$* as the CONDITIONAL PROBLEM AT y.

The term "conditional" is intended to point out that, in the above framework, the optimal values that the objective function q takes are conditional on the value that the variable y takes. Hence, the optimal value of z may depend on the given value of y.

2.7 Optimality Equation

Now, an application of the *Principle of Conditional Optimization* to the conditional problems yields the following obvious result:

Corollary 2.7.1

$$\textit{Optimality Equation}: \ p^\star = \operatorname*{opt}_{y\in Y} \ p(y) \tag{2.27}$$

Attention is called to the fact that this equation is not a definition of p^*. Rather, it specifies the relation between p^*, defined by (2.18), and $p(y)$, defined by (2.26). This relation is so compelling that it is sometimes mistaken for a definition.

2.8 Solution Procedure

A cursory examination of the optimality equation (2.27) suffices to conclude that the solution procedure for *Problem P*, defined by (2.18), would run as follows:

Procedure A:

1. For every $y \in Y$ solve *Problem P(y)* and determine the value of $p(y)$.
2. Determine the value of p^* by optimizing $p(y)$ over $y \in Y$. □

The following examples demonstrate this procedure in action.

2.8.1 Example

Consider the case where

$$p^* := \max_{x_1,x_2} \{\sqrt{x_1} + \sqrt{x_2}\ \} \tag{2.28}$$

$$x_1 + x_2 \leq r \ , \ r > 0 \tag{2.29}$$

$$x_1, x_2 \geq 0 \tag{2.30}$$

In view of the formulation of *Problem P°*, set

$$\text{opt} = \max \ ; \ Y = [0, r] \ ; \ Z(y) = [0, r - y] \tag{2.31}$$

$$q(y, z) = \sqrt{y} + \sqrt{z} \tag{2.32}$$

Given that in the context of *Problem* $P(y)$ the variable y is treated as a known constant, the family of conditional problems would therefore be of the following form:

$$\begin{aligned} p(y) :&= \max_{0\le z\le r-y} \{\sqrt{y} + \sqrt{z}\} \\ &= \sqrt{y} + \max_{0\le z\le r-y} \sqrt{z} \qquad (2.33)\end{aligned}$$

Also, since the optimal values of z corresponding to each value of y are given parametrically by $z^*(y) = r - y$, it follows that

$$p(y) = \sqrt{y} + \sqrt{r-y}\ ,\ 0 \le y \le r \qquad (2.34)$$

Thus, to determine the value of p^*, we can now invoke the optimality equation

$$\begin{aligned} p^* :&= \max_{0\le y\le r} p(y) && (2.35)\\ &= \max_{0\le y\le r} \left\{\sqrt{y} + \sqrt{r-y}\right\} && (2.36)\\ &= \sqrt{2r} && (2.37)\end{aligned}$$

Note that because the objective function being maximized is *concave* with y, the optimal value of y, namely $y^* = r/2$, can be obtained by equating the first derivative of the objective function to zero. The optimal solution is then $y^* = z^* = r/2$, yielding $p^* = \sqrt{2r}$. □

2.8.2 Example

Consider the case where

$$\begin{aligned} p^* :&= \max_{(x_1,x_2)} \{q_1(x_1) + q_2(x_2)\} && (2.38)\\ & x_1 + x_2 \le 5 && (2.39)\\ & x_1, x_2 \in \{0,1,2,3,\dots\} && (2.40)\end{aligned}$$

and the functions q_1 and q_2 are defined as follows:

x	0	1	2	3	4	5
$q_1(x)$	4	7	6	3	8	7
$q_2(x)$	2	5	4	6	5	9

Set,

$$\text{opt} = \max\ ;\ Y = \{0,1,2,3,4,5\}\ ;\ z(y) = \{0,1,2,\dots,5-y\} \qquad (2.41)$$

$$q(y,z) = q_1(y) + q_2(z) \qquad (2.42)$$

In this case the conditional problems would have this form

$$\begin{aligned} p(y) :&= \max_{z\in Z(y)} \{q_1(y) + q_2(z)\}\ ,\ y \in Y && (2.43)\\ &= q_1(y) + \max_{z\in Z(y)} q_2(z) && (2.44)\\ &= q_1(y) + \max\{q_2(z) : z \in \{0,1,2,\dots,5-y\}\} && (2.45)\end{aligned}$$

Hence,

$$\begin{aligned}
p(0) &= q_1(0) + \max\big\{q_2(z) : z \in \{0,1,2,3,4,5\}\big\} = 4+9 = 13\\
p(1) &= q_1(1) + \max\big\{q_2(z) : z \in \{0,1,2,3,4\}\big\} = 7+6 = 13\\
p(2) &= q_1(2) + \max\big\{q_2(z) : z \in \{0,1,2,3\}\big\} = 6+6 = 12\\
p(3) &= q_1(3) + \max\big\{q_2(z) : z \in \{0,1,2,\}\big\} = 3+5 = 8\\
p(4) &= q_1(4) + \max\big\{q_2(z) : z \in \{0,1\}\big\} = 8+5 = 13\\
p(5) &= q_1(5) + \max\big\{q_2(z) : z \in \{0\}\big\} = 7+2 = 9
\end{aligned}$$

Maximizing $p(y)$ over $y \in Y$ would therefore yield

$$\begin{aligned}
p^* &= \max\{p(y) : y \in \{0,1,2,3,4,5\}\} && (2.46)\\
&= \max\{13,13,12,8,13,9\} = 13 && (2.47)
\end{aligned}$$

It follows then that there are three optimal solutions here: $(y,z) = (0,5)$, $(y,z) = (1,3)$ and $(y,z) = (4,1)$. $\square$

Let us now pause here for a moment to reflect on what we have done so far.

2.9 Time Out: Direct Enumeration!

To be sure, the solution procedure outlined above for *Problem P* amounts to no more than a *systematic enumeration* of all the feasible solutions pertaining to this problem. That a strategy of this kind is being bothered with at all, let alone treated in all seriousness as a perfectly legitimate solution procedure for this problem, may perhaps strike some readers as peculiar. Let us therefore examine whether this is indeed as peculiar as it might appear at first.

You will recall that to make the problem with which we started out, namely $p^* := \underset{x\in X}{\text{opt}}\ q(x)$, amenable to a dynamic programming treatment, we had to impute more substance to the solution set X. To this end we assumed that X is a subset of the Cartesian product of *two* sets, thereby obtaining *Problem P*, namely

$$p^* := \underset{\substack{y\in Y\\ z\in Z(y)}}{\text{opt}}\ q(y,z) \tag{2.48}$$

It turns out, however, that whatever inroads were made on the initial indeterminateness of X and q, attributing this property to X falls short of injecting X and q with sufficient content to give the problem a structure that is conducive to analysis. As matters stand then, all that *Problem P* is able to give rise to is an optimality equation that can be solved only through direct *enumeration.* This means, of course, that more needs to be done to imbue *Problem P* with a structure that is amenable to a substantive analysis. As we shall progressively see, the measures that will be taken to bring this about will have the effect of raising direct enumeration to the level of dynamic programming.

It ought to be noted, therefore, that one of the principal theses that I argue in this book is that direct enumeration lies at the bottom of dynamic programming. Namely, that dynamic programming is in fact a distilled form of direct enumeration that is obtained by ridding direct enumeration of some of its blatantly unattractive features. So, the question that I turn to now is:

How is direct enumeration transformed into dynamic programming?

As we shall see, a key element in this process is the concept *equivalent conditional problems*. This concept enables to effect a refinement in the optimality equation induced by the *Principle of Conditional Optimization*. However, before I can show this, I must first explain the concept itself. Indeed, prior to this, we need to remind ourselves of the precise meaning of the notion *equivalent problems* in this context.

2.10 Equivalent Conditional Problems

Consider the following problem:

$$\textit{Problem 1:} \quad q_1 := \operatorname*{opt}_{a\in A} \{c + g(a)\} \tag{2.49}$$

where c is a given constant and g is a real-valued function on some set A. Since (2.49) implies that

$$q_1 = c + \operatorname*{opt}_{a\in A} \; g(a) \tag{2.50}$$

it follows that the value of c is immaterial for determining the optimal solutions to this problem. This entails in turn that solving *Problem 1* is in fact tantamount to solving the following problem:

$$\textit{Problem 2:} \quad q_2 := \operatorname*{opt}_{a\in A} \; g(a) \tag{2.51}$$

The two problems can therefore be considered EQUIVALENT in that both have the same feasible and optimal solution(s). Also, in this particular example, the kinship between these two problems is further manifested in an intimate relation between the optimal values of their respective objective functions, namely $q_1 = c + q_2$.

This clear, let us now examine the notion *equivalent conditional problems* in the framework of *Problem P*.

2.10.1 Example

Suppose that *Problem P* is of the form

$$p^* := \operatorname*{opt}_{\substack{y\in Y\\ z\in Z(y)}} \{v(y)+w(z)\} \tag{2.52}$$

$$Y := \{(y_1,y_2) : y_1+y_2 \leq r, y_1,y_2 \geq 0\} \ , \ r>0 \tag{2.53}$$

$$Z(y) := [0, r-(y_1+y_2)] \ , \ y=(y_1,y_2)\in Y \tag{2.54}$$

In this case the conditional problems would take the form

$$p(y) := \max_{z\in Z(y)} \{v(y)+w(z)\} \ , \ y\in Y \tag{2.55}$$

Since in this context the variable y is treated as a known parameter, it follows that

$$p(y) = v(y) + \max_{z\in Z(y)} w(z) \tag{2.56}$$

$$= v(y) + \max_{0\leq z\leq r-(y_1+y_2)} w(z) \tag{2.57}$$

$$= c + \max_{0\leq z\leq s} w(z) \tag{2.58}$$

where $c=v(y)$ and $s=r-(y_1+y_2)$.

We may conclude then that *Problem P*(y) is equivalent to the following *modified problem*:

$$p'(y) := \max_{z\in Z'(s)} w(z) \ , \ s=r-(y_1+y_2) \tag{2.59}$$

where

$$Z'(s) := [0,s] \ , \ 0\leq s\leq r \tag{2.60}$$

In other words, if we define

$$S : = [0,r] \tag{2.61}$$

$$t(y) : = r-(y_1+y_2) \ , \ y\in Y \tag{2.62}$$

then for every pair $y,y'\in Y$ such that $t(y)=t(y')$, *Problem P*(y) will be equivalent to *Problem P*(y') by virtue of their possessing the same feasible solutions as well as the same optimal solutions. □

At this point I need to introduce the following notation. Let X^* denote the set of optimal solutions to *Problem P*, that is, define

$$X^* := \left\{x^*\in X : q(x^*) = \operatorname*{opt}_{x\in X} q(x)\right\} \tag{2.63}$$

Similarly, define

$$Z^*(y) := \left\{ z^* \in Z(y) : q(y,z^*) := \operatorname*{opt}_{z\in Z(y)} \ q(y,z) \right\} \ , \ y \in Y \tag{2.64}$$

so that by definition $Z^*(y)$ denotes the set of optimal solutions for *Problem* $P(y)$. Because X^* is a subset of X and X itself consists of pairs $x = (y,z)$ such that $y \in Y$ and $z \in Z(y)$, it follows that X^* consists of such pairs as well. Also, note that by definition $Z^*(y) \subseteq Z(y)$, $\forall y \in Y$.

Let us examine then the role of the concept "equivalent conditional problems" in bringing about the refinement of the optimality equation induced by the *Principle of Conditional Optimization*.

2.11 Modified Problems

Consider the following class of problems associated with *Problem* P:

$$\textit{Problem } P'(s)\text{:} \qquad p'(s) := \operatorname*{opt}_{z\in Z'(s)} \ q'(s,z) \ , \ s \in S \tag{2.65}$$

where S is some set, and for every $s \in S$ the set $Z'(s)$ is a subset of $\widetilde{Z}$. I shall refer to *Problem* $P'(s)$ as the MODIFIED PROBLEM AT s.

Clearly, the factor linking *Problem* $P'(s)$ and *Problem* P is the set $\widetilde{Z}$. Also, recall that the definition of *Problem* P postulates that $X \subseteq \widetilde{Y} \times \widetilde{Z}$.

Now, incorporating the above into our discussion of equivalent problems, we can define

$$Z^{\circ}(s) := \left\{ z^{\circ} \in Z'(s) : q'(s,z^{\circ}) = \operatorname*{opt}_{z\in Z'(s)} q'(s,z) \right\} \ , \ s \in S \tag{2.66}$$

Namely, let $Z^{\circ}(s)$ denote the set of optimal solutions for *Problem* $P'(s)$. This phrasing immediately suggests *Problem* $P(y)$ as a connecting link between *Problem* $P'(s)$ and *Problem* P. Consider then the following.

Assumption 2.11.1 *There exists a function t on Y with values in S such that*

$$Z'(t(y)) = Z(y) \ , \ \forall y \in Y \tag{2.67}$$

$$Z^{\circ}(t(y)) = Z^*(y) \ , \ \forall y \in Y \tag{2.68}$$

Let t^{-1} denote the inverse of t, namely define

$$t^{-1}(s) := \{y \in Y : t(y) = s\} \ , \ s \in S \tag{2.69}$$

By construction, if *Assumption 2.11.1* holds, then for every $s \in S$ the set $t^{-1}(s)$ consists of elements of Y whose conditional problems are equivalent.

Continuing with *Example 2.10.1,* notice that if we set

$$q'(s,z) := w(z) \ , \ s \in S, z \in Z'(s) \tag{2.70}$$

then the scheme (S,t) defined by (2.61)-(2.62) and the sets $\{Z'(s) : s \in S\}$ defined by (2.60) conform with the requirements of *Assumption 2.11.1.*

The following is therefore an immediate consequence of the foregoing analysis:

Corollary 2.11.1 *Suppose that Assumption 2.11.1 holds and let δ be any map from S to $\widetilde{Z}$ such that $\delta(s) \in Z^{\circ}(s), \forall s \in S$ for which the set $t^{-1}(s)$ is not empty. Then,*

$$p(y) = q(y, \delta(t(y))) \ , \ \forall y \in Y \tag{2.71}$$

In this case, the optimality equation has this form:

$$p^* = \operatorname*{opt}_{y \in Y} \ q(y, \delta(t(y))) \tag{2.72}$$

Looking at this equation, it is clear that the solution plan that it entails calls for the solution of the conditional problems through the related modified problems. In other words, a solution procedure that would run as follows:

Procedure B:

Step 1: For every element $s \in S$, solve *Problem* $P'(s)$ and set $\delta(s)$ to be any element of $Z^{\circ}(s)$.

Step 2: Determine the value of p^* in accordance with the modified optimality equation (2.72). □

Let us now compare the modified optimality equation (2.72) with the optimality equation (2.27). The idea here is to determine whether it stands to be more efficient than its predecessor. To do this, we need to establish how many conditional problems are solved on average by a single modified problem. So, let us define,

$$E := \frac{|Y|}{|S|} \tag{2.73}$$

where $|A|$ denotes the cardinality of set A, namely $|A|$ is equal to the number of elements contained in A. We now compute the value of E for a concrete case.

2.11.1 Example

Consider the case where the conditional problems are of the form

$$p(y) := \max_{z \in Z(y)} \{v(y) + w(z)\} \ , \ y \in Y \tag{2.74}$$

where

$$Y := \{(y_1, y_2, \dots, y_N) : y_n \in \{0,1\}, 1 \leq n \leq N\} \tag{2.75}$$

$$Z(y) := \left\{ z : z \leq N - \sum_{n=1}^{N} y_n, z \in \{0,1,2,\dots\} \right\} \tag{2.76}$$

Given that for every $1 \leq n \leq N$ the variable y_n can take either one of two values, that is $y_n \in \{0,1\}$, it follows that $|Y| = 2^N$. Now, suppose that to comply with *Assumption 2.11.1* we define

$$S := \{0,1,2,\dots,N\} \tag{2.77}$$

$$t(y) := N - \sum_{n=1}^{N} y_n \ , \ y \in Y \tag{2.78}$$

$$Z'(s) := \{0,1,2,\dots,s\} \ , \ s \in S \tag{2.79}$$

$$q'(s,z) := w(z) \ , \ s \in S, z \in Z'(s) \tag{2.80}$$

Then clearly $|S| = N+1$, and therefore

$$E = \frac{|Y|}{|S|} = \frac{2^N}{N+1} \tag{2.81}$$

Values of $|S|$, $|Y|$ and E corresponding to a sample of values of N are listed below.

N	10	15	20	30
$\lvert S\rvert$	11	16	21	31
$\lvert Y\rvert$	1,024	32,768	1,048,576	11,073,741,824
E	93	2,048	49,932	34,636,833

In a word then, it is patently clear that in this case the set S is likely to be considerably smaller than the set Y. □

2.12 The Role of a Decomposition Scheme

Although, as we have just seen, the optimality equation (2.72) is likely to be more efficient than (2.27) in the sense that it involves less conditional problems, the former still has some serious failings. For one thing, the fact that it requires that a record be kept of the values of $\{\delta(s) : s \in S\}$ can translate in practice into heavy storage (memory) requirements. For instance, should the elements comprising $\widetilde{Z}$ be large arrays, the impact on the computations in Step 1 of *Procedure B* would be huge.

At any rate, even if this factor were totally ignored, there would still be sufficient reasons for seeking further improvements of the foregoing procedure. Let us consider then how such an improvement can be brought about.

Recall that the move that set off the the derivation of the optimality equation was the decomposition of the solution set X into its components, namely

$$X = \{(y, z) : y \in Y, z \in Z(y)\} \tag{2.82}$$

Suppose now that we apply this tactic to the objective function q of *Problem P*. As we shall see next, decomposing the objective function will further distill the optimality equation. In anticipation of this, however, I need to introduce the following:

Assumption 2.12.1 *There exists a real-valued function ρ on $Y \times \mathbb{R}$ such that*

$$q(y, z) = \rho(y, q'(t(y), z)) \ , \ \forall y \in Y, \ z \in Z(y) \tag{2.83}$$

Any collection $(S, t, q', \{Z'(s) : s \in S\})$ satisfying the requirements laid down by *Assumption 2.11.1-2.12.1* will henceforth be referred to as a DECOMPOSITION SCHEME for *Problem P.*

Continuing with *Example 2.11.1,* since by construction

$$q(y, z) = v(y) + q'(t(y), z) \ , \ \forall y \in Y, z \in Z(y) \tag{2.84}$$

we can set

$$\rho(y, a) := v(y) + a \ , \ y \in Y, a \in \mathbb{R} \tag{2.85}$$

The scheme specified by (2.77)-(2.85) thus constitutes a decomposition scheme for the instance of *Problem P* considered in this example.

I shall now proceed to formulate an optimality equation for *Problem P* based on a decomposition scheme and the set of modified problems induced by it. But first I need to point out a number of things.

By definition,

$$p(y) = q(y, z) \ , \ \forall z \in Z^*(y) \tag{2.86}$$

is true for every $y \in Y$.

Hence, if *Assumption 2.12.1* holds, it follows that $Z^*(y) = Z^\circ(t(y))$ for all $y \in Y$ so that

$$p(y) = q(y, z) \ , \ \forall z \in Z^\circ(t(y)) \tag{2.87}$$

Furthermore, if *Assumption 2.11.1* holds as well, then clearly

$$p(y) = \rho(y, q'(t(y), z)) \ , \ \forall z \in Z^\circ(t(y)) \tag{2.88}$$

for all $y \in Y$, so that

$$p(y) = \rho(y, p'(t(y))) \ , \ \forall y \in Y \tag{2.89}$$

recalling that by definition

$$p'(s) = q'(s, z) \ , \ \forall s \in S, \ z \in Z^\circ(s) \tag{2.90}$$

So, *Assumption 2.11.1 - 2.12.1,* in conjunction with (2.27), yield:

Corollary 2.12.1

$$p^* = \operatorname*{opt}_{y\in Y} \ \rho(y, p'(t(y))) \tag{2.91}$$

Continuing with *Example 2.11.1,* in view of the above, we obtain the following optimality equation:

$$p^* = \operatorname*{opt}_{y\in Y}\{v(y) + p'(t(y))\} \tag{2.92}$$

recalling that by definition

$$p'(s) := \operatorname*{opt}_{z\in Z'(s)} \ q'(s,z) \ , \ s \in S \tag{2.93}$$

$$= \operatorname*{opt}_{z\in Z'(s)} \ w(z) \tag{2.94}$$

The optimality equation given by (2.91) suggests the following solution procedure for *Problem P.*

Procedure C:

Step 1: For each $s \in S$, determine the value of $p'(s)$ in accordance with (2.65).

Step 2: Determine the value of p^* by solving the optimization problem on the right-hand side of (2.91). □

The essential point here is that the solution of (2.91) requires only the values of $\{p'(s) : s \in S\}$, each of which is a *scalar.* Let us examine then what are the ramifications of this fact for *Procedure C.*

2.12.1 Example

Consider the following problem

$$p^* := \operatorname*{opt}_{(x_1,\dots,x_N)} \prod_{n=1}^{N} q_n(x_n) \tag{2.95}$$

$$\sum_{n=1}^{N} x_n \leq r \ , \ r > 0 \tag{2.96}$$

$$x_n \in \{0,1,2,\dots\} \ , \ n = 1,2,3,\dots,N \tag{2.97}$$

where $\{q_n\}$ is a sequence of real-valued functions on $[0,r]$ such that

$$q_n(a) > 0 \ , \ \forall 1 \leq n \leq N, a \in [0,r] \tag{2.98}$$

Now, suppose that the solution set is decomposed as follows:

$$X = \{(y,z) : y \in Y, z \in Z(y)\} \tag{2.99}$$

where

$$Y := \big\{y \in [0,r] : y \in \{0,1,2,3,\dots\}\big\} \tag{2.100}$$

$$Z(y) := \left\{z : \sum_{n=1}^{N-1} z_n \leq r - y, z_n \in \{0,1,2,3,\dots\}\right\} \tag{2.101}$$

In this case, the following decomposition scheme will be considered:

$$S := [0,r] \tag{2.102}$$

$$t(y) := r - y \ , \ y \in Y \tag{2.103}$$

$$Z'(s) := \left\{z : \sum_{n=1}^{N-1} z_n \leq s, z_n \in \{0,1,2,3,\dots\}\right\} \tag{2.104}$$

$$q'(s,z) := \prod_{k=1}^{N-1} q_{k+1}(z_k) \tag{2.105}$$

$$\rho(y,a) := q_1(y) \times a \ , \ y \in Y, a \in \mathbb{R} \tag{2.106}$$

To ascertain that this scheme is consistent with *Assumption 2.11.1,* note that by construction

$$q(x_1,x_2,\dots,x_N) = \prod_{n=1}^{N} q_n(x_n) \tag{2.107}$$

$$= q_1(x_1) \times \prod_{k=1}^{N-1} q_{k+1}(x_{k+1}) \tag{2.108}$$

Hence, for any $x = (y,z)$ such that $y \in Y$ and $z \in Z(y)$ we would have

$$q(y,z) = q_1(y) \times \prod_{k=1}^{N-1} q_{k+1}(z_k) \tag{2.109}$$

$$= q_1(y) \times q'(s,z) \ , \ s = t(y) \tag{2.110}$$

Next, the conditional problems would be of this form

$$p(y) := \max_{z \in Z(y)} q(y,z) \ , \ y \in Y \tag{2.111}$$

$$= \max_{z \in Z(y)} \left\{q_1(y) \times \prod_{k=1}^{N-1} q_{k+1}(z_k)\right\} \tag{2.112}$$

whereas the modified problems would be defined thus:

$$p'(s) := \max_{z \in Z'(s)} q'(s,z)$$

$$= \max_{z \in Z'(s)} \prod_{k=1}^{N-1} q_{k+1}(z_k) \tag{2.113}$$

Considering then that by construction

$$Z'(t(y)) = Z(y) \ , \ \forall y \in Y \tag{2.114}$$
$$q(y,z) = q_1(y) \times q'(t(y),z) \ , \ \forall y \in Y, z \in Z(y) \tag{2.115}$$

and in view of the fact that $q_1(y) > 0$, $\forall y \in Y$, it follows from (2.112)-(2.113) that $z \in Z^*(y)$ if, and only if, $z \in Z^o(s)$, $s = t(y)$. Hence,

$$Z^*(y) = Z^\circ(t(y)) \ , \ \forall y \in Y \tag{2.116}$$

In other words, *Assumption 2.11.1* is satisfied. And finally, because by construction

$$\rho(y, q'(t(y),z)) = q_1(y) \times q'(t(y),z) \ , \ y \in Y, z \in Z(y) \tag{2.117}$$
$$= q(y,z) \tag{2.118}$$

it follows that *Assumption2.12.1* is also satisfied, the conclusion thus being that the scheme outlined above is a decomposition scheme for *Problem P*.

Turning now to the solution procedure. Suppose that $N = 100$. As *Procedure B* prescribes solving *Problem P*(y) for every $y \in Y$ and storing the optimal solution recovered, memory requirements will be inordinate. This is so because each element of $Z(y)$ is, in this case, a vector consisting of 99 components. In contrast, *Procedure C* requires only that for each $s \in S$ the scalar $p'(s)$ be stored. □

We again reached a point in the discussion where we need to pause for a critical assessment of our findings thus far.

2.13 Dynamic Programming Problem — Revisited

Having just seen what role a decomposition scheme plays in the derivation of an optimality equation for *Problem P*, the following question is inescapable:

What guarantees are there that a decomposition scheme exits?

In other words, what assurances would we have in a given case that a decomposition scheme is indeed available for the problem concerned?

To be able to bring into sharp focus the vital necessity of finding a satisfactory reply to this question, let us retrace the logical evolution of our discussion.

Recall that I started with this formulation:

$$p^* := \operatorname*{opt}_{(y,z)\in X} q(y,z), \ X \subset \widetilde{Y} \times \widetilde{Z} \tag{2.119}$$

I then went on to decompose the solution set X to obtain

$$\textit{Problem } P: \quad p^* := \operatorname*{opt}_{\substack{y\in Y\\ z\in Z(y)}} \; q(y,z) \tag{2.120}$$

Invoking the *Principle of Conditional Optimization*, namely

$$\operatorname*{opt}_{\substack{y\in Y\\ z\in Z(y)}} \; q(y,z) = \operatorname*{opt}_{y\in Y} \left\{ \operatorname*{opt}_{z\in Z(y)} \; q(y,z) \right\} \tag{2.121}$$

I deduced what I termed a set of conditional problems, namely

$$\textit{Problem } P(y): \quad p(y) := \operatorname*{opt}_{z\in Z(y)} \; q(y,z) \ , \ y \in Y \tag{2.122}$$

thereby obtaining the following optimality equation

$$p^* = \operatorname*{opt}_{y\in Y} \; p(y) \tag{2.123}$$

On showing that the solution procedure called for by this equation would in fact amount to a systematic enumeration of all the feasible solutions for *Problem P,* I invoked the notion of a modified problem to obtain

$$\textit{Problem } P'(s): \quad p'(s) := \operatorname*{opt}_{z\in Z'(s)} \; q'(s,z) \ , \ s \in S \tag{2.124}$$

assuming that a function t can be constructed such that

$$Z'(t(y)) = Z(y) \ , \ \forall y \in Y \tag{2.125}$$

$$Z^{\circ}(t(y)) = Z^*(y) \ , \ \forall y \in Y \tag{2.126}$$

where $Z^*(y)$ and $Z^{\circ}(s)$ denote the set of optimal solutions for *Problem* $P(y)$ and *Problem* $P'(s)$, respectively. This format yielded the following optimality equation.

$$p^* = \operatorname*{opt}_{y\in Y} \; q(y, \delta(t(y))) \ , \ \delta(s) \in Z^{\circ}(s), \forall s \in S \tag{2.127}$$

I then went on to argue that this optimality equation called for further refinement, for which purpose I introduced the notion of a decomposition scheme. In other words, I assumed that there exists a function ρ such that

$$q(y,z) = \rho(y, q'(t(y), z)) \ , \ \forall y \in Y, z \in Z(y) \tag{2.128}$$

This in turn produced the following optimality equation

$$p^* = \operatorname*{opt}_{y\in Y} \; \rho(y, p'(t(y))) \tag{2.129}$$

So, having come full circle, I shall now have to establish under what conditions does a decomposition scheme for *Problem P* exist. To be precise, I shall have to identify sufficient conditions under which any instance of *Problem P* would be certain to have at least one decomposition scheme. And to bring matters to a head, I shall now proceed to update the definition of a dynamic programming problem as follows:

Definition 2.13.1 *Problem P is said to be a* DYNAMIC PROGRAMMING PROBLEM *if it possesses a scheme $(S, t, q', \rho, \{Z'(s) : s \in S\})$ satisfying Assumption 2.11.1 and Assumption 2.12.1.*

With this updated definition, the need to find a satisfactory reply to the question concerning the existence of decomposition schemes takes on a special urgency. Because, what this definition brings out is that a satisfactory reply to this question will have the force of an assurance that dynamic programming rests on a secure foundation.

2.14 Trivial Decomposition Scheme

All it takes is a moment's reflection to conclude that a decomposition scheme for *Problem P* exists *as a matter of principle.* Because, when the requirements laid down by *Assumption 2.11.1 and Assumption 2.12.1* are examined, it turns out that they do no more than impose some minor modeling technicalities on *Problem P.* To see that this is indeed so, consider the following.

Definition 2.14.1 *In relation to Problem P, define the following decomposition scheme:*

$$S := Y \tag{2.130}$$

$$t(y) := y \ , \ \ y \in Y \tag{2.131}$$

$$Z'(t(y)) := Z(y) \ , \ \ y \in Y \tag{2.132}$$

$$q'(t(y), z) := q(y, z) \ , \ \ \forall y \in Y, \ \ z \in Z(y) \tag{2.133}$$

$$\rho(s, a) := a \ , \ \ s \in S, \ \ a \in \mathbb{R} \tag{2.134}$$

I shall refer to this scheme as the TRIVIAL DECOMPOSITION SCHEME *for Problem P.*

It is patently clear that the trivial decomposition scheme fully obeys the requirements of *Assumption 2.11.1* as well as those of *Assumption 2.12.1.* This means of course that this scheme is applicable to **any** instance of *Problem P.*

Or in more general terms, there always exists a decomposition scheme for *Problem P.* Hence,

Corollary 2.14.1 *Any instance of Problem P is a dynamic programming problem irrespective of the particular structure of q, Y and $\{Z(y) : y \in Y\}$.*

Problem P, in other words, is assured to possess the trivial decomposition scheme. The point to note here, however, is that the optimality equation deriving from the trivial scheme is identical to the equation produced by the *Principle of Conditional Optimization*. The implication is then that the optimality equation

$$p^* = \operatorname*{opt}_{y\in Y}\ \rho\left(y, p'\left(t(y)\right)\right) \tag{2.135}$$

would reduce to

$$p^* = \operatorname*{opt}_{y\in Y}\ p'\left(t(y)\right) \tag{2.136}$$

$$= \operatorname*{opt}_{y\in Y}\ p(y) \tag{2.137}$$

In other words, the dynamic programming optimality equation rendered by the trivial decomposition scheme would give rise to *direct enumeration*.

This, of course, is hardly surprising, considering that all that this trivial scheme does is to replace the symbol y by the symbol s, the symbol q by the symbol q', and so on.

Let us now pull together the above findings to see where we stand.

On the one hand we have an assurance that a universally applicable decomposition scheme exists as a matter of principle. On the other, this scheme has been shown to revert dynamic programming to direct enumeration. It is important, therefore, to be clear on the meaning and role of this assurance. The ensuing chapters will progressively show that, for the most part, one would be able to devise decomposition schemes that are far more effective than the trivial scheme. At the same time, it will also emerge that even though a scheme may prove to be more effective than the trivial decomposition scheme, it may still generate *computationally ineffcient* optimality equations.

The conclusion to be drawn from all this is that the assurance that a decomposition scheme exists in principle is of *methodological* importance. It provides the guarantees that one would **always** be able to bring an instance of *Problem P* to a valid dynamic programming optimality equation. *Practically*, however, it is of little help, for it gives no indication whatsoever as to the viability of the resulting optimality equation as far as computation is concerned.

2.15 Summary and a Look Ahead

We saw that the dynamic programming optimality equation is the result of an upgrading of the optimality equation yielded by the *Principle of Conditional Optimization*. This refinement is made possible by the concept of *equivalent conditional problems*, itself reliant on two notions:

- *Decomposition Scheme*
- *Modified Problems*

In the process, I identified a set of basic techniques that dynamic programming deploys to obtain an optimality equation.

This done, I shall next have to show how these concepts and techniques are employed to construct decomposition schemes that are more complex than the universal (trivial) scheme. In other words, I shall have to explain how these are used to handle optimization problems where the solution set X is expressed explicitly as a subset of the Cartesian product of *any* number of sets.

To expound these matters I shall use the multistage decision model as a medium of discourse. So, my next task is to introduce this model.

3

Multistage Decision Model

3.1 Introduction

The problems that are commonly known as *sequential* or *multistage decision problems* are generally considered to be representations of real-world situations where a sequence of decisions is made to attain a certain goal subject to specified constraints. The feature considered the defining characteristic of the decision-making processes manifest in these situations is that a decision made at any given time is affected by its predecessors and invariably affects its successors.

In turn, the mathematical models depicting problems of this kind are termed *sequential* or *multistage decision models*. The stock components comprising such models are as follows:

1. A sequence consisting of elements denoting the *stages* of the decision-making process.
2. A set whose elements represent the *state* of the process in each stage.
3. A collection of sets where each set contains elements designating the *feasible decisions* pertaining to a stage-state pair.
4. A *transition function* embodying the dynamics of the states as they evolve over stages.
5. An *objective function* stipulating the overall return or cost generated by any sequence of decisions.

These are the rudimentary building blocks with which models designed to portray multistage decision processes of various degrees of complexity are constructed.

In this chapter I formulate a prototype model out of these key ingredients with the view to show that this is precisely the means for describing, analyzing and solving optimization problems in general.

To be precise, I argue that its origins notwithstanding, as a mathematical model the multistage decision model can be a tool for the treatment of optimization problems *as such*, including problems that ostensibly are couched in language that is different from that of the multistage decision model.

My chief objective in arguing this point is to explain the rationale behind dynamic programming's approach to optimization problems. Dynamic programming, I contend, more than any other optimization method, regards an optimization problem as an embodiment of a multistage decision process. Naturally, the plan of attack that it puts forward reflects this position. So, to have a good grasp of dynamic programming's treatment of optimization problems, it is essential to have a good grasp of the medium that it uses for this purpose — the MULTISTAGE DECISION MODEL.

Once the model is outlined, I shall describe the rules — namely *policies* — that in this framework govern the determination of the optimal decisions for the underlying problems.

3.2 A Prototype Multistage Decision Model

The model that I am about to set out is designed to serve as a means for the formulation and treatment of optimization problems whose solution set X is a subset of the Cartesian product of a *fixed* number of sets, say $M > 1$. To this end, let us restate our generic optimization problem, namely *Problem P*, as follows:

$$\textit{Problem } P: \qquad p := \underset{(x_1,\dots,x_M)\in X}{\text{opt}} \; q(x_1,\dots,x_M) \tag{3.1}$$

where $X \subseteq X' := X_1 \times X_2 \times \cdots \times X_M$ and q is a real-valued function on X. Let X^* denote the set of optimal solutions to this problem. Assume that X^* is not empty, namely that the problem has at least one optimal solution.

That it is at all conceivable to regard problems of this type as multistage decision problems is borne out by the following observation. Simply let X be a set consisting of the sequences of feasible decisions under consideration and let M be the number of decision *stages*. Then x_m will be interpreted as a decision made at stage m and q as a function indicating the degree of desirability of elements of X. To all intent and purposes then, *Problem P* will constitute a *multistage decision problem.*

The following example illustrates how this would work in the case of a specific instance of the general type defined by *Problem P.*

3.2.1 Example

Consider the problem

$$p := \underset{(x_1,\dots,x_N)}{\text{opt}} \; \sum_{n=1}^{N} q_n(x_n) \tag{3.2}$$

$$\sum_{n=1}^{N} x_n \le r \;,\; x_n \ge 0,\; n = 1,2,3,\dots,N \tag{3.3}$$

Note that because (3.3) implies that

$$0 \le x_n \le r - \sum_{\substack{k=1\\k\neq n}}^{N} x_k \;,\; n = 1,2,3,\dots,N \tag{3.4}$$

we can set

$$X_m := [0,r] \;,\; m = 1,2,3,\dots,M \tag{3.5}$$

Thus, to obtain a formulation consistent with the format of *Problem P,* set

$$M = N \tag{3.6}$$

$$\text{opt} = \max \tag{3.7}$$

$$q(x_1, \ldots, x_M) := \sum_{m=1}^{M} q_m(x_m) \tag{3.8}$$

$$X' := \big[0, r\big]^M \tag{3.9}$$

$$X := \left\{ (x_1, \ldots, x_M) \in X' : \sum_{m=1}^{M} x_m \leq r \right\} \quad \Box \tag{3.10}$$

This established, I shall now state the model that from now on will be considered the prototype against which multistage decision problems are to be tested; and the blueprint according to which optimization problems falling under the *Problem P* format are to be recast as multistage decision problems.

Definition 3.2.1 *A* MULTISTAGE DECISION MODEL *is a collection* (N, S, D, T, S_1, g) *where*

- $\cdot$ *N is a positive integer specifying the number of* DECISION STAGES *comprising the decision-making process. Define* $\mathbb{N} := \{1, 2, 3, \ldots, N\}$. *Thus, for $N = \infty$ the set N denotes the set of positive integers.*
- $\cdot$ *S is a non-empty set entitled the* STATE SPACE. *Its elements are called* STATES.
- $\cdot$ *D is a function such that to each pair $(n, s) \in \mathbb{N} \times S$ it assigns a subset of some set $\mathbb{D}$. The set $D(n, s)$ is referred to as the* SET OF FEASIBLE DECISIONS PERTAINING TO STATE s AT STAGE n. *If the set $D(n, s)$ is empty, the state s is considered to be non-feasible at stage n. The set $\mathbb{D}$ is called the* DECISION SPACE.
- $\cdot$ *T is a function on $\mathbb{N} \times S \times \mathbb{D}$ with values in S called the* TRANSITION FUNCTION. *For any triplet (n, s, x) such that $n \in \mathbb{N}$, $s \in S$, and $x \in D(n, s)$, the element $s' = T(n, s, x)$ designates the state at stage $n + 1$ resulting from the application of decision x to state s at stage n.*
- $\cdot$ *S_1 is a non-empty subset of S whose elements are named* INITIAL STATES.
- $\cdot$ *g is a real-valued function on $S_1 \times \mathbb{D}^N$ termed the* OBJECTIVE FUNCTION. *The value yielded by $g(s, x_1, x_2, \ldots, x_N)$ is understood to specify the overall return, namely cost or benefit, arising from the decisions $(x_1, \ldots, x_N)$ made at stages $1, 2, 3, \ldots, N$ respectively, given that the initial state of the process is $s \in S_1$.*

Figure 3.1 gives a schematic description of the dynamics of the decision-making process at a typical stage.

Occasionally, we shall have to distinguish between two cases of the value of N:

- $\cdot$ *Truncated* processes: N is *finite*

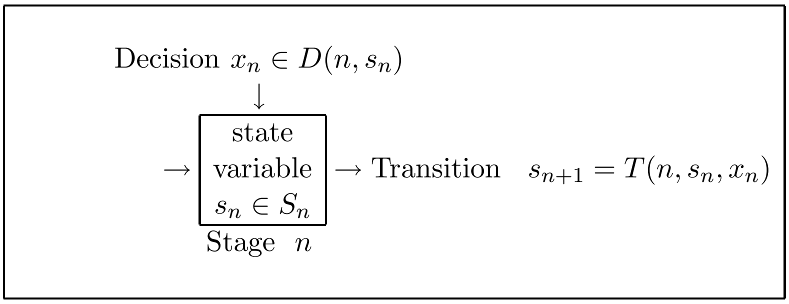

Decision $x_n \in D(n, s_n)$
state
variable
$s_n \in S_n$
Stage n
Transition $s_{n+1} = T(n, s_n, x_n)$

Figure 3.1: A scheme of the state dynamics

· *Non-truncated* process: N is not finite.

Note that in the case of truncated processes there are N decision stages at each of which a decision is made. The final state of the process is observed at stage $n = N + 1$ where the process terminates and no decision is made. The state observed at the final stage $N + 1$, namely s_{N+1}, will be referred to as the FINAL STATE of the process.

In the case of non-truncated processes, there is no final stage. However, if the sequence of states $s_1, s_2, s_3 \ldots$ converges to some state $\hat{s} \in S$, then this state can be regarded as the final state of the process.

Armed with this model, let us now proceed to examine the following (parametric) family of optimization problems.

Definition 3.2.2 *Problem P(s),* $s \in S_1$:

$$f(s) := \underset{(x_1,\ldots,x_N)}{\text{opt}} \; g(s, x_1, x_2, \ldots, x_N) \tag{3.11}$$

$$x_n \in D(n, s_n) \ , \ n \in \mathbb{N} \tag{3.12}$$

$$s_{n+1} = T(n, s_n, x_n) \ , \ n \in \mathbb{N} \tag{3.13}$$

$$s_1 = s \tag{3.14}$$

I shall refer to Problem $P(s)$ as the INITIAL PROBLEM AT s. *Let $X(s)$ denote the set of feasible solutions for Problem $P(s)$, namely define*

$$X(s) := \{(x_1, \ldots, x_N) : x_n \in D(n, s_n), n \in \mathbb{N}\} \tag{3.15}$$

where $s_1 = s$, and $s_{n+1} = T(n, s_n, x_n), n \in \mathbb{N}$.

Also, let $X^*(s)$ denote the set of optimal solution for Problem $P(s)$. □

Unlike *Problem P,* in whose case the decision-making *process*, notably the multistage aspect thereof, had to practically be read into the formulation, *Problem $P(s)$* very directly and unmistakably presents us with a situation that conveys the following:

> The initial state of the decision-making process being $s_1 = s$, a sequence of decisions $(x_1, \ldots, x_N)$ needs to be determined such that the decisions

(a) are feasible, that is, $x_n \in D(n, s_n)$, where s_n is the state generated by s_1 and $(x_1, \ldots, x_{n-1})$, and

(b) they optimize the overall cost or benefit stipulated by the objective function g.

The expression $s_{n+1} = T(n, s_n, x_n)$ indicates that the dynamics of the process is governed by the transition function T. That is, it indicates that bringing the sequence of decisions $(x_1, \ldots, x_N)$ into play at the initial stage $n = 1$, where the state is s_1, sets off the sequence of states $(s_2, \ldots, s_N)$, where $s_2 = T(1, s_1, x_1)$, $s_3 = T(2, s_2, x_2)$, etc. And finally, incorporating the set S_1 into the definition of *Problem P(s)* gives the initial state the status of a *parameter*. Now let us go back to our example.

Continuing with *Example 3.2.1,* the constraint defined by (3.4) suggests that the state variables should be defined as follows:

$$s_1 = r \ , \ S_1 = \{r\} \tag{3.16}$$

$$s_n = r - \sum_{k=1}^{n-1} x_k \ , \ n > 1 \tag{3.17}$$

In this case, (3.4), in conjunction with (3.17), imply that

$$0 \leq x_n \leq s_n \ , \ n \in \mathbb{N} \tag{3.18}$$

so that to comply with *Definition 3.1.2*, the following can be set:

$$S = D = [0, r] \tag{3.19}$$

$$D(n, s_n) = [0, s_n] \ , \ s_n \in S, n \in \mathbb{N} \tag{3.20}$$

Next, from (3.17) it follows that the transition function T can be defined as follows:

$$s_{n+1} = r - \sum_{k=1}^{n} x_k \tag{3.21}$$

$$= r - \sum_{k=1}^{n-1} x_k - x_n \tag{3.22}$$

$$= s_n - x_n \tag{3.23}$$

that is,

$$T(n, s_n, x_n) = s_n - x_n \ , \ n \in \mathbb{N}, s_n \in S, x_n \in D(n, s_n) \tag{3.24}$$

Finally, the objective function g is defined as follows:

$$g(s, x_1, x_2, \ldots, x_N) = q(x_1, \ldots, x_N) \tag{3.25}$$

$$= \sum_{n=1}^{N} q_n(x_n) \tag{3.26}$$

Hence, *Problem P(s)* would assume the form:

$$f(s) := \underset{(x_1,\dots,x_N)}{\text{opt}} \sum_{n=1}^{N} q_n(x_n) \ , \ s \in S_1 \tag{3.27}$$

$$0 \leq x_n \leq s_n \ , \ n \in \mathbb{N} \tag{3.28}$$

$$s_{n+1} = s_n - x_n \ , \ n \in \mathbb{N} \tag{3.29}$$

Observe that should the objective be to solve *Problem P(s)* for a range of values of r, say, $r \in [0, r^*]$, we would simply reset

$$D = S = S_1 = [0, r^*] \tag{3.30}$$

We have just seen then that the problem in *Example 3.2.1* can be framed in terms of both the format of *Problem P* — call it *Format P* — and the format of *Problem P(s)* — call it *Format P(s)*. This being so, the following question arises: is this merely a coincidence or is it rather an indication that the two formats are interchangeable in principle?

I shall work out an answer to this question in the next section. But before I can do this I need to clarify a number of technical details.

Assumption 3.2.1 *For each $s \in S_1$ the initial problem at s, namely Problem $P(s)$, has at least one optimal solution.*

This assumption clearly holds if for each $s \in S_1$ the set $X(s)$ is finite and non-empty. More generally, it holds in cases obeying *Weierstrass' Theorem*, that is, cases where for each $s \in S_1$ the set $X(s)$ is compact under an appropriate topology on $X(s)$ and where the function $g(s, \cdot)$ is continuous on $X(s)$.

A sequence of decisions $(x_1, \dots, x_N)$ will be deemed *feasible* with respect to state $s \in S_1$, if this sequence will constitute a feasible solution to *Problem $P(s)$*, namely if $(x_1, \dots, x_N) \in X(s)$. Similarly, this sequence will be deemed *optimal* with respect to state s if it constitutes an optimal solution to *Problem $P(s)$*.

3.3 Problem vs Problem Formulation

Going back now to the question posed in the foregoing section, the issue boils down to this: what is the relation between *Format P* and *Format $P(s)$*? To be able to treat this question formally, let

A := set of all optimization problems subsumed by *Format P*.
B := set of all optimization problems subsumed by *Format $P(s)$*.

The question is then: what is the relation between the sets A and B? The answer to this is:

Theorem 3.3.1

$$A = B \tag{3.31}$$

In other words, *Format P* and *Format P(s)* designate one and the same type of optimization problem. I shall prove this in two stages.

Lemma 3.3.1

$$B \subseteq A \tag{3.32}$$

PROOF. Let (N, S, D, T, S_1, g) be any multispage decision model and let s be any element of S_1. To establish that $B \subseteq A$ we need to show that there exist a sequence of sets $(X_m : m \in N)$; a set $X \subseteq X_1 \times X_2 \times \cdots \times X_M$; and a real valued function q on X such that

$$X = X(s) \tag{3.33}$$

and

$$q(x_1, \dots, x_N) = g(s, x_1, \dots, x_N) \ , \ \forall (x_1, \dots, x_N) \in X(s) \tag{3.34}$$

keeping in mind that $X(s)$ denotes the set of feasible solutions for *Problem P(s)*. Observe then that since by definition $X(s) \subseteq \mathbb{D}^N$, we can set $M = N$ and $X_m = \mathbb{D}$, whereupon $X(s)$ clearly emerges as a subset of the Cartesian product of the sets $(X_m : m \in \mathbb{N})$. This completes the proof, as X and q can now be *defined* by (3.33) and (3.34) respectively. □

The reason that this result flows so readily from its antecedents is that *Problem P,* by definition, encompasses any conceivable optimization problem whose feasible solutions can be represented as sequences of a *fixed length.* Given then that all the feasible solutions pertaining to *Problem P(s)* are sequences of a fixed length, it follows that any problem falling under *Format P(s)* would necessarily fall under *Format P.*

Lemma 3.3.2

$$A \subseteq B \tag{3.35}$$

Before providing a formal argument for this lemma, let us first elucidate its import. *Lemma 3.3.2* argues that, given an instance of *Problem P,* there exists a multistage decision model (N, S, D, T, S_1, g) and an initial state $s_1 \in S_1$, under whose terms *Problem P(s_1)* would be equivalent to *Problem P* in that both would have *the same objective function, the same feasible solutions* and hence *the same optimal solutions.* It should be appreciated that because this proposition contends that this is true for *any* instance of *Problem P,* the said multistage decision model and initial state must by necessity be of a simple structure — one that is readily derived from the structures of q and X. With this in mind, consider now the following.

Definition 3.3.1 *In the framework of Problem P let the multistage decision model* $(N^\circ, S^\circ, D^\circ, T^\circ, S_1^\circ, g^\circ)$ *be defined as follows:*

- *Stages. Set* $N^\circ := M$ *so that by definition* $\mathbb{N}^\circ := \{1, 2, \dots, M\}$.
- *State space. Define,*

$$S^\circ := \bigcup_{n=1}^{M+1} S_n^\circ \tag{3.36}$$

$$S_n^\circ := \{(n-1) \uparrow y : y \in X\}\,,\ 1 \leq n \leq M+1 \tag{3.37}$$

and $k \uparrow y$ *denotes the sequence comprising the first* k *elements of the sequence* y*, namely*

$$k \uparrow y := (y_1, y_2, \dots, y_k) \tag{3.38}$$

Thus, by construction S_n° *is the projection of* X *on* $X_1 \times X_2 \times \cdots \times X_{n-1}$.

- *Feasible decisions. Define,*

$$D^\circ(n,s) := \begin{cases} \varnothing & ,\ s \in S_n^\circ \\ \{x : x \in X_n, (s,x) \in S_{n+1}^\circ\} & ,\ s \notin S_n^\circ \end{cases}\ ,\ n \in \mathbb{N} \tag{3.39}$$

where $\varnothing$ *denotes the empty set.*

- *Transition function. In line with the definition of* S *in (3.37), set*

$$T^\circ(n,s,x) := (s,x)\ ,\ n \in \mathbb{N}\ ,\ s \in S_n^\circ, x \in D^\circ(n,s) \tag{3.40}$$

- *Initial states. In line with (3.37) let* $S_1^\circ := \{\varnothing\}$*, where* $\varnothing$ *denotes the empty sequence.*
- *Objective function. Define,*

$$g^\circ(s,z) := q(z)\ ,\ s \in S_1^\circ, z \in X \tag{3.41}$$

I shall refer to $(N^\circ, S^\circ, D^\circ, T^\circ, S_1^\circ, g^\circ)$ *as the* TRIVIAL MULTISTAGE DECISION MODEL *induced by Problem P.*

The following observations need to be made with regard to certain components of this model:

1. Since S_1° is the singleton $\{\varnothing\}$, it follows from (3.41) that g° *is identical to* q *on* X. Thus, to show that *Problem P(s),* $s = \varnothing$, is equivalent to *Problem P* it will suffice to demonstrate that $X(s) = X$, recalling that $X(s)$ denotes the set of feasible solutions for *Problem P(s).*
2. By construction, the set S_n° is the projection of X on the set $X_1 \times X_2 \times \cdots \times X_{n-1}$. It therefore follows from (3.39)-(3.40) that if the sequence $(x_1, \dots, x_M)$ is an element of X, it must satisfy the condition

$$x_n \in D^\circ(n, s_n^\circ)\ ,\ \forall n \in \mathbb{N} \tag{3.42}$$

where $s_1^\circ = \varnothing$ and $s_{k+1}^\circ = T^\circ(n, s_k^\circ, x_n)$, the implication thus being that

$$X \subseteq X(s_1^\circ) \tag{3.43}$$

3. Let $y = (x_1, \dots, x_N)$ be any element of $X(s_1^\circ)$ and consider the sequence $\{s_n^\circ\}$ generated by y and s_1°. Then, (3.39)-(3.40) imply that

$$s_{n+1}^\circ = (x_1, x_2, \dots, x_n) \in S_{n+1}^\circ \ , \ \forall n \in \mathbb{N} \tag{3.44}$$

This can be restated as follows:

$$y \in X(s_1^\circ) \Longrightarrow (n \uparrow y) \in (n \uparrow X) \ , \ n \in \mathbb{N} \tag{3.45}$$

where $(n \uparrow X)$ denotes the projection of X on $X_1 \times X_2 \times \cdots \times X_n$.

4. If M is finite, then (3.45) entails that

$$y \in X(s_1^\circ) \Longrightarrow (M \uparrow y) \in (M \uparrow X) \tag{3.46}$$

from which it follows that

$$y \in X(s_1^\circ) \Longrightarrow y \in X \tag{3.47}$$

for M being finite clearly implies $(M \uparrow X) = X$.
The inference is therefore that if M is finite then

$$X(s_1^\circ) \subseteq X \tag{3.48}$$

Thus, in cases where M is finite, (3.48) in conjunction with (3.43) provide that $X = X(s_1^\circ)$. Since we have already established that g° is identical to q on X the following must be true.

Corollary 3.3.1 *If M is finite then the initial problem induced by the trivial multistage decision model is equivalent to Problem P.*

This will insure that *Lemma 3.3.2,* hence *Theorem 3.3.1,* hold true when M is finite. To clinch the proof of *Theorem 3.3.1,* however, we still need to consider the case where $M = \infty$, that is, where *Problem P* takes the following form

$$\textit{Problem P:}\ \ p := \underset{(x_1,x_2,\dots)}{\text{opt}} \ q(x_1, x_2\ , \dots) \ , \ X \subseteq X' := \mathop{\times}_{m=1}^{\infty} X_m \tag{3.49}$$

Let K be any positive integer and set

$$X'_m := \begin{cases} X_m & ,1 \leq m < K \\ (k-1) \downarrow X' & ,m = K \end{cases} \tag{3.50}$$

where

$$n \downarrow X' := \{(x_{n+1}, \dots, x_N) : (x_1, \dots, x_N) \in X'\} \tag{3.51}$$

By construction then,

$$X' = X'_1 \times X'_2 \times \cdots \times X'_K \tag{3.52}$$

and therefore *Problem P* can be redefined as follows:

$$\textit{Problem P: } p := \underset{(x_1,\dots,x_K)}{\text{opt}} \quad q(x_1,\dots,x_K), \ X \subseteq X' = \mathop{\times}_{m=1}^{K} X'_m \tag{3.53}$$

What this means then is that, if $M = \infty$, *Problem P* will easily lend itself to a formulation under whose terms the solution set X will be expressed as a subset of the Cartesian product of a finite number of sets. Formally then,

PROOF of *Lemma 3.3.2.*
If M is finite invoke *Corollary 3.3.1.* If $M = \infty$ rephrase *Problem P* so as to allow X to be stated as a subset of the Cartesian product of a finite number of sets and then invoke *Corollary 3.3.1.* □

At this point I need to explain why the distinction between the case where M is finite and that where $M = \infty$ is called for. To do this, it is sufficient to point out that in the latter case the conditions $y \in X'$ and $(m \uparrow y) \in (m \uparrow X)$ for all $m \geq 1$ do not guarantee that $y \in X$. There is therefore no assurance that the trivial multistage decision model is such that $X(s_1^\circ)$ is equal to X.

3.3.1 Example

Consider the case where $M = \infty$, $X_m = \{0, 1\}$, $\forall m$, and

$$X = \{y : y \in \{0,1\}^\infty \ , \ y \neq (1,1,1,1,\dots)\} \tag{3.54}$$

That is, let X be the set consisting of all the infinite-dimension boolean vectors excluding the vector $y' = (1, 1, 1, \dots)$. Then, clearly $(m \uparrow y') \in (m \uparrow X)$ for all $m \geq 1$, yet $y' \notin X$. □

In short, the difficulty arising in the case where $M = \infty$ is that, failing the assurance that $X(s_1^\circ)$ is a subset of X, there is no certainty that $X = X(s_1^\circ)$. The implication is therefore that additional guarantees must be provided to insure that *any* instance of *Problem P* will allow a recasting as an instance of *Problem P(s)* for some $s \in S_1$. Consider then the following.

Definition 3.3.2 *The solution set X of Problem P is said to be a* REGULAR SUBSET *of the set X' if*

$$\big\{y \in X' \textit{and } (m \uparrow y) \in (m \uparrow X), \forall m \in \mathbb{N}\big\} \Longrightarrow y \in X \tag{3.55}$$

An instance of Problem P is said to be REGULAR, *if for this instance, X is a regular subset of X'.*

Notice that this definition entails that if M is finite, then any instance of *Problem P* is regular. The following example features a regular instance of *Problem P* in a case where $M = \infty$. More generally, the above regularity condition requires X to be a *closed set* under the assumed topology.

3.3.2 Example

Consider the case where $M = \infty$, $X_m = [0, r]$, $r \geq 0$, $\forall m$, and

$$X := \left\{ (x_1, x_2, \dots) : \sum_{m=1}^{\infty} a_m x_m \leq r, 0 \leq x_m \leq r \right\} \tag{3.56}$$

I shall show that if the coefficients $\{a_m\}$ are non-negative then X is a regular subset of X'. Let y be any element of X' such that $(m \uparrow y) \in (m \uparrow X)$ for all $m \geq 1$. What needs to be shown is that if the coefficients $\{a_m\}$ are non-negative then $y \in X$. Note then that since the coefficients $\{a_m\}$ are non-negative and $X_m = [0, r]$, it follows that

$$(m \uparrow X) = \left\{ (x_1, \dots, x_m) : \sum_{k=1}^{m} a_k x_k \leq r, 0 \leq x_k \leq r \right\} \ , \ \forall m \geq 1 \tag{3.57}$$

If we define

$$r'_m := \sum_{k=1}^{m-1} a_k y_k \ , \ m \geq 1 \tag{3.58}$$

then the sequence $\{r'_m\}$ will be monotone non-decreasing. Furthermore, since $(m \uparrow y) \in (m \uparrow X)$, $\forall m \geq 1$, it also follows that $r \geq r'_m$, $\forall m \geq 1$, meaning that the sequence $\{r'_m\}$ is bounded above by r. Hence, its limit is also bounded above by r. The implication is therefore that

$$\sum_{m=1}^{\infty} a_m y_m \leq r \tag{3.59}$$

which in turn implies that $y \in X$. It follows that X is a regular subset of X'. Thus, any instance of *Problem P* with a solution set of the form defined above is regular. $\square$

In short then, if $M = \infty$ one can either "truncate" X as outlined by (3.50)-(3.53) and then invoke *Corollary 3.3.1,* or settle for *Problem P* being regular and invoke the following.

Lemma 3.3.3 *Given any regular instance of Problem P, there exist a multistage decision model* (N, S, D, T, S_1, g) *and an initial state* $s_1 \in S_1$ *such that* $N = M$ *and Problem* $P(s_1)$ *is equivalent to this instance of Problem P.*

PROOF. Consider any regular instance of *Problem P*. Then, in view of (3.43) it follows that X is a subset of the set of feasible solutions to *Problem* $P(s_1^\circ)$ induced by the multistage decision model, that is, $X \subseteq X(s_1^\circ)$. Also, this instance being regular, implies that $X(s_1^\circ) \subseteq X$, thus implying that $X = X(s_1^\circ)$. Hence, since g° is identical to q on X it follows that the two problems are equivalent. $\square$

Clearly, it would be utterly counter-productive to become embroiled in all the minor technical matters that arise from the optimization problem not being regular. Therefore, the following assumption will accompany us through the remainder of the book.

Assumption 3.3.1 *Problem P is regular, namely X is a regular subset of $X' := X_1 \times X_2 \times \cdots \times X_M$.*

The analysis of the relation between *Format P* and *Format $P(s)$* can now be summarized as follows.

However disparate the two formats appear to be on the surface, the truth is that both give expression to the same class of optimization problems — any problem in this class has the property that all its feasible solutions are sequences of a *fixed length.*

Since all it takes to bring a problem to conform with this requirement is to allow it to absorb, if necessary, *dummy* decision variables, stages, etc., we may conclude that any optimization problem can, in principle, be formulated in terms of *Format $P(s)$*; or, in other words, that any optimization problem can as a matter of principle assume the format of a multistage decision problem. The converse is also true, any multistage decision problem can assume the format of *Problem P.* In view of our findings in *Chapter 2,* it ought to be clear though that of interest to us are *only* cases where $M \geq 2$.

And a final point: as the initial state proves to be the only conceivable object in *Format $P(s)$* that is capable of providing the link between *Problem P* and *Problem $P(s)$,* it should be expected to figure as a parameter in *Problem P* when *Problem $P(s)$* is restated to assume the *Format P.*

3.3.3 Example

Consider the case where the family of initial problems is of the following form:

$$f(s) := \max_{(x_1,x_2,\dots)} \sum_{n=1}^{\infty} q_n(x_n) \ , \ s \in S_1 = S := [0,r], r \geq 0 \tag{3.60}$$

$$0 \leq x_n \leq s_n \ , \ n \in \mathbb{N} := \{1,2,3,\dots.\} \tag{3.61}$$

$$s_{n+1} = \alpha(s_n - x_n) \ , \ n \in \mathbb{N}, 0 < \alpha < 1 \tag{3.62}$$

where $s_1 = s$. Since the above implies that

$$0 \leq x_n \leq s_n \leq s_{n+1} \leq r \tag{3.63}$$

reformulating *Problem P(s)* in terms of *Format P* would mean setting $M = \infty$ and $X_m = [0,s]$, in which case $X' = [0,s]^\infty$.

To determine the composition of the decision set X, notice that the sequence of states generated by an initial state $s_1 = s$ and a sequence of decisions $(x_1, \ldots, x_N)$ is as follows:

$$
\begin{aligned}
s_1 &= s && (3.64)\\
s_2 &= \alpha(s_1 - x_1) && (3.65)\\
s_3 &= \alpha(s_2 - x_2) = \alpha(\alpha(s_1 - x_1) - x_2) = \alpha^2 s_1 - \alpha^2 x_1 - \alpha x_2 && (3.66)\\
s_4 &= \alpha(s_3 - x_3) = \alpha(\alpha^2 s_1 - \alpha^2 x_1 - \alpha x_2 - x_3)\\
&= \alpha^3 s_1 - \alpha^3 x_1 - \alpha^2 x_2 - \alpha x_3 && (3.67)
\end{aligned}
$$

and by induction

$$s_n = s_1 \alpha^{n-1} - \sum_{k=1}^{n-1} x_k \alpha^{n-k} \ , \ n \geq 1 \tag{3.68}$$

Hence, for any given $s_1 \in S_1$, *Problem* $P(s_1)$ would be restated in the form of *Problem P* as follows:

$$p := \max_{(x_1, x_2, \ldots) \in X} \sum_{m=1}^{\infty} q_m(x_m) \tag{3.69}$$

where

$$X := \left\{ (x_1, x_2, \ldots) : 0 \leq x_m \leq s_1 \alpha^{m-1} - \sum_{k=1}^{n-1} \alpha^{m-k} x_k \right\} \tag{3.70}$$

As noted above, the initial state is stated explicitly in the formulation of the set X. □

In summary, having shown that any optimization problem capable of assuming *Format P* will by necessity lend itself to *Format P(s)* and vice versa, we can now conclude that problems normally regarded as multistage decision problems are in truth ordinary optimization problems.

The format that I shall adopt henceforth as a standard problem formulation framework will, however, be *Format* $P(s)$, that is, the multistage decision model. Because, as we shall see throughout, the type of constructs and terminology that it furnishes, that is, the type of setting that it lays out for the treatment of an optimization problem — proves precisely suitable for the purposes of dynamic programming.

3.4 Policies

One of the merits of formulating optimization problems in terms of a multistage decision model is that this position provides an extremely illuminating perspective on these problems. It inspires a broadening of the notion of a solution to an optimization problem, thus furnishing deeper insight into the problem and the ways of tackling it. Specifically, the integration of such concepts as "state" and "stage" in the formulation of an optimization problem invites an approach to the solution that no longer, as in the case of *Problem P,* makes do with simply determining the optimal decisions. Rather, the goal now becomes to identify the RULES behind these decisions, namely the rules according to which these decisions are made. Rules of this kind are termed POLICIES.

So, a policy being a rule prescribing decisions, a *feasible policy* is a rule yielding a feasible sequence of decisions, and an *optimal policy* is a rule — necessarily feasible — whose application yields an optimal sequence of decisions.

Let us now examine what kind of decision rules would the multistage decision model give rise to.

You will recall that one of the functions of the *state variable*, within the framework of the multistage decision model, is to supply the information used in identifying the set of feasible decisions pertaining to a given stage. More precisely, the state observed at any stage of the process provides the information that is required to determine which decisions are feasible for this stage. Clearly, this implies that a policy would be defined in terms of its relation to the state variable. But what is more, it further follows that the most natural policy imaginable in this framework would be a policy of the form $\delta = \delta(n, s)$, that is a policy such that *all* it would require for engendering the decision x_n at stage n is the information furnished by the state presently observed at this stage.

Stated more rigorously, such a policy is a function on $\mathbb{N} \times S$ with values in $\mathbb{D}$, where to comply with the feasibility constraints $x_n \in D(n, s_n)$, $\forall n \in \mathbb{N}$, it would satisfy the condition $\delta(n, s_n) \in D(n, s_n)$ for any pair $(n \in \mathbb{N}, s_n \in S)$ for which the set $D(n, s_n)$ is not empty. Policies possessing this characteristic are called MARKOVIAN.

Ideally, the Markovian policy would be optimal for all the initial states $s \in S_1$ of *Problem* $P(s)$. However, as we shall shortly see, the conditions underlying the multistage decision model — by themselves — do not furnish the assurance that such a policy always exists. We therefore have to investigate what conditions need to hold in order to secure the existence of a Markovian policy that is optimal for all the initial states. Also, to deepen our understanding of the Markovian policy it is desirable to examine it vis-a-vis other policies that can be used in the framework of the multistage decision model. So, in addition to the class of Markovian policies, I shall define three other classes of policies in an increasing order of complexity of their *domain of definition.*

Class # 1: Let $\Delta^{(1)}$ denote the set of all the functions defined on $\mathbb{N}$ with values in $\mathbb{D}$, and let δ be any element of $\Delta^{(1)}$. In this case, the decision generated at stage n is defined as follows: $x_n = \delta(n)$. A policy of this type then merely simulates a sequence of decisions. That is, although it is a function, all that δ does is to enunciate a sequence of decisions, which it is important to note, takes no notice whatsoever of the initial state of the process, nor of any subsequent states and decisions.

Theorem 3.4.1 *Let s be any element of S_1. Then, there exists a policy $\delta \in \Delta^{(1)}$ that is optimal for Problem P(s).*

PROOF. Let s be any element of S_1 and let $(x_1^*, \dots, x_N^*)$ be any optimal solution to *Problem P(s).* Consider now the element $\delta \in \Delta^{(1)}$ defined as follows:

$$\delta(n) = x_n^* \ , \ n \in \mathbb{N} \tag{3.71}$$

Observe that applying this policy will always generate the sequence of decisions $(x_1^*, \dots, x_N^*)$ at stage $1, 2, \dots, N$ respectively, regardless of what the initial state happens to be. Since $(x_1^*, \dots, x_N^*)$ is an optimal solution to *Problem $P(s)$*, it follows that δ is optimal for *Problem $P(s)$.* $\square$

It goes without saying, however, that under the terms of the multistage decision model there is no assurances that there exists a policy $\delta \in \Delta^{(1)}$ that is optimal (simultaneously) for **all** the initial states. In fact, this is the case only in trivial problems.

3.4.1 Example

Consider the following case.

$$f(s) := \max_{(x_1,\dots,x_N)} \sum_{n=1}^{N} x_n^2 \tag{3.72}$$

$$x_n \in D(n,s) := [0, s_n] \ , \ n \in \mathbb{N} \tag{3.73}$$

$$S_1 = S = D = [0, r] \ , \ r \geq 0 \tag{3.74}$$

$$T(n, s_n, x_n) = s_n - x_n \ , \ n \in \mathbb{N} \tag{3.75}$$

Since $s_1^* := 0$ is an element of S_1 and the only feasible solution for *Problem* $P(s_1^*)$ is the sequence $(x_1^*, \dots, x_N^*)$ where $x_n^* = 0$, $\forall n \in \mathbb{N}$, it follows that the only element of $\Delta^{(1)}$ that is *feasible* for all the initial states is the policy defined as follows:

$$\delta(n) = 0 \ , \ \forall n \in \mathbb{N} \tag{3.76}$$

However, because this policy is not optimal for $s > 0$, it follows that no policy in $\Delta^{(1)}$ is optimal for all the initial states. $\square$

The next class of policies that I consider is more intricate than the previous class in that policies of this type take account of the initial state of the process.

Class # 2: Let $\Delta^{(2)}$ denote the set of all the functions defined on $S_1 \times \mathbb{N}$ with values in $\mathbb{D}$, and let δ be any element of $\Delta^{(2)}$. In this case the sequence of decisions $(x_1, \ldots, x_N)$ generated by δ and an initial state $s_1 \in S_1$ is defined thus:

$$x_n = \delta(s_1, n) \ , \ n \in \mathbb{N} \tag{3.77}$$

Consider now the following obvious result.

Theorem 3.4.2 *There exists a policy $\delta \in \Delta^{(2)}$ which is optimal (simultaneously) for all the initial problems.*

Proof. For each $s \in S_1$, let δ_s be any element of $\Delta^{(1)}$ that is optimal for the initial problem at s, recalling that *Theorem 3.4.1* provides that such policies exist. Now, consider the policy $\delta \in \Delta^{(2)}$ defined as follows

$$\delta(s, n) = \delta_s(n) \ , \ s \in S_1, n \in \mathbb{N} \tag{3.78}$$

By construction, the sequence of decisions generated by applying δ to some $s \in S_1$ is identical to the sequence of decisions generated by applying δ_s to s. Because for each $s \in S_1$ the policy δ_s is optimal for *Problem P(s)*, it follows that δ is optimal for all the initial states. □

In short, once the initial state is a factor in the policies' domain of definition, it is possible to assure the existence of at least one policy that is optimal for all the initial problems. One need hardly point out the benefit of being able to count on this option.

Class # 3: The effect of incorporating the initial state in the policy's domain of definition clearly suggests that a policy, which not only takes into account the initial state, but all the ensuing states and decisions up to the current stage of the process should also be considered. Such a policy's domain of definition would be the set:

$$H := \bigcup_{n=1}^{N} H_n \tag{3.79}$$

where

$$H_1 := S_1 \tag{3.80}$$

$$H_n := \{(s_1, x_1, s_2, x_2, \ldots, s_n) : x_k \in D(k, s_k), 1 \leq k \leq n)\} \tag{3.81}$$

for $2 \leq n \leq N$, where $s_1 \in S_1$, and $s_{k+1} = T(k, s_k, x_k), 1 \leq k \leq n$.

The import of these definitions is that H_n is composed of sequences consisting of n states and $n-1$ decisions. Each sequence represents a realizable history of the decision process which ends at stage n, including the state observed at this stage. The set H consists of all the feasible partial and complete *histories* of the decision making process, where a typical (partial) history is say,

$$h_n = (s_1, x_1, s_2, x_2, \ldots, s_n) \in H_n \tag{3.82}$$

Now, let $\Delta^{(3)}$ denote the set of all the functions defined on H with values in D such that if $h_n = (s_1, x_1, s_2, x_2, \ldots, s_n) \in H$ then $\delta(h_n) \in D(n, s_n)$. Obviously, policies of this kind take into account, at each stage, the totality of information available about the process up to the stage in question. As such, they are without any doubt, the most comprehensive deterministic policies imaginable.

Because the elements of $\Delta^{(2)}$ can be expressed as degenerate cases of the elements of $\Delta^{(3)}$, the implication of *Theorem 3.4.2* for this case is as follows:

Corollary 3.4.1 *There exists a policy* $\delta \in \Delta^{(3)}$ *that is optimal (simultaneously) for all the initial states.*

And now, let us turn to the class of policies that are central in dynamic programming.

3.5 Markovian Policies

First, we need to remind ourselves of the precise meaning of the concept *Markovian*. You will recall then that an object defined as *Markovian* is understood to be "devoid of memory". The idea here is that the only trace of realized histories that is discernible in the object concerned is the impact made on it by the state being observed.

It is important to be clear on what is and what is not meant by the impact being limited to the state being observed. What *is* meant is that the effect of a realized history on the object is *narrowed down* to the effect of the state being observed. What *is not* meant is that the object is thereby rendered detached from or totally uninfluenced by the states and decisions realized in the preceding stages.

Indeed, if this were so it would have amounted to a contradiction in terms, as the state at any given stage is affected by the previous state and decision through the dynamics $s_{n+1} = T(n, s_n, x_n)$.

We have already encountered a number of Markovian objects in the discussion. For instance, the transition function T is clearly Markovian, and so is the function D which specifies the feasible decisions available at each stage.

A *policy* labeled *Markovian* would therefore be a decision-making rule generating decisions such that the decision at stage n is determined solely on grounds of s_n — the state observed at stage n. The formal definition of policies of this type is as follows.

Definition 3.5.1 *Let Δ denote the set of all the functions δ defined on $\mathbb{N}\times S$ with values in $\mathbb{D}$ such that $\delta(n,s) \in D(n,s)$ for each pair $(n,s) \in \mathbb{N}\times S$ for which the set $D(n,s)$ is not empty. Policies that are members of this set are said to be* Markovian.

Now, if $\delta \in \Delta$ then $x_n = \delta(n,s)$ is understood to be the decision generated by δ at stage n given that the state s is observed at this stage. Therefore, the initial state being $s \in S_1$, bringing δ into play generates the following sequence of states and decisions:

$$s_1 = s \tag{3.83}$$

$$x_1 = \delta(1,s_1) \tag{3.84}$$

$$s_2 = T(1,s_1,x_1) \tag{3.85}$$

$$x_2 = \delta(2,s_2) \tag{3.86}$$

$$\cdots \quad \cdots$$

$$\cdots \quad \cdots$$

$$s_{n+1} = T(n,s_n,x_n)\ ,\ x_n = \delta(n,s_n) \tag{3.87}$$

Next, because the elements of $\Delta^{(1)}$ are degenerate cases of the elements of Δ, the implication of *Theorem 3.4.1* for this case is:

Corollary 3.5.1 *For every initial state $s \in S_1$ there exists a Markovian policy that is optimal for Problem $P(s)$.*

However, as demonstrated by the following example, there is no assurance that Δ contains a policy that is optimal for *all* the initial states.

3.5.1 Example

Consider the case where *Problem $P(s)$* is defined as follows:

$$f(s_1) := \max_{(x_1,x_2)} (s_1)\times(x_1)\times(x_2)\ ,\ s_1 \in S_1 := \{-1,1\} \tag{3.88}$$

$$x_n \in D(n,s_n) := \{-|s_n|, |s_n|+n\}\ ,\ n = 1,2 \tag{3.89}$$

$$s_2 = T(1,s_1,x_1) := x_1 \tag{3.90}$$

where $|a|$ denotes the absolute value of $a \in \mathbb{R}$.

Case 1. $s_1 = 1$.
Problem P(1) has a unique optimal solution, that is $X^*(1) = \{(2,3)\}$, which yields

$$f(1) = 1\times 2\times 3 = 6 \tag{3.91}$$

Therefore, any Markovian policy that is optimal for *Problem P(1)* must satisfy the conditions

$$\delta(1,1) = 2 \tag{3.92}$$

$$\delta(2,2) = 3 \tag{3.93}$$

observing that applying $x_1 = 2$ to the initial state $s_1 = 1$ yields the state

$$s_2 = x_1 = 2 \tag{3.94}$$

Case 2. $s_1 = -1$.
Problem P(1) has the unique optimal solution $z = (x_1 = 2, x_2 = -2)$, which yields

$$f(-1) = (-1) \times (2) \times (-2) = 4 \tag{3.95}$$

Therefore, any Markovian policy that is optimal for *Problem* $P(-1)$ must satisfy the conditions

$$\delta(1, -1) = \ \ 2 \tag{3.96}$$
$$\delta(2, 2) = -2 \tag{3.97}$$

again, because applying $x_1 = 2$ to the initial state $s_1 = -1$ yields the state $s_2 = x_1 = 2$. Now, as no Markovian policy can satisfy both (3.93) and (3.97), it follows that no Markovian policy can, in this case, be optimal for both $s_1 = 1$ and $s_1 = -1$.

Note that in contrast to this state-of-affairs *Theorem 3.4.2* implies that the set $\Delta^{(2)}$ contains at least one policy that is optimal for both *Problem* $P(1)$ and *Problem* $P(-1)$. And to be sure, a quick scan of this set reveals that the policy defined by

$$\delta(s_1, n) := \begin{cases} 2 & , \ s_1 = 1, n = 1 \\ 3 & , \ s_1 = 1, n = 2 \\ 2 & , \ s_1 = -1, n = 1 \\ -2 & , \ s_1 = -1, n = 2 \end{cases} \tag{3.98}$$

is indeed optimal for both *Problem* $P(1)$ and *Problem* $P(-1)$. □

What we have just encountered then is a situation where both D and T are Markovian, yet none of the Markovian policies turned out to be optimal for all the initial problems. However surprising this situation may appear at first, it has a ready explanation. At this stage, it suffices to indicate that to ensure that at least one of the Markovian policies is optimal for all the initial states, the *objective function g* must also have appropriate Markovian properties.

3.6 Remarks on the Notation

The above discussion has clearly brought out the need to explain the thinking behind the notation that will be used henceforth to denote the state and decision variables in their various manifestations.

One need hardly point out the importance of good notation for a lucid exposition and that every effort should be made in this regard. In dynamic programming, framing good notation is a fairly demanding task as one is continually required to strike a proper balance between the need to give accurate expression to nuances of relationships between objects and that of guarding against unduly cluttering the symbolic representations with excessive detail.

For instance, in the case of the state and decision variables, we need to distinguish between *generic* state and decision variables, namely elements of S and D respectively, and states and decisions pertaining to a given *stage*.

Clearly, this means that the most logical thing to do is to use conventional *subscripting* to indicate that certain states and decisions are affiliated with certain stages. Employing this convention will thus enable saying that s_n designates the state pertaining to stage n and x_n a decision pertaining to stage n.

However, as excessive subscripting tends to encumber the notation, I shall, whenever possible, avoid subscripting the state and decision variables, leaving it to the **context** to bring out the distinction between the generic case and the instances associated with given stages.

For example, consider the case where we want to say that the transition function under consideration is of the form

$$T(n, s_n, x_n) = s_n - x_n \tag{3.99}$$

Since the stage variable is an explicit argument of T, subscripting s and x is clearly superfluous. I shall therefore write

$$T(n, s, x) := s - x \tag{3.100}$$

instead. Similarly, I shall write

$$D(n, s, x) := [0, s] \tag{3.101}$$

rather than

$$D(n, s_n) := [0, s_n] \tag{3.102}$$

In view of my special use of the subscripting convention I shall be unable to use it to denote *components* of s and x. So, for this purpose I shall use parentheses. For example, $s_n(k)$ will denote the k-th component of s_n.

Finally, to designate a *sequence of decisions* I shall use the following notation: the symbol y will typically denote a generic sequence of decisions of the form $(x_1, x_2, \ldots, x_k)$ whereas z will denote its remainder, namely $(x_{k+1}, x_{k+2}, \ldots, x_N)$. This choice of symbols is consistent with the convention adopted in *Chapter 2.*

3.7 Summary

In conclusion, I sum up the main points of this chapter. We saw that although manifestly dissimilar, *Format P* and *Format $P(s)$* are in effect different formulations of the same class of optimization problems. To be precise, they are different statements of a prototype optimization problem typified by its feasible solutions being sequences of a *fixed length.*

We saw that all it takes to formulate an optimization problem — which ostensibly seems to express a situation different from the one captured by the multistage decision model — in terms of the multistage decision model is to phrase it so as to satisfy this model's fundamental requirements. We are thus reminded that problems do not confront us with labels attached to them. Namely, that it is not a case of the problem being or not being a "multistage decision problem" but rather of the formulation rendering it as such. The fact that we take shortcuts to refer to problems via appellations should in no way obscure this fact. It is extremely important to appreciate this point to avoid misrepresentations of the objects comprising the formulations. Strictly speaking, these are merely mathematical entities.

The principal advantage of *Format $P(s)$* over *Format P* is in the battery of new concepts — stage, state, transitions, etc. — that it makes available for the formulation of an optimization problem. This type of phrasing of an optimization problem inspires an understanding of the solution that reaches behind the optimal decisions themselves, to identify the rules, to wit policies, that operate at bottom.

Of the four types of policies described in this chapter, the Markovian class was singled out as the most characteristic of dynamic programming. We saw, however, that under the terms of the deterministic multistage decision model delineated here, the existence of a Markovian policy that is optimal for all the initial problems is not a foregone conclusion.

In the next chapter I shall describe the derivation of a dynamic programming optimality equation for *Problem $P(s)$*; and in subsequent chapters I shall demonstrate that the optimal policies yielded by this equation are indeed Markovian.

3.8 Bibliographic Notes

The multistage decision model defined in this chapter is basically in line with what has come to be viewed over the years as a standard model for formulating deterministic multistage decision problems (e.g. Bellman [1957a], Nemhauser [1966], Yakowitz [1969], White [1969], Elmaghraby [1970], Sniedovich [1986a]). In certain formulations of this model the states

$(s_2, s_3, s_4, \dots, s_N)$ are treated as explicit arguments of the objective function g. Notice, however, that in my treatment of this model, as outlined in this chapter, these states are taken to be uniquely determined by s_1 and $(x_1, \dots, x_N)$, that is

$$s_{n+1} = T(n, s_n, x_n) \ , \ n \in \mathbb{N} \tag{3.103}$$

This position dispenses with the need to make them explicit arguments of the objective function.

Another model — studied in *Chapter 10* — that has become a staple model in dynamic programming is designed specifically for problems where the sequences of feasible solutions are of various lengths, and the functions D and T are independent of the stage variable. This model is used extensively in automata theory (Karp and Held [1967], Ibaraki [1973, 1974, 1975, 1976]), graph theory, and in other areas (e.g. Mitten [1964], Denardo and Mitten [1967], Morin and Esogbue [1970], Morin [1982]).

And finally, I need to point out that to provide for an adequate treatment of *stochastic* problems, the list of policies defined in *Section 3.3* needs to be extended to include what are known as *randomized* policies (e.g. Denardo [1965], Hinderer [1970], Sobel [1970], Rossman [1977], Sniedovich and Nielsen [1983]).

4

Dynamic Programming — An Outline

4.1 Introduction

In this chapter I trace out the derivation of a dynamic programming functional equation for *Problem* $P(s)$. My main objective is to show how the techniques identified in the analysis of *Problem P* in *Chapter 2* yield this equation when an optimization problem is stated in terms of *Format* $P(s)$. I therefore do not, at this stage, go into the specifics of how the constructs of *Format* $P(s)$ that figure in the derivation process are arrived at. I defer discussion on this matter to the second part of the book. On the other hand, I take up a number of general questions whose appreciation, I believe, is vital for a correct understanding of the derivation process. In particular, I explain what is meant by the contention that an instance of *Problem P* is a dynamic programming problem; and I show that **any** regular instance of *Problem P* is a dynamic programming problem.

4.2 Preliminary Analysis

You will recall that our basic supposition about *Problem P* in *Chapter 2* was that the solution set X is a subset of the Cartesian product of two sets, that is, we assumed that $M = 2$. Assuming the same about *Problem* $P(s)$, let us first establish the main outlines of the process that yields the dynamic programming optimality equation for an optimization problem cast in terms of *Format* $P(s)$.

Assume then that the objective function g is now termed g_1. In this case, *Problem* $P(s)$ would have this form:

$$f_1(s) := \operatorname*{opt}_{(x_1,x_2)} \; g_1(s, x_1, x_2) \ , \ s \in S_1 \tag{4.1}$$

$$x_1 \in D(1, s), x_2 \in D(2, s_2), s_2 = T(1, s, x_1) \tag{4.2}$$

First, the problem would be restated as follows:

$$f_1(s) := \operatorname*{opt}_{\substack{x_1 \in D(1,s)\\ x_2 \in D(2,s_2)\\ s_2 = T(1,s,x_1)}} \; g_1(s, x_1, x_2) \ , \ s \in S_1 \tag{4.3}$$

whereupon an appeal to the *Principle of Conditional Optimization* would yield the following:

$$f_1(s) = \operatorname*{opt}_{x_1 \in D(1,s)} \left\{ \operatorname*{opt}_{\substack{x_2 \in D(2,s_2)\\ s_2 = T(1,s,x_1)}} \; g_1(s, x_1, x_2) \right\} \ , \ s \in S_1 \tag{4.4}$$

The family of conditional problems induced by (4.4) would then have this form:

Problem $P(s,x) : s \in S_1, x \in D(1,s)$.

$$f_1(s,x) := \underset{\substack{x_2 \in D(2,s_2) \\ s_2 = T(1,s,x)}}{\text{opt}} \ g_1(s,x,x_2) \tag{4.5}$$

so that (4.4)-(4.5) would yield

$$f_1(s) = \underset{x \in D(1,s)}{\text{opt}} \ f_1(s,x) \ , \ \forall s \in S_1 \tag{4.6}$$

We would then proceed to define

$$S_2 := \{T(1,s,x) : s \in S_1, x \in D(1,s)\} \tag{4.7}$$

and assume that there exist a real-valued function g_2 on $S_2 \times \mathbb{D}$ and a real-valued function ρ on $S_1 \times D \times \mathbb{R}$ such that

$$g_1(s,x_1,x_2) = \rho(s,x_1,g_2(T(1,s,x_1),x_2)) \tag{4.8}$$

for all $s \in S_1$, $x_1 \in D(1,s)$, $x_2 \in D(2,s_2)$, and $s_2 = T(1,s,x_1)$. And because (4.2), in conjunction with (4.4), imply that

$$f_1(s,x) = \underset{\substack{x_2 \in D(2,s_2) \\ s_2 = T(1,s,x)}}{\text{opt}} \ \rho(s,x,g_2(s_2,x_2)) \ , \ \forall s \in S_1, x \in D(1,s) \tag{4.9}$$

the family of modified problems induced by the function g_2 would have this form:

Problem $P'(s) : s \in S_2$.

$$f_2(s) := \underset{x \in D(2,s)}{\text{opt}} \ g_2(s,x) \tag{4.10}$$

To bring the function f_1 defined by (4.4) to the form of the function f_2 we would assume that the following holds:

Assumption: Markovian Condition. *Let $X^*(s,x)$ denote the set of optimal solutions for Problem $P(s,x)$ and let $X'(s)$ denote the set of optimal solutions to Problem $P'(s)$. Then,*

$$X^*(s,x) = X'(T(1,s,x)) \ , \ \forall s \in S_1, x \in D(1,s) \tag{4.11}$$

What gives this condition its Markovian quality is its implying that for every pair $(s \in S_1, x \in D(1,s))$, the set of optimal solutions for *Problem* $P(s,x)$ is determined solely on the grounds of $s' = T(1,s,x)$. In other words,

$$X^*(s,x) = X^*(s'',x'') \tag{4.12}$$

for any quadruplet (s, x, s'', x'') such that $s, s'' \in S_1$, $x \in D(1, s)$, $x'' \in D(1, s'')$, and $T(1, s, x) = T(1, s'', x'')$.

Next, given that by definition

$$g_2(T(1, s, x), x_2) = f_2(T(1, s, x)) \ , \forall x_2 \in X'(T(1, s, x)) \tag{4.13}$$

for any pair $(s \in S_1, x \in D(1, s))$, the *Markovian Condition* ensures that

$$g_2(T(1, s, x), x_2) = f_2(T(1, s, x)) \ , \ \forall x_2 \in X^*(s, x) \tag{4.14}$$

And finally, since by definition,

$$f_1(s, x) = \ \rho(s, x, g_2(T(1, s, x), x_2)) \ , \ \forall x_2 \in X^*(s, x) \tag{4.15}$$

it follows from (4.5)-(4.6) that

$$f_1(s, x) = \rho(s, x, f_2(T(1, s, x))) \tag{4.16}$$

Hence, (4.3), in conjunction with (4.7), yield

$$f_1(s) = \operatorname*{opt}_{x \in D(1,s)} \ \rho(s, x, f_2(T(1, s, x))) \ , \ \forall s \in S_1 \tag{4.17}$$

This, then, is the dynamic programming optimality equation for *Problem $P(s)$* when $N = 2$.

If we ignore the fine details, the main thrust of the process can be described as follows: We begin by defining,

$$f_1(s) := \operatorname*{opt}_{\substack{x_1 \in D(1,s) \\ x_2 \in D(2,s_2) \\ s_2 = T(1,s,x_1)}} \ g_1(s, x_1, x_2) \ , \ s \in S_1 \tag{4.18}$$

whereupon the *Principle of Conditional Optimization* yields

$$f_1(s) = \operatorname*{opt}_{x_1 \in D(1,s)} \left\{ \operatorname*{opt}_{\substack{x_2 \in D(2,s_2) \\ s_2 = T(1,s,x_1)}} \ g_1(s, x_1, x_2) \right\} \ , \ \forall s \in S_1 \tag{4.19}$$

Assuming that g_1 is decomposable, then

$$f_1(s) = \operatorname*{opt}_{x_1 \in D(1,s)} \left\{ \operatorname*{opt}_{\substack{x_2 \in D(2,s_2) \\ s_2 = T(1,s,x_1)}} \ \rho(s, x_1, g_2(s_2, x_2)) \right\} \ , \ \forall s \in S_1 \tag{4.20}$$

Furthermore, if the *Markovian Condition* holds, then clearly

$$f_1(s) = \operatorname*{opt}_{x_1 \in D(1,s)} \ \rho(s, x_1, f_2(T(1, s, x_1))) \ , \ \forall s \in S_1 \tag{4.21}$$

Let us now examine this typical dynamic programming functional equation in the framework of a specific problem.

4.2.1 Example

Consider the case where

$$S = S_1 = D = [0,r]\ ,\ r \geq 0 \tag{4.22}$$

$$D(n,s) = [0,s]\ ,\ s \in S,\ n = 1,2 \tag{4.23}$$

$$T(n,s,x) = s - x\ ,\ s \in S,\ x \in D(n,s),\ n = 1,2 \tag{4.24}$$

$$g_1(s,x_1,x_2) = q_1(x_1) + q_2(x_2) \tag{4.25}$$

Now, define the following:

$$g_2(s,x) := q_2(x)\ ,\ s \in S_2 := [0,r],\ x \in D(2,s) \tag{4.26}$$

$$\rho(s,x,a) := q_1(x) + a,\ s \in S_1\ ,\ x \in D(1,s,x),\ a \in \mathbb{R} \tag{4.27}$$

By construction then,

$$\begin{aligned}\rho(s,x_1,g_2(T(1,s,x),x_2)) &= q_1(x_1) + g_2(T(1,s,x_1),x_2) && (4.28)\\ &= q_1(x_1) + q_2(x_2) && (4.29)\\ &= g_1(s,x_1,x_2) && (4.30)\end{aligned}$$

as required by (4.8). The conditional problems therefore have this form:

$$\begin{aligned}f_1(s,x) : &= \underset{\substack{x_2 \in D(2,s_2)\\ s_2 = T(1,s,x)}}{\text{opt}} \{q_1(x) + q_2(x_2)\} && (4.31)\\ &= \underset{\substack{0 \leq x_2 \leq s_2\\ s_2 = s-x}}{\text{opt}} \{q_1(x) + q_2(x_2)\} && (4.32)\\ &= \underset{0 \leq y \leq s-x}{\text{opt}} \{q_1(x) + q_2(y)\} && (4.33)\\ &= q_1(x) + \underset{0 \leq y \leq s-x}{\text{opt}} q_2(y) && (4.34)\end{aligned}$$

The modified problems take this form:

$$\begin{aligned}f_2(s) &= \underset{y \in D(2,s)}{\text{opt}} q_2(y)\ ,\ s \in S_2 && (4.35)\\ &= \underset{0 \leq y \leq s}{\text{opt}} q_2(y) && (4.36)\end{aligned}$$

Note that as the definitions of $f_2(s)$ and $f_1(s,x)$ imply that $y \in X^*(s,x)$ if, and only if, $y \in X'(T(1,s,x))$, it follows that the *Markovian Condition* holds. The dynamic programming optimality equation is therefore of this form:

$$\begin{aligned}f_1(s) &= \underset{x \in D(1,s)}{\text{opt}} \{q_1(x) + f_2(T(1,s,x))\}\ ,\ s \in S_1 && (4.37)\\ &= \underset{x \in D(1,s)}{\text{opt}} \{q_1(x) + f_2(s-x)\}\ ,\ 0 \leq s \leq r && (4.38)\\ &= \underset{0 \leq x \leq s}{\text{opt}} \{q_1(x) + f_2(s-x)\} \qquad \square && (4.39)\end{aligned}$$

This clear, let us now consider how the process evolves when $N > 2$.

4.3 Markovian Decomposition Scheme

As the underlying premise is that $N > 2$, then for every $n \geq 2$ let S_n denote the set of all the feasible states pertaining to stage n, namely define the following:

$$S_{n+1} := \{T(n,s,x) : s \in S_n, x \in D(n,s)\} \ , \ n \in \mathbb{N} \tag{4.40}$$

recalling that S_1 is part of the model. Also, for every pair ($n \in \mathbb{N}$, $s \in S_n$) define

$$X(n,s) := \{(x_n, \dots, x_N) : x_m \in D(m, s_m), n \leq m \leq N\} \tag{4.41}$$

where $s_n = s$, and $s_{m+1} = T(m, s_m, x_m), n \leq m \leq N$.

Since by construction, for any $s \in S_1$ the set $X(1,s)$ consists of all the feasible solutions to *Problem* $P(s)$, the latter can be stated as follows:

$$\textit{Problem } P(s): \qquad f(s) := \operatorname*{opt}_{z \in X(1,s)} g(s,z) \ , \ s \in S_1 \tag{4.42}$$

And now, let us examine what conditions are assumed to hold in this case to enable the derivation of a dynamic programming functional equation.

First consider the following:

Definition 4.3.1 *Let (N, S, D, T, S_1, g) be a multistage decision model. The objective function g is said to be* SEPARABLE *if there exist a real-valued function ρ and a sequence of real-valued functions $G = (g_n : n \in \mathbb{N})$ such that*

ρ is defined on $\mathbb{N} \times S \times \mathbb{D} \times \mathbb{R}$

g_n is defined on $S_n \times \mathbb{D}^{N-n+1}$ *respectively,* $n \in \mathbb{N}$

$$g_1(s,z) = g(s,z) \ , \ \forall s \in S_1, z \in X(1,s) \tag{4.43}$$

$$g_n(s,x,z) = \rho(n,s,x,g_{n+1}(T(n,s,x),z)) \tag{4.44}$$

for all $1 \leq n < N, s \in S_n, x \in D(n,s)$ and $z \in X(n+1, T(n,s,x))$.

We consider (ρ, G) a DECOMPOSITION SCHEME *, ρ a* COMPOSITION FUNCTION *and for each $n \in \mathbb{N}$ we refer to g_n as the* MODIFIED OBJECTIVE FUNCTION *at stage n.*

Assume then that the objective function is separable and let (ρ, G) be its decomposition scheme. As we have seen, the derivation process is set off by the *Principle of Conditional Optimization*. To be able bring the principle into play in the case where $N > 2$, we first define the following family of problems:

Definition 4.3.2 *Problem P(n,s)* , $n \in \mathbb{N}, s \in S_n$:

$$f_n(s) := \operatorname*{opt}_{z \in X(n,s)} g_n(s,z) \tag{4.45}$$

We shall refer to Problem $P(n,s)$ as the MODIFIED PROBLEM *at (n,s). Let $X^*(n,s)$ denote the set of optimal solutions for Problem $P(n,s)$, namely, define*

$$X^*(n,s) := \{z \in X(n,s) : g_n(s,z) = f_n(s)\} \ , \ n \in \mathbb{N}, s \in S_n \tag{4.46}$$

Now, by construction,

$$X(n,s) = \{(x,z) : x \in D(n,s), z \in X(n+1, T(n,s,x))\} \tag{4.47}$$

for all $n \in \mathbb{N}$ and $s \in S_n$. It therefore follows from (4.45) that

$$f_n(s) = \operatorname*{opt}_{\substack{x\in D(n,s)\\ z\in X(n+1,s')\\ s'=T(n,s,x)}} g_n(s,x,z) \ , \ n \in \mathbb{N}, s \in S_n \tag{4.48}$$

Thus, applying the *Principle of Conditional Optimization* to the right-hand side of (4.48) yields

$$f_n(s) = \operatorname*{opt}_{x\in D(n,s)} \left\{ \operatorname*{opt}_{\substack{z\in X(n+1,s')\\ s'=T(n,s,x)}} g_n(s,x,z) \right\} \ , \ n \in \mathbb{N}, \ s \in S_n \tag{4.49}$$

The set of inner problems in (4.49) can formally be defined as follows.

Definition 4.3.3 *Problem $P(n,s,x) : n \in \mathbb{N}, s \in S_n, x \in D(n,s)$.*

$$f_n(s,x) := \operatorname*{opt}_{\substack{z\in X(n+1,s')\\ s'=T(n,s,x)}} g_n(s,x,z) \tag{4.50}$$

We shall refer to Problem $P(n,s,x)$ as the CONDITIONAL PROBLEM *at (n,s,x). Let $X^*(n,s,x)$ denote the set of optimal solutions for Problem $P(n,s,x)$.*

Before we can continue I need to point out that although g_n acts as the objective function at stage $n \in \mathbb{N}$ for both the set of modified problems and the set of conditional problems pertaining to n, in the case of *Problem $P(n,s,x)$* the function g_n is optimized only with respect to $(x_{n+1}, \dots, x_N)$ with $x_n = x$, whereas in the case of *Problem $P(n,s)$* it is optimized with respect to $(x_n, \dots, x_N)$. Going back now to (4.49)-(4.50), observe that these yield the following:

Corollary 4.3.1

$$f_n(s) = \operatorname*{opt}_{x\in D(n,s)} f_n(s,x) \ , \ \forall 1 \le n < N, s \in S_n \tag{4.51}$$

In other words, (4.51) is derived directly on grounds of the *Principle of Conditional Optimization*, irrespective of the particular structure of the composition function ρ in a given problem. The following is another condition of fundamental importance.

Definition 4.3.4 *A decomposition scheme* (ρ, G) *is said to be* MARKOVIAN *if*

$$X^*(n,s,x) = X^*(n+1, T(n,s,x)) \ , \ \forall 1 \leq n < N, s \in S_n, x \in D(n,s) \quad (4.52)$$

As in the case of the *Markovian Condition* defined in *Section 4.1,* the Markovian character of (ρ, G) is in its implying that the set of optimal solutions for any conditional problem is equal to the set of optimal solutions for the modified problem that it generates.

This entails in turn that the set of optimal solutions for *Problem* $P(n,s,x)$ is determined solely on the basis of the value of $s' = T(n,s,x)$. That is,

$$X^*(n,s,x) = X^*(n,s',x') \quad (4.53)$$

for any collection (n,s,x,s',x') such that $n \in \mathbb{N}$, $s, s' \in S_n$, $x \in D(n,s)$, $x' \in D(n,s')$ and $T(n,s,x) = T(n,s',x')$.

Clearly, this is a typical Markovian property. Now, let us examine a concrete case to see how these conditions would come into play.

4.3.1 Example

Consider the following optimization problem:

$$p := \operatorname*{opt}_{(x_1,\dots,x_N)} \sum_{n=1}^{N} q_n(x_n) \quad (4.54)$$

$$\sum_{n=1}^{N} x_n \leq r \ , \ r \geq 0 \quad (4.55)$$

In keeping with the *Problem* $P(s)$ format, set

$$S = D = [0,r] \quad (4.56)$$

$$D(n,s) = [0,s] \ , \ n \in \mathbb{N}, s \in S \quad (4.57)$$

$$T(n,s,x) = s - x \ , \ n \in \mathbb{N}, s \in S, x \in D(n,s) \quad (4.58)$$

$$S_1 = \{r\} \quad (4.59)$$

$$g(s, x_1, x_2, \dots, x_N) = \sum_{n=1}^{N} q_n(x_n) \quad (4.60)$$

so that

$$S_2 = \{T(1,s,x) : s \in S_1, x \in D(1,s)\} \quad (4.61)$$

$$= \{r - x : 0 \leq x \leq r\} = [0,r] \quad (4.62)$$

$$= S \quad (4.63)$$

and by induction,

$$S_n = S = [0,r] \ , \ \forall 1 \leq n \leq N+1 \quad (4.64)$$

Consider now the decomposition scheme (ρ, G) whose composition function is expressed as follows

$$\rho(n, s, x, a) := q_n(x) + a \ , \ 1 \le n < N, s \in S_n, x \in D(n, s) \tag{4.65}$$

and the modified objective functions are of this form:

$$g_n(s, x_n, x_{n+1}, \dots, x_N) := \sum_{k=n}^{N} q_k(x_k) \tag{4.66}$$

By construction therefore,

$$\begin{align} g_n(s, x_n, x_{n+1}, \dots, x_N) &= q_n(x_n) + \sum_{k=n+1}^{N} q_k(x_k) \tag{4.67}\\ &= q_n(x_n) + g_{n+1}(s', x_{n+1}, x_{n+2}, \dots, x_N) \tag{4.68}\\ &= \rho(n, s, x_n, g_{n+1}(s', x_{n+1}, \dots, x_N)) \tag{4.69}\end{align}$$

where $s' = T(n, s, x_n)$.

This means that (ρ, G) is a proper decomposition scheme. The following exercise will demonstrate that the scheme is Markovian. Observe that the set of conditional problems has this form:

$$\begin{align} f_n(s, x) &:= \underset{\substack{z \in X(n+1, s') \\ s' = T(n,s,x)}}{\text{opt}} g_n(s, x, z) \tag{4.70}\\ &= \underset{\substack{z \in X(n+1, s') \\ s' = T(n,s,x)}}{\text{opt}} \{q_n(x) + g_{n+1}(s', z)\} \tag{4.71}\end{align}$$

Since x is treated as known, it follows that

$$f_n(s, x) = q_n(x) + \underset{\substack{z \in X(n+1, s') \\ s' = T(n,s,x)}}{\text{opt}} g_{n+1}(s', z) \tag{4.72}$$

Thus, in view of the second term on the right-hand side of this equation being identical to the modified problem at $(n+1, s')$, it follows that *Problem* $P(n, s, x)$ and *Problem* $P(n+1, T(n, s, x))$ are equivalent, to thereby indicate that the decomposition scheme is Markovian. $\square$

Before I can proceed to formulate the dynamic programming optimality equation by means of the Markovian decomposition scheme two facts need to be pointed out.

First, it is characteristic of any decomposition scheme that for each initial state $s \in S_1$, the modified problem at s is equivalent to the initial problem associated with s. Namely *Problem* $P(1, s)$ is equivalent to *Problem* $P(s)$ for any $s \in S_1$. This means of course that solving *Problem* $P(1, s)$ is tantamount to solving *Problem* $P(s)$. For this reason our focus will be on the modified problem.

Second, unless it is postulated that any modified problem and any conditional problem induced by the decomposition scheme have at least one optimal solution, a mass of technical details that are marginal to the discussion would have to be accounted for. To avoid this I shall proceed on the assumption that:

Assumption 4.3.1 *$X^*(n,s) \neq \varnothing, \forall n \in N, s \in S_n$ and $X^*(n,s,x) \neq \varnothing, \forall 1 \leq n < N, s \in S_n$ where $\varnothing$ denotes the empty set.*

Needless to say, this assumption also guarantees that the sets $X(n,s)$ and $X(n,s,x)$ are not empty, namely that each modified problem and each conditional problem have at least one *feasible* solution.

4.4 Optimality Equation

Assume that *Problem P(s)* has a decomposition scheme (ρ, G) and let (n,s,x) be any triplet such that $1 \leq n < N, s \in S_n$ and $x \in D(n,s)$. Now, by definition,

$$g_n(s,x,z) = f_n(s,x) \ , \ \forall z \in X^*(n,s,x) \tag{4.73}$$

so that (4.44) yields

$$\rho(n,s,x,g_{n+1}(T(n,s,x),z)) = f_n(s,x) \ , \ \forall z \in X^*(n,s,x) \tag{4.74}$$

Next, because by definition,

$$g_{n+1}(T(n,s,x),z) = f_{n+1}(s,x) \ , \ \forall z \in X^*(n+1,T(n,s,x)) \tag{4.75}$$

we can conclude the following:

Corollary 4.4.1 *If the decomposition scheme (ρ, G) is Markovian then*

$$f_n(s,x) = \rho(n,s,x,f_{n+1}(T(n,s,x))) \tag{4.76}$$

for all $1 \leq n < N, s \in S_n, x \in D(n,s)$.

Recall that, as already established in *Corollary 4.3.1*, given the *Principle of Conditional Optimization*, any decomposition scheme must satisfy

$$f_n(s) = \operatorname*{opt}_{x \in D(n,s)} f_n(s,x) \ , \ \forall 1 \leq n < N, s \in S_n \tag{4.77}$$

So, this result together with (4.76) furnish the grounds for concluding the following:

Theorem 4.4.1 *Optimality Equation:*
Assume that the decomposition scheme is Markovian. Then,

$$f_n(s) = \underset{x\in D(n,s)}{\text{opt}} \ \rho(n,s,x,f_{n+1}(T(n,s,x))) \ , \ \forall 1 \leq n < N, s \in S_n \tag{4.78}$$

With the optimality equation in hand, the following remarks are in order:

1. The reader is reminded that the optimality equation does not amount to a definition of $\{f_n(s)\}$. Indeed, $f_n(s)$ is defined by (4.45), namely

$$f_n(s) := \underset{z\in X(n,s)}{\text{opt}} \ g_n(s,z) \ , \ n \in \mathbb{N}, s \in S_n \tag{4.79}$$

   Rather, the optimality equation should be understood as a statement about the relation between the sets $\{f_n(s) : s \in S_n\}$ and $\{f_{n+1}(s) : s \in S_{n+1}\}$.
2. If $N = \infty$, then (4.78) is stated thus

$$f_n(s) = \underset{x\in D(n,s)}{\text{opt}} \ \rho(n,s,x,f_{n+1}(T(n,s,x))) \ , \ \forall n \in \mathbb{N}, s \in S_n \tag{4.80}$$

   whereas if N is finite then (4.78) holds for all $1 \leq n < N$, and for $n = N$ we have, by definition,

$$f_N(s) := \underset{x\in D(N,s)}{\text{opt}} \ g_N(s,x), s \in S_N \tag{4.81}$$

   which means that *Problem P(N,s)* is an "easy" problem, comprising as it does a single decision variable.
3. Because the objects $\{f_n\}$ defined by (4.45) can be treated as real-valued functions defined on the sets $\{S_n\}$ respectively, it is customary to call the equality equation FUNCTIONAL EQUATION. In this book the terms optimality equation and functional equation will be used interchangeably.
4. An explanation of the rationale behind my use of the symbol f_n is required.

   This symbol is employed throughout the book to designate two manifesly distinct objects: the optimal value yielded by the objective function of *Problem* $P(n,s)$ which is denoted by $f_n(s)$, and the optimal value yielded by the objective function of *Problem* $P(n,s,x)$ which is denoted by $f_n(s,x)$.
   This treatment of the symbol f_n is in concert with my approach to the notation which has already been discussed to some extent in *Chapter 3.* You will recall that I noted in *Section 3.5* that the system of notation required by a formal exposition of dynamic programming calls for a delicate balance between two seemingly countervailing requirements — the need to formulate a minutely detailed notation that gives full and accurate expression to objects and the relations between them; and the

need to maintain a high degree of clarity of presentation which calls for an uncluttered notation. In a word, one is required to frame a notation that is descriptive, yet concise.

My position on this problem is that an economical yet descriptive notation can be worked out by systematically capitalizing on the basic links that occur between primitive concepts, objects and relationships. Thus, exploiting the kinship between the set of modified problems and that of conditional problems, f_n is assigned to two apparently disparate yet intimately related objects.

It ought to be noted, though, that because f_n is being treated in the functional equation of dynamic programming as an (unknown) function on S_n, the use of $f_n(s, x)$ for the purpose described above may perhaps be questioned. To anticipate objection note that, f_n can be defined formally as a real valued function on $S_n \cup (S_n \times \mathbb{D})$. In this guise, the optimality equation is concerned only with the values that f_n attains on S_n.

A rationale similar to the one described above is behind the use of other symbols and conventions. For example, the symbol X^* is used for three different purposes. In addition to its independent standing, it also forms part of two composite symbols designating two distinct, yet kindred objects, namely the sets $X^*(n, s)$ and $X^*(n, s, x)$.

4.5 Dynamic Programming Problems

Having outlined the derivation of a dynamic programming functional equation, I shall now proceed to frame a *formal* definition of a dynamic programming problem. You will recall that our point of departure (*Section 2.1*) was the premiss that any optimization problem that can be boiled down to an optimality equation is regarded a dynamic programming problem. We are now in a position to tighten this characterization of a dynamic programming problem into a *rigorous* definition, so as to obtain a *binding* criterion on which a problem will be judged. It seems rather obvious that the basic ingredients of this definition should be found in the derivation process outlined in the preceding sections. So, keeping this process in mind, let us begin with a formal definition of a dynamic programming *model.*

Definition 4.5.1 *Dynamic Programming Model:*
A DYNAMIC PROGRAMMING MODEL *is any collection* $(N, S, D, T, S_1, g, \rho, G)$ *such that:*

· (N, S, D, T, S_1, g) *is a multistage decision model.*
· (ρ, G) *is a decomposition scheme for this model.*
· *The optimality equation (4.78) holds.*

In view of this we can argue as follows:

Definition 4.5.2 *Dynamic Programming Problem:*
An instance of Problem P is said to be a DYNAMIC PROGRAMMING PROBLEM *if there exist*

(1) a dynamic programming model; and

(2) an initial state $s \in S_1$ such that Problem $P(s)$ is equivalent to Problem $P(1,s)$, that is, $X(1,s) = X(s)$ and $g(s,z) = q(z), \forall z \in X(s)$, where, as you recall, X denotes the set of feasible solutions to Problem P and $X(1,s)$ denotes the set of feasible solutions to Problem $P(1,s)$. Observe that these conditions entail that Problem P, Problem $P(s)$ and Problem $P(1,s)$ all have the same set of optimal solutions, that is, $X^*(1,s) = X^*(s) = X^*$.

Ignoring the fine technical details, what this definition asserts is that a dynamic programming problem is any instance of *Problem P* that can be formulated as a multistage decision problem in a manner that:

· Preserves the original structure of the objective function and feasible solutions; and

· The multistage decision problem possesses a decomposition scheme yielding the optimality equation specified by (4.78).

That said, the question naturally arises whether one would be able to use this definition as a means to establish whether a *given* instance of *Problem P* is a dynamic programming problem. Recall that *Problem P* has the following format:

$$\textit{Problem P:}\ \ p := \underset{z \in X}{\text{opt}}\ q(z)\ ,\ \ X \subset X' := X_1 \times X_2 \times \cdots \times X_M \tag{4.82}$$

To illustrate this point, let us examine it in the context of a specific problem.

4.5.1 Example

Consider the following instance of *Problem P,* where $\{A_n\}$ and $\{B_n\}$ are real-valued functions on the interval $[0,r]$, $r > 0$:

$$p := \underset{(x_1,\dots,x_N)}{\text{opt}} \frac{\displaystyle\sum_{n=1}^{N} A_n(x_n)}{\displaystyle\sum_{n=1}^{N} B_n(x_n)} \tag{4.83}$$

$$\sum_{n=1}^{N} x_n \le r\ ,\ \ x_n \ge 0, n = 1,2,\dots,N \tag{4.84}$$

Would an appeal to *Definition 4.5.2* enable determining *straightaway* whether this is a dynamic programming problem? □

Clearly, what I am driving at is that we would need a far more *instructive* means to be able to determine directly whether a specific problem is a dynamic programming problem. But if this is so, what is the place and role of *Definition 4.5.2* in our discussion?

Definition 4.5.2 is a rigorous statement of the essential properties of a dynamic programming problem. It thus gives a precise formulation to the requirements that a problem must meet to count as a dynamic programming problem, so in this sense it is a *criterion.* However, precisely for this reason it is a criterion that applies a posteriori. That is, one would have to put a problem to the test of this criterion to establish whether the problem complies with it. As we shall shortly see though, it is possible to formulate a more direct criterion.

Be that as it may, the thing to note about *Definition 4.5.2* is that it highlights two points. First, in addition to requiring that an instance of *Problem P* be amenable to a multistage decision problem formulation, it requires that the objective function be *separable* and the *optimality equation* be valid.

Second, it remains totally unconcerned about whether the optimality equation is *solvable.* It thus gives a formal stamp to my thesis that dynamic programming's primary concern is to bring an optimization problem to the form of a functional equation. The solution of the equation is a separate matter that is not necessarily the "responsibility" of dynamic programming. Dynamic programming neither determines whether an equation is tractable nor does it necessarily furnish the means for the solution.

This is so because, as we have seen in *Section 2.1,* in each case the solution of the functional equation hangs entirely on the properties that the objects figuring in the equation would happen to have.

I should add here that were the equation's tractability the criterion deciding what is a dynamic programming problem, agreement on this question would be impossible. Considerations such as the problem's size and the capabilities of the computer used to solve a given optimality equation would prove the deciding factors. Thus, the same problem would be deemed a dynamic programming problem by someone using a super computer, but a non-dynamic programming problem by someone using a personal computer. By the same token, a problem deemed a non-dynamic dynamic programming problem in 2010 would miraculously turn into a dynamic programming problem in 2020, and so on.

Of course, this is not peculiar to dynamic programming, it is true of many other optimization methods (e.g. *Linear Programming*).

Let us then consider the following:

Theorem 4.5.1 *Any regular instance of Problem P is a dynamic programming problem.*

At this stage it is hardly surprising that I should propose this thesis as a criterion for determining what is a dynamic programming problem. After all, I have been arguing it, in one way or another, from the start.

All that is left then is to demonstrate its validity by means of a formal argument. To do this, it suffices to show that there exists a Markovian decomposition scheme associated with the trivial multistage decision model. This is so in view of *Theorem 4.4.1* and given that *Lemma 3.3.2* provides that any regular instance of *Problem P* can be stated in terms of the trivial multistage decision model described in *Definition 3.3.1.* Consider then the following.

Lemma 4.5.1 *The trivial multistage decision model specified in Definition 3.3.1 has a Markovian decomposition scheme.*

Proof. Recall that the objects comprising the trivial multistage decision model are defined as follows:

$$N^\circ = M \ , \ \mathbb{N}^\circ := \{1,2,3,\dots,N^\circ\} \tag{4.85}$$

$$S^\circ : = \bigcup_{n=1}^{N^\circ} S_n^\circ \ , \ S_n^\circ := \{(n-1)\uparrow y : y \in X\} \tag{4.86}$$

$$D^\circ(n,s) : = \begin{cases} \varnothing & , \ s \notin S_n^\circ \\ \{x \in X_n : (s,x) \in S_n^\circ\} & , \ s \in S_n^\circ \end{cases} \tag{4.87}$$

$$T^\circ(n,s,x) : = (s,x) \ , \ n \in \mathbb{N}^\circ, s \in S_n^\circ, x \in D^\circ(n,s) \tag{4.88}$$

$$g^\circ(s,z) : = q(z) \ , \ s \in S_1^\circ, z \in X \tag{4.89}$$

Now, since it follows from (4.86) that

$$S_{n+1}^\circ = \{T^\circ(n,s,x) : s \in S_n^\circ, x \in D^\circ(n,s)\} \ , \ \forall n \in \mathbb{N}^\circ \tag{4.90}$$

these sets are consistent with the definition of the sets $\{S_n\}$ in (4.40), meaning that they can be state spaces in the multistage decision model.

We now have to construct a decomposition scheme for the objective function defined by (4.89) and we need to show that this scheme is Markovian. Consider then the scheme $(\rho^\circ, G^\circ = \{g_n^\circ\})$ where

$$g_n^\circ(s,z) := g^\circ(s,z) \ , \ n \in N^\circ, s \in S_n^\circ, z \in X(n,s) \tag{4.91}$$

$$\rho^\circ(n,s,x,a) := a \ , \ n \in N^\circ, s \in S_n^\circ, x \in D^\circ(n,s), a \in \mathbb{R} \tag{4.92}$$

bearing in mind that $X(n,s)$ denotes the set of feasible solutions to *Problem* $P(n,s)$.

Note that (4.91) entails that $g_n^\circ(s,z) = g^\circ(s,z)$, $\forall s \in S_1^\circ$, $z \in X(1,s)$ and furthermore that

$$\begin{aligned} g_n^\circ(s,x,z) &= g_{n+1}^\circ(T^\circ(n,s,x),z)) & (4.93) \\ &= \rho^\circ(n,s,x,g_{n+1}^\circ(T^\circ(n,s,x))) & (4.94) \end{aligned}$$

for all $1 \leq n < N^\circ$, $s \in S_n^\circ$, $x \in D^\circ(n,s)$ and $z \in X(n+1, T(n,s,x))$.

This means that the objective function g° is separable, thus showing (ρ°, G°) to be a decomposition scheme for the trivial multistage decision model. Our next task is to show that this decomposition scheme is Markovian.

So, let (n,s,x) be any triplet such that $1 \leq n < N^\circ$, $s \in S_n^\circ$, and $x \in D^\circ(n,s)$. The object is to show that *Problem* $P(n,s,x)$ and *Problem* $P(n+1, T^\circ(n,s,x))$ have the same set of optimal solutions. Observe then that

$$\textit{Problem } P(n,s,x): \qquad f_n(s,x) := \operatorname*{opt}_{\substack{z \in Z(n+1,s') \\ s'=T^\circ(n,s,x)}} g_n^\circ(s,x,z) \tag{4.95}$$

$$\textit{Problem } P(n+1,s'): \quad f_{n+1}(s') := \operatorname*{opt}_{\substack{z \in X(n+1,s') \\ s'=T^\circ(n,s,x)}} g_{n+1}^\circ(s',z) \tag{4.96}$$

Finally, note that by construction $s' = T^\circ(n,s,x) = (s,x)$ and that from (4.93) it follows that g_{n+1}° and g_n° are essentially identical. Therefore, (4.95)-(4.96) in fact imply that the two problems are identical. Hence,

$$X^*(n,s,x) = X^*(n+1, T^\circ(n,s,x)) \tag{4.97}$$

the inference thus being that the decomposition scheme is MARKOVIAN. □

Setting the mathematics aside for a moment, let us attempt to elucidate the contention made by *Theorem 4.5.1* and *Lemma 4.5.1.* What these results assert is that there is a universal multistage decision model that is amenable to decomposition by a universal Markovian decomposition scheme — universal in that they apply to any regular instance of *Problem P.* The end product of the decomposition is a dynamic programming model whose defining property is that any one of its modified problems is identical to the conditional problem inducing it.

It should be pointed out in this connection that because the trivial multistage decision model is characterized by states of the form

$$s_n = (x_1, x_2, \ldots, x_{n-1}) \ , \ 1 < n \leq N \tag{4.98}$$

in cases where N is finite we have

$$S_{N+1}^\circ = X \tag{4.99}$$

Therefore, the objective function q of *Problem P* and the objective function g° of *Problem P(s)*, as well as the modified objective functions $\{g_n^\circ\}$, can all be viewed as real valued functions on the *final state space* S_{N+1}°.

But even if N is not finite, the modified and conditional problems are of the following form:

$$f_n^\circ(s) := \operatorname*{opt}_{z \in X(n,s)} q(n,s) \ , \ n \in \mathbb{N}^\circ, s \in S_n^\circ \tag{4.100}$$

$$f_n^\circ(n,s,x) := \operatorname*{opt}_{\substack{z \in X(n+1,s') \\ s'=T^\circ(n,s,x)}} q(s,x,z) \tag{4.101}$$

for $1 \le n < N, s \in S_n^\circ, x \in D^\circ(n,s)$ respectively, where q is the objective function of *Problem P.*

Bearing in mind then that $T^\circ(n,s,x) = (s,x)$, it follows immediately that *Problem* $P(n,s,x)$ is identical to *Problem* $P(n+1, T^\circ(n,s,x))$. Consequently, the resulting composition function ρ° has the degenerate form $\rho^\circ(n,s,x,a) = a$, which in turn yields the following dynamic programming functional equation:

$$f_n^\circ(s) = \underset{x \in D^\circ(n,s)}{\text{opt}} \; f_{n+1}^\circ(T^\circ(n,s,x)) \ , \ 1 \le n < N^\circ, s \in S_n^\circ \tag{4.102}$$

If N° is finite, then by definition

$$f_{N^\circ}^\circ(s) := \underset{x \in D^\circ(N^\circ,s)}{\text{opt}} \; g_{N^\circ}^\circ(s,x) \ , \ s \in S_{N^\circ}^\circ \tag{4.103}$$

$$= \underset{x \in D^\circ(N^\circ,s)}{\text{opt}} \; q(s,x) \tag{4.104}$$

Of course, the dynamic programming functional equation stemming from the trivial multistage decision model and its trivial decomposition scheme is of little interest as it turns out to be identical to the equation obtained from a successive application of the *Principle of Conditional Optimization* to *Problem P.* In other words, since by construction S_n° is the projection of X on $X_1 \times X_2 \times \cdots \times X_{n-1}$, the solution of this equation would require the enumeration of all the feasible solutions to *Problem P.* In short, the functional equation deriving from the trivial model can be expressed in terms of the decision varaibles as follows:

$$f_n(x_1, \ldots, x_{n-1}) = \underset{x_n \in D^\circ(n,x_1,x_2,\ldots,x_{n-1})}{\text{opt}} \; f_{n+1}(x_1, \ldots, x_{n-1}, x_n) \tag{4.105}$$

All this adds up to the following:

Should the trivial model turn out to be the only multistage decision model that would be available for modeling the problem concerned, dynamic programming would revert to DIRECT ENUMERATION . This, however, in no way detracts from the significance of *Theorem 4.5.1.* If anything, it accentuates the theorem's fundamental import. Because, if to substantiate its validity *Theorem 4.5.1* had to be anchored in a model that gives rise to direct enumeration, the inference clearly is that this theorem touches at the very roots of dynamic programming.

4.6 The Final State Model

It appears that the trivial multistage decision model is the only conceivable model that is capable of supplying the requisite mathematical framework for demonstrating the validity of *Theorem 4.5.1.* The question obviously arising is: which of this model's properties makes this possible? And the answer to this is that the property that is instrumental in making this model so appropriate a medium for proving *Theorem 4.5.1* is its *final-state property.*

To explain this point I need to introduce into the discussion the notion "final state". The concept final-state is self-explanatory when it is used in the context of N being finite. In this case, this term designates the terminal state that emanates from the initial state s_1 and the entire sequence of decisions $(x_1, x_2, \dots, x_N)$ in accordance with the precepts of the transition function T. For example, if the transition function is of the form

$$T(n, s, x) = s - x \tag{4.106}$$

then the final state resulting from s_1 and $(x_1, x_2, \dots, x_N)$ is determined as follows:

$$s_2 = T(1, s_1, x_1) = s_1 - x_1 \tag{4.107}$$

$$s_3 = T(2, s_2, x_2) = s_2 - x_2 = (s_1 - x_1) - x_2 = s_1 - (x_1 + x_2) \tag{4.108}$$

$$\begin{aligned} s_4 &= T(3, s_3, x_3) = s_3 - x_3 \\ &= (s_2 - (x_1 + x_2)) - x_3 = s_1 - (x_1 + x_2 + x_3) \end{aligned} \tag{4.109}$$

and by induction

$$s_{n+1} = s_1 - \sum_{m=1}^{n} x_m \ , \ 1 \leq n \tag{4.110}$$

Hence,

$$s_{N+1} = s_1 - \sum_{n=1}^{N} x_n \tag{4.111}$$

In short, if N is finite, the final state is the state yielded by a successive application of the transition function T to the pairs $\{(s_n, x_n) : n \in \mathbb{N}\}$, the state in question being the consequence of $T(N, s_N, x_N)$, where s_N is the state generated by s_1 and $(x_1, \dots, x_{N-1})$.

If N is not finite, the final state of the process would be the limit of the sequence of states $(s_n : 1 \leq n < m)$ where m goes to infinity. The underlying supposition naturally is that such a limit exists. Thus, if it is supposed in the above case that N is not finite, the final state would be

$$s_\infty := \lim_{m \to \infty} s_m \tag{4.112}$$

$$= \lim_{m \to \infty} \left\{ s_1 - \sum_{n=1}^{m} x_n \right\} \tag{4.113}$$

$$= s_1 - \sum_{n=1}^{\infty} x_n \tag{4.114}$$

the assumption being that the sequence $(s_n : n \geq 1)$ has a limit. This assumption is valid in many dynamic programming applications, but there are cases where it does not hold. Methodologically, however, it is perfectly legitimate to ignore the technical questions that need to be reckoned with in such cases as our aim is to focus on the *"final state model"* and its role in place in dynamic programming.

Definition 4.6.1 *Final State Model:*
A multistage decision model (N, S, D, T, S_1, g) is said to be a FINAL STATE MODEL *if it possesses a decomposition scheme $(\rho,\ G = \{g_n\})$ such that the composition function ρ has this form:*

$$\rho(n, s, x, a) = a \ , \ \forall n \in \mathbb{N}, s \in S_n, x \in D(n, s), a \in \mathbb{R} \tag{4.115}$$

Recall that by definition, any decomposition scheme must satisfy the condition

$$g_n(s, x, z) = \rho(n, s, x, g_{n+1}(T(n, s, x), z)) \tag{4.116}$$

for all $1 \leq n < N,\ s \in S_n,\ x \in D(n, s)$ and $z \in X(n+1, T(n, s, x))$, where $X(n+1, s')$ denotes the set of feasible solutions to *Problem* $P(n+1, s')$. It follows therefore that it is characteristic of any decomposition scheme compliant with (4.115) to have the following property

$$\begin{aligned} g_n(s_n, x_n, x_{n+1}, \ldots, x_N) &= g_{n+1}(s_{n+1}, x_{n+1}, x_{n+2}, \ldots, x_N) && (4.117) \\ &= g_{n+2}(s_{n+2}, x_{n+2}, x_{n+3}, \ldots, x_N) && (4.118) \\ &= g_{n+3}(s_{n+3}, x_{n+3}, x_{n+4}, \ldots, x_N) && (4.119) \end{aligned}$$

and by induction

$$g_n(s_n, x_n, x_{n+1}, \ldots, x_N) = g_m(s_m, x_m, x_{m+1}, \ldots, x_N) \ , \ \forall m > n \tag{4.120}$$

where $s_{n+1} = T(n, s_n, x_n)$, $s_{n+2} = T(n+1, s_{n+1}, x_{n+1})$, etc.

In other words, the impact that the sequence $(s_n, x_n, x_{n+1}, \ldots, x_{m-1})$ has on g_n reduces to the effect of the state s_m on this function. Of course if N is finite then g_n is affected only by (s_N, x_N).

Now, it is abundantly clear that the trivial multistage decision model complies with the requirement stipulated by (4.115). Hence,

Corollary 4.6.1 *Any regular instance of Problem P can be formulated as a final state multistage decision problem.*

And, as it is equally clear that any decomposition scheme satisfying (4.115) is trivially Markovian, then,

Corollary 4.6.2 *Any final state multistage decision model has a Markovian decomposition scheme.*

4.6.1 Example

Consider again the problem featured in *Example 4.5.1,* that is,

$$p := \underset{(x_1,\dots,x_N)}{\text{opt}} \frac{\sum_{n=1}^{N} A_n(x_n)}{\sum_{n=1}^{N} B_n(x_n)} \tag{4.121}$$

$$\sum_{n=1}^{N} x_n \leq r \ , \ r \geq 0 \tag{4.122}$$

$$x_n \geq 0 \ , \ \forall n \geq 1 \tag{4.123}$$

Case 1. $N < \infty$. Let the state variables be triplets of the form $s_n = (a_n, b_n, r_n)$, where

$$a_1 = b_1 = 0 \tag{4.124}$$

$$r_1 = r \tag{4.125}$$

$$a_n := \sum_{m=1}^{n-1} A_m(x_m) \ , \ 1 < n \leq N+1 \tag{4.126}$$

$$b_n := \sum_{m=1}^{n-1} B_m(x_m) \ , \ 1 < n \leq N+1 \tag{4.127}$$

$$r_n := r - \sum_{m=1}^{n-1} x_m \ , \ 1 < n \leq N+1 \tag{4.128}$$

Then, by construction,

$$a_{n+1} = a_n + A_n(x_n) \tag{4.129}$$

$$b_{n+1} = b_n + B_n(x_n) \tag{4.130}$$

$$r_{n+1} = r_n - x_n \tag{4.131}$$

Thus, we can set

$$T(n, s_n, x_n) = (a_n + A_n(x_n), b_n + B_n(x_n), r_n - x_n) \tag{4.132}$$

where $s_n = (a_n, b_n, r_n)$.

Next, to ensure that the constraints are met, set

$$D(n, s_n, x_n) = [0, r_n] \ , \ s_n = (a_n, b_n, r_n) \tag{4.133}$$

Since the initial state under this formulation is the triplet $s_1 = (0, 0, r)$, set $S_1 = \{(0, 0, r)\}$ and let the state space be $S = \mathbb{R}^3$. Finally, define the objective function as follows:

$$g(s_1, x_1, x_2, \ldots, x_N) = \frac{\displaystyle\sum_{n=1}^{N} A_n(x_n)}{\displaystyle\sum_{n=1}^{N} B_n(x_n)} \tag{4.134}$$

Now, consider the following modified objective functions:

$$g_n(s_n, x_n, x_{n+1}, \ldots, x_N) = \frac{a_n + \displaystyle\sum_{m=n}^{N} A_m(x_m)}{b_n + \displaystyle\sum_{m=n}^{N} B_m(x_m)} \tag{4.135}$$

where $s_n = (a_n, b_n, r_n)$. Then by construction,

$$g_n(s_n, x_n, x_{n+1}, \ldots, x_N) = \frac{a_n + A_n(x_n) + \displaystyle\sum_{m=n+1}^{N} A_m(x_m)}{b_n + B_n(x_n) + \displaystyle\sum_{m=n+1}^{N} B_m(x_m)} \tag{4.136}$$

hence

$$\begin{aligned} g_n(s_n, x_n, x_{n+1}, \ldots, x_N) &= g_{n+1}(T(n, s_n, x_n), x_{n+1}, x_{n+2}, \ldots, x_N) && (4.137)\\ &= \rho(n, s_n, x_n, g_{n+1}(s_{n+1}, x_{n+1}, \ldots, x_N)) && (4.138) \end{aligned}$$

where $s_{n+1} = T(n, s_n, x_n)$ and ρ is the degenerate composition function specified by (4.115).

To establish that this is a proper decomposition scheme, we need to show that

$$g_1(s_1, x_1, x_2, \ldots, x_N) = g(s_1, x_1, x_2, \ldots, x_N) \tag{4.139}$$

Hence, since $s_1 = (0, 0, r)$, the definition of g_1 yields

$$\begin{aligned} g_1(s_1, x_1, x_2, \ldots, x_N) &= \frac{0 + \displaystyle\sum_{n=1}^{N} A_n(x_n)}{0 + \displaystyle\sum_{n=1}^{N} B_n(x_n)} \quad , \; s_1 = (0, 0, r) && (4.140)\\ &= \frac{\displaystyle\sum_{n=1}^{N} A_n(x_n)}{\displaystyle\sum_{n=1}^{N} B_n(x_n)} && (4.141)\\ &= g(s_1, x_1, x_2, \ldots, x_N) && (4.142) \end{aligned}$$

The multistage decision model formulated above is then a final state model. The final state resulting from $(s_n, x_n, x_{n+1}, \dots, x_N)$ is the following:

$$s_{N+1} = \left(a_n + \sum_{m=n}^{N} A_m(x_m), b_n + \sum_{m=n}^{N} B_m(x_m), r_n - \sum_{m=n}^{N} x_m\right) \tag{4.143}$$

recalling that $s_n = (a_n, b_n, r_n)$.

Case 2. $N = \infty$.

To express explicitly that N is not finite, the problem under consideration would be rephrased as follows:

$$p := \operatorname*{opt}_{(x_1,\dots,x_N)} \frac{\sum_{n=1}^{\infty} A_n(x_n)}{\sum_{n=1}^{\infty} B_n(x_n)} \tag{4.144}$$

$$\sum_{n=1}^{\infty} x_n \leq r \ , \ r \geq 0 \tag{4.145}$$

$$x_n \geq 0 \ , \ n = 1, 2, 3, 4, \dots \tag{4.146}$$

The model's formulation remains unchanged despite N not being finite in this case, which means that we still have

$$s_n := \left(\sum_{m=1}^{n-1} A_m(x_m), \sum_{m=1}^{n-1} B_m(x_m), r - \sum_{m=1}^{n-1} x_m\right) \tag{4.147}$$

Therefore, the final state is defined thus

$$s_\infty : = \lim_{n\to\infty} s_n \tag{4.148}$$

$$= \left(\sum_{n=1}^{\infty} A_n(x_n) \ , \ \sum_{n=1}^{\infty} B_n(x_n) \ , \ r - \sum_{n=1}^{\infty} x_n\right) \tag{4.149}$$

Should it turn out that no such limits exist, then the inference to be drawn is that the original problem to be precise, the objective function g was ill defined. $\square$

In summary then, it follows from (4.115-4.116) that the functional equation deriving from a final state model is of the form

$$f_n(s) = \operatorname*{opt}_{x\in D(n,s)} f_{n+1}(T(n,s,x)) \ , \ 1 \leq n < N, s \in S_n \tag{4.150}$$

and if N is finite, then by definition

$$f_N(s) := \operatorname*{opt}_{x\in D(N,s)} g_N(s,x) \ , \ s \in S_N \tag{4.151}$$

The final state model has a special place in dynamic programming. This is due not only to the fact that many problems naturally lend themselves to a final state formulation, but primarily to the model's important methodological function in the theory. Its stark simplicity, rudimentary structure and hence, universal applicability, make it the ideal framework for setting out the basic ideas of dynamic programming. This — as I shall argue in *Chapter 13* — seems to explain Bellman's choice of the final state model as the medium for formulating the essential ingredients of dynamic programming, notably the *Principle of Optimality.*

4.7 Principle of Optimality

In this concluding section, I very briefly discuss Bellman's *Principle of Optimality.* My main objective is to show that the principle is, in fact, equivalent to the *Markovian Condition.* I shall thereby square my explanation of the derivation of the functional equation with that of Bellman's explanation. At this stage I merely give a rough sketch of this point. In *Chapter 13,* where I conduct an in-depth investigation of the principle, I give a fuller account of this matter.

Let us go back then to the analysis in *Section 4.3.* The optimality equation, as you will recall, emerged from this analysis as a direct result of the *Markovian Condition*

$$X^*(n,s,x) = X^*(n+1,T(n,s,x)) \tag{4.152}$$

for all $1 \leq n < N, s \in S_n, x \in D(n,s)$, where $X^*(n,s,x)$ and $X^*(n+1,s')$ denote the sets of optimal solutions for *Problem* $P(n,s,x)$ and *Problem* $P(n+1,s')$ respectively. To give (4.152) a verbal phrasing it will read as follows:

> *z is an optimal solution for Problem $P(n,s,x)$ if, and only if, it is an optimal solution for the modified problem induced by (n,s,x), namely Problem $P(n+1,T(n,s,x))$.*

Now, suppose that in lieu of (4.152), we require the following:

$$X^*(n,s,x) \subseteq X^*(n+1,T(n,s,x)) \tag{4.153}$$

for all $1 \leq n < N, s \in S_n, x \in D(n,s)$. Likewise, phrasing the latter verbally, it will read as follows:

> *If z is an optimal solution for Problem $P(n,s,x)$, it must also be an optimal solution to the modified problem induced by (n,s,x), namely Problem $P(n+1,T(n,s,x))$.*

I shall now show that (4.153) implies (4.152). This will clearly imply that any decomposition scheme satisfying the former can be counted on to produce a valid dynamic programming functional equation. Consider then the following. Given the decomposition scheme (ρ, G), then by definition of $X^*(n, s, x)$, it follows that

$$g_n(s, x, z) = f_n(s, x) \ , \ \forall z \in X^*(n, s, x) \tag{4.154}$$

and

$$g_n(s, x, z) = \rho(n, s, x, g_{n+1}(T(n, s, x), z)) \ , \ \forall z \in X^*(n, s, x) \tag{4.155}$$

for any triplet (n, s, x) such that $1 \leq n < N$, $s \in S_n$ and $x \in D(n, s, x)$ and this regardless of whether or not (4.153) is satisfied.

In other words, what we have seen is that **any** decomposition scheme typically has this property: $\forall 1 \leq n < N, s \in S_n, x \in D(n, s, x), z \in X^*(n, s, x)$

$$f_n(s, x) = \rho(n, s, x, g_{n+1}(T(n, s, x), z)) \tag{4.156}$$

Now, let (n, s, x, z) be any quadruplet satisfying the requirement given by (4.46) and assume that (4.153) holds. This implies that $z \in X^*(n + 1, s')$ where $s' = T(n, s, x)$. Therefore, since by definition $z \in X^*(n+1, s')$ implies that

$$g_{n+1}(s', z) = f_{n+1}(s') \tag{4.157}$$

it necessarily follows that

$$\begin{align} f_n(s, x) &= \rho(n, s, x, f_{n+1}(s')) \tag{4.158} \\ &= \rho(n, s, x, g_{n+1}(s', z)) \ , \ \forall z \in X^*(n + 1, s') \tag{4.159} \\ &= g_n(s, x, z) \ , \ \forall z \in X^*(n + 1, s') \tag{4.160} \end{align}$$

This, of course, entails that $X^*(n + 1, T(n, s, x)) \subseteq X^*(n, s, x)$. Thus, the following result holds.

Corollary 4.7.1 *If (4.153) holds, then (4.152) must hold as well. That is, (4.153) implies (4.152). Furthermore, since (4.152) entails (4.153), it follows that (4.152) and (4.153) are in fact, equivalent.*

The inference is then that postulating the *Markovian Condition* specifies by (4.152) or the condition specified by (4.153) amounts to the same thing; and what this translates to is that the optimality equation can be treated either as an immediate consequence of (4.153) or as an immediate consequence of (4.152).

Now, this understanding of the optimality equation where it is taken to be a direct result of (4.153) constitutes the very core of Bellman's approach to dynamic programming. Indeed, so central is it to his perception of dynamic programming, that (4.153) is accorded the status of a *principle* in Bellman's formulation of dynamic programming. In his unique style, Bellman [1957a, *p.* 83] phrased this fundamental relation as follows:

> PRINCIPLE OF OPTIMALITY. *An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.*

Read in terms of the terminology introduced in this chapter, the principle asserts the following:

> *Whatever the stage n, state s, and decision x are, if z is an optimal solution for Problem $P(n,s,x)$, then z must also be an optimal solution for Problem $P(n+1,T(n,s,x))$.*

It is abundantly clear from this phrasing that the principle is fully equivalent to (4.153).

As a final note I should add that right from its introduction the *Principle of Optimality* has attracted much comment. Its wording has been claimed to be too loose, its validity has been called into question and reservations have been expressed as to the very legitimacy of regarding it a *principle.* I shall take up these issues in *Chapter 13.* For now, it suffices to say that these critical comments have by and large been due to a lack of appreciation that in his first book Bellman introduced the principle in the context of a *final state model.* To this I might add that it has also not been sufficiently appreciated that no justice can be done to the principle unless it is examined from the viewpoint of its relation to the functional equation of dynamic programming.

4.8 Summary

We saw that the two central mathematical constructs in the derivation of the dynamic programming functional equation are:

· The multistage decision model.
· The decomposition scheme.

These, as I demonstrated, exist as a matter of principle. On these grounds I showed that, subject to the technical regularity condition articulated by *Assumption 3.3.1,* any instance of *Problem P* is a dynamic programming problem, irrespective of the particular structures of the objective function q and the decision set X. I then went on to argue that the only general model apparently capable of upholding the validity of this proposition is the trivial multistage decision model, thereby showing it to be the most rudimentary and universally applicable dynamic programming model.

Following that I introduced the "final state model" as the framework within which the trivial composition function preserves its structure in the sense that all its modified objective functions are identical. I then showed that final state models are Markovian and that any regular instance of *Problem P* can be stated as a final state problem.

Finally, having shown that the optimality equation is a direct implication of either the *Markovian Condition* or the *Principle of Optimality* — which as we have seen, are different expressions of one and the same relation — I demonstrated that the dynamic programming optimality equation rests on a robust foundation.

This established, I shall now have to determine what types of dynamic programming problems are tractable. To do this, I shall investigate the techniques that are available for the solution of dynamic programming functional equations and the computational requirements of dynamic programming algorithms.

Our topic in the next two chapters will be then the solution of the dynamic programming functional equation.

5

Solution Methods

5.1 Introduction

In this chapter I describe, in broad terms, the main methods for solving dynamic programming functional equations. The focus is on the rationale behind these methods and their mode of operation.

Over the years numerous solution techniques that capitalize on specific properties of the functional equation have also been developed. But these will not be discussed here either as my objective in this chapter is to profile the *major approaches* to solving the equation.

Picking up where we left off then, our starting point is the thesis that once a problem has been brought by the dynamic programming treatment to the form of a functional equation, the equation is solved by *whatever means are available.* It is handled not as a dynamic programming functional equation as such, but as an equation pure and simple. Commensurate with the equation's character, it is solved either by *standard* analytic methods, or by *standard* numeric methods, or even by *simple* enumeration techniques.

These methods can be classified, both conceptually and technically, in two groups:

· Direct methods
· Successive approximation methods

Direct methods are straightforward transliterations of the dynamic programming functional equation. That is, they literally activate the statement made by the functional equation, executing it, as it were, "verbatim" by means of an iterative procedure. In contrast, successive approximation methods put into action an iterative approximation scheme where, beginning with a rough approximation, the functional equation is solved iteratively in a manner that gradually updates — improves — the current approximation.

It should be noted that in both cases there are a number of variations on the main theme. Also, often the same functional equation can be solved by both direct and successive approximation methods so that there may be a choice in this matter.

But on the whole each approach is tailored to solve equations that are yielded by either one of these two types of models:

· *Finite horizon* models
· *Infinite horizon* models

Direct methods typically handle functional equations derived from finite horizon models, whereas successive approximation methods typically handle equations derived from infinite horizon models.

The distinction between these two classes of models is induced by a key feature of the multistage decision model — the value of N, namely the number of decision stages. Finite horizon models are characterized by a finite

N whereas infinite horizon models have infinitely many — but countable — decision stages. These models are also known as *truncated* and *non-truncated* models, respectively.

To demonstrate these two general approaches in action, I shall use a generic *additive* functional equation as the medium of discourse. Equations of this type are the most commonly encountered dynamic programming functional equations.

5.2 Additive Functional Equations

As we shall see in *Chapter 10,* dynamic programming provides five major composition functions. Out of these the most prevalent is clearly the following:

$$\rho(n,s,x,a) = w(n,s,x) + a \ , \ (n,s,x) \in NSD', a \in \mathbb{R} \tag{5.1}$$

where w is a real-valued function on

$$NSD := \{(n,s,x) : n \in \mathbb{N}, s \in S_n, x \in D(n,s,x)\} \tag{5.2}$$

and

$$NSD' := \{(n,s,x) : 1 \leq n < N, s \in S_n, x \in D(n,s)\} \tag{5.3}$$

For obvious reasons this composition function is called *additive*. Note that in this case we have

$$\begin{align} g_n(s_n, x_n, x_{n+1}, \dots, x_N) &= \rho(n, s_n, x_n, g_{n+1}(s_{n+1}, x_{n+1}, x_{n+2}, \dots, x_N)) \tag{5.4} \\ &= w(n, s_n, x_n) + g_{n+1}(s_{n+1}, x_{n+1}, \dots, x_N) \tag{5.5} \end{align}$$

where $s_{n+1} = T(n, s_n, x_n)$. Hence, *Problem* $P(n,s,x)$ takes the following form:

$$f_n(s,x) := \underset{\substack{z \in X(n+1,s') \\ s'=T(n,s,x)}}{\text{opt}} \{w(n,s,x) + g_{n+1}(s')\} \ , \ (n,s,x) \in NSD' \tag{5.6}$$

Since the second term on the right-hand side of this equation in fact amounts to *Problem* $P(n+1,s')$, plainly in this case *Problem* $P(n,s,x)$ and *Problem* $P(n+1,s')$ have the same set of optimal solutions, namely

$$X^*(n,s,x) = X^*(n+1, T(n,s,x)) \ , \ \forall (n,s,x) \in NSD' \tag{5.7}$$

The implication is then that any multistage decision model whose objective function has an additive composition function is Markovian. The dynamic programming functional equation induced by such a decomposition scheme is of the following form:

$$f_n(s) := \underset{x\in D(n,s)}{\text{opt}} \{w(n,s,x) + f_{n+1}(T(n,s,x))\} \ , \ 1 \leq n < N, s \in S_n \quad (5.8)$$

Functional equations of this kind are called *additive*. In the first part of the book our concern is exclusively with additive functional equations.

5.3 Truncated Functional Equations

Functional equations of this type are associated with multistage decision problems where N, the total number of decision stages, is finite. This gives rise to N unknown functions $\{f_n : 1 \leq n \leq N\}$ defined as follows:

$$f_n(s) := \underset{(x_n,\dots,x_N)}{\text{opt}} \ g_n(s, x_n, x_{n+1}, \dots, x_N) \ , \ n \in \mathbb{N}, s \in S_n \quad (5.9)$$

$$x_k \in D(k, s_k) \ , \ n \leq k \leq N \quad (5.10)$$

$$s_{k+1} = T(k, s_k, x_k) \ , \ n \leq k \leq N \quad (5.11)$$

$$s_1 = s \quad (5.12)$$

Thus, one would construct the decomposition scheme $(\rho, \{g_n : 1 \leq n \leq N\})$ to obtain a functional equation of this form:

$$f_n(s) = \underset{x\in D(n,s)}{\text{opt}} \ \rho(n, s, x, f_{n+1}(T(n,s,x))) \ , \ 1 \leq n < N, s \in S_n \quad (5.13)$$

Clearly, solving such an equation entails recovering the values of the unknown functions $\{f_n : 1 \leq n \leq N\}$ defined by (5.9)-(5.12).

So, the question is how would this task be approached when N is finite. Observe then that (5.1)-(5.5) implies that

$$f_N(s) := \underset{x\in D(N,s)}{\text{opt}} \ g_N(s,x) \ , \ s \in S_N \quad (5.14)$$

It is natural, therefore, to commence the solution process for the values of the unknown functions $\{f_n : 1 \leq n \leq N\}$ at $n = N$ and proceed through $n = N-1$, $N-2$, $N-3, \dots, 2$, 1 — *in this order*. In other words, the unknown functions $\{f_n\}$ would be determined *iteratively* by means of a simple procedure that can be described as follows:

5.3.1 Procedure

Step 1. Initialization.
Set $n = N$ and for each $s \in S_N$ determine the value of $f_N(s)$

by solving the problem specified by the right-hand side of (5.14), namely

$$f_N(s) \leftarrow \operatorname*{opt}_{x\in D(N,s)} g_N(s,x) \tag{5.15}$$

Step 2. Stopping rule.
If $n = 0$, stop. Otherwise, go to Step 3.

Step 3. Iteration.
Set $n = n - 1$ and for each $s \in S_n$ determine the value of $f_n(s)$ by solving the problem specified by the right-hand side of (5.13), namely

$$f_n(s) \leftarrow \operatorname*{opt}_{x\in D(n,s)} \rho(n,s,x,f_{n+1}(T(n,s,x))) \tag{5.16}$$

Step 4. Go to Step 2. □

Let us examine how this procedure would be used in a specific case.

5.3.2 Example: Knapsack Problem

Consider the following problem:

$$p := \max_{(x_1,\dots,x_N)} \sum_{n=1}^{N} w_n x_n \tag{5.17}$$

$$\sum_{n=1}^{N} v_n x_n \le v^* \tag{5.18}$$

$$x_n \in \{0,1,2,3,\dots\} \tag{5.19}$$

where the parameters $\{v_n\}$ and v^* are positive integers.

Optimization problems of this type are commonly known as *knapsack problems.* They are understood to capture the following situation.

Imagine N piles each containing infinitely many “identical” items: All the items in any pile n, have the same weight, w_n, and the same volume, v_n. The question is this: given a knapsack of volume v^*, what configuration of items selected from each pile would maximize the weight of the knapsack?

Cast as a multistage decision problem, the above problem would be formulated as follows:

$$\text{opt} = \max \tag{5.20}$$

$$S = \mathbb{D} = \{0,1,2,3,\dots,v^*\} \tag{5.21}$$

$$D(n,s) = \{0,1,2,\dots,\lfloor s/v_n\rfloor\} \ , \ s \in [0,v^*] \tag{5.22}$$

$$T(n,s,x) = s - v_n x \tag{5.23}$$

$$S_1 = \{v^*\} \tag{5.24}$$

$$g_n(s, x_n, x_{n+1}, \dots, x_N) = \sum_{m=n}^{N} w_m x_m \tag{5.25}$$

$$= \sum_{m=n}^{N} r(m, s_m, x_m) \ , \ r(m, s, x) := w_m x \tag{5.26}$$

where $\lfloor z \rfloor :=$ *Integer part of z.*

Thus, the functional equation would take the following form:

$$f_n(s) = \max_{x \in D(n,s)} \{r(n, s, x) + f_{n+1}(T(n, s, x))\} \ , \ 1 \leq n < N, s \in S_n \tag{5.27}$$

$$= \max_{x \in D(n,s)} \{w_n x + f_{n+1}(s - v_n x)\} \tag{5.28}$$

$$= \max \{w_n x + f_{n+1}(s - v_n x) : x \in \{0, 1, \dots, \lfloor s/v_n \rfloor\}\} \tag{5.29}$$

with

$$f_N(s) := \max_{x \in D(n,s)} r(N, s, x) \ , \ s \in S_N, x \in D(N, s) \tag{5.30}$$

$$= \max \{x w_N : x \in \{0, 1, 2, \dots, \lfloor s/v_N \rfloor\}\} \tag{5.31}$$

Because $a_N > 0$, it follows that

$$f_N(s) = w_N \lfloor s/v_N \rfloor \ , \ s \in S_N \tag{5.32}$$

Now, the iterative procedure requires the sets $\{S_n\}$. Recall then that by definition

$$S_{n+1} = \{T(n, s, x) : s \in S_n, x \in D(n, s)\} \ , \ 1 \leq n \leq N \tag{5.33}$$

$$= \left\{s - v_n x : s \in S_n, x \in \{0, 1, 2, \dots, \lfloor s/v_n \rfloor\}\right\} \tag{5.34}$$

with S_1 being known; in our case $S_1 = \{v^*\}$.

However, because the sets $\{S_n\}$ do not admit of a neat closed-form representation, it is sometimes convenient to solve the functional equation at each stage n for all $s \in S$ rather than only for $s \in S_n$, observing that $S_n \subseteq S$, for all n. Of course, this can be extremely wasteful in cases where v and v_n are very large. But, for our purposes, this point can be illustrated for a case where these values are very "small" and so, we set $S_n = S, \forall n > 1$.

Suppose then that the problem's parameters have the following values:

n	1	2	3	4	5	6
w_n	8	6	7	9	3	4
v_n	7	5	6	8	3	4

$N = 6; v^* = 27$

The direct iterative procedure would thus perform as follows:

Step 1: Initialization. Set $n = N = 6$. In view of (5.32), determining the

Table 5.1: Dynamic Programming Tableau for *Example 5.3.2*

s	$f_6(s)$	$f_5(s)$	$f_4(s)$	$f_3(s)$	$f_2(s)$	$f_1(s)$
0	0	0	0	0	0	0
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	3	3	3	3
4	4	3	4	4	4	4
5	4	4	4	4	6	6
6	4	6	6	7	7	7
7	4	7	7	7	7	8
8	8	8	9	9	9	9
9	8	9	9	10	10	10
10	8	10	10	11	12	12
11	8	11	12	12	13	13
12	12	12	13	14	14	14
13	12	13	13	14	15	15
14	12	14	15	16	16	16
15	12	15	16	17	18	18
16	16	16	18	18	19	19
17	16	17	18	19	20	20
18	16	18	19	21	21	21
19	16	19	21	21	22	22
20	20	20	22	23	24	24
21	20	21	22	24	25	25
22	20	22	24	25	26	26
23	20	23	25	26	27	27
24	24	24	27	28	28	28
25	24	25	27	28	30	30
26	24	26	28	30	31	31
27	24	27	30	31	32	32

values of $\{f_N(s) : s \in S\}$ is a rather simple task involving no more than evaluating $\lfloor s/v_N \rfloor$ and multiplying the result by w_N. That is,

$$f_N(s) = w_6 \times \lfloor s/v_6 \rfloor = 4 \times \lfloor (s/4 \rfloor \ , \ s \in \{0, 1, 2, \dots, 27\} \tag{5.35}$$

These values are listed in Table 5.1
Step 2: Stopping rule. Since $n = 6 - 1$, go to Step 3.

Step 3: Iteration. Set $n = n - 1 = 6 - 1 = 5$. To establish the values of $f_5(s)$, we solve (5.29) for $n = 5$, and $s \in \{0, 1, 2, \dots, 27\}$, namely

$$f_5(s) = \max\left\{w_5 x + f_6(s - v_5 x) : x \in \{0, 1, 2, \dots, \lfloor s/v_5 \rfloor\}\right\} \tag{5.36}$$
$$= \max\left\{3x + f_6(s - 3x) : x \in \{0, 1, \dots, \lfloor (s/3 \rfloor\}\right\} \tag{5.37}$$

For example, for $s = 14$ we obtain

$$f_5(14) = \max\left\{3x + f_6(14 - 3x) : x \in \{0, \dots, \lfloor 14/3 \rfloor\}\right\} \tag{5.38}$$
$$= \max\{0 + f_6(14), 3 + f_6(11), 6 + f_6(8), 9 + f_6(5), 12 + f_6(2)\} \tag{5.39}$$

On assigning f_6 its values from Table 5.1 we obtain the following

$$f_5(14) = \max\{0 + 12, 3 + 8, 6 + 8, 9 + 4, 12 + 0\} \tag{5.40}$$
$$= \max\{12, 11, 14, 13, 12\} = 14 \tag{5.41}$$

The values of $\{f_5(s) : s \in S\}$ are listed in Table 5.1.

Step 4. Go to Step 2.

Step 2. $n = 5 - 1$. Go to Step 3.

Step 3. Iteration. $n = n - 1 = 5 - 1 = 4$. We now solve (5.29) for $n = 4$, and $s \in \{0, 1, 2, \dots, 27\}$, namely

$$f_4(s) = \max\ \left\{w_4 x + f_5(s - v_4) : x \in \{0, 1, 2, \dots, \lfloor s/v_4 \rfloor\}\right\} \tag{5.42}$$
$$= \max\ \left\{9x + f_5(s - 8x) : x \in \{0, 1, 2, \dots, \lfloor s/8 \rfloor\}\right\} \tag{5.43}$$

For example, $s = 19$ yields

$$f_4(19) = \max\ \left\{9x + f_5(19 - 8x) : x \in \{0, 1, \dots, \lfloor 19/8 \rfloor\}\right\} \tag{5.44}$$
$$= \max\ \left\{9x + f_5(19 - 8x) : x \in \{0, 1, 2\}\right\} \tag{5.45}$$

Assigning f_5 the values taken from Table 5.1 we obtain the following:

$$f_4(19) = \max\ \{0 + 19, 9 + 11, 18 + 3\} \tag{5.46}$$
$$= \max\ \{19, 20, 21\} = 21 \tag{5.47}$$

The values of $\{f_4(s) : s \in S\}$ are given in Table 5.1.

As the functions f_3, f_2, and f_1 would be determined in a similar fashion there is no need to repeat the procedure for $n = 3, 2, 1$. The results for these values of n are given in Table 5.1. In *Section 7.2* I describe how the sequence of optimal decisions is determined for problems of this type.

Note that Table 5.1 provides more information than is in fact required for the objective stipulated by (5.17)-(5.19), which is to compute the value of p corresponding to $v^* = 27$, namely $f_1(27)$.

Attention is also called to the fact that, far more efficient dynamic programming solution strategies would be available to solve the knapsack problem in question. These will be examined later in our discussion.

The sole reason for using the above strategy for this purpose is the excellent framework it makes available for highlighting the key ingredients of the formulation and solution of the dynamic programming functional equation stemming from a multistage decision model. □

Having demonstrated a numeric solution of the functional equation let us now examine an analytic solution.

5.3.3 Example

Consider the following problem:

$$p(r) := \max_{(x_1,\dots,x_N)} \sum_{n=1}^{N} \beta^{n-1} x_n^{1/2} \ , \ \beta > 0 \tag{5.48}$$

$$\sum_{n=1}^{N} x_n \leq r \ , \ r > 0 \tag{5.49}$$

$$x_n \geq 0 \ , \ n = 1, 2, \dots, N \tag{5.50}$$

Observe that with no loss of generality we can assume that $r = 1$, because obtaining an optimal solution for this case enables an easy determination of the optimal solutions for the other values of r. Formally, this would involve dividing the constraints by r and invoking the transformation $y_n := x_n/r$. The immediate implication would be that $p(r) = \sqrt{r}\,p(1)$. However, for the purposes of this discussion it will be more instructive to forego this "normalization" procedure.

Suppose then that this is a "resource allocation" problem. In this case, s_n denotes the amount of resource available at stage n following the allocation of $x_1, x_2, \dots, x_{n-1}$ units of the resource in the preceding stages.

Thus, if we start with $s_1 = r$ units we have

$$s_n = r - \sum_{k=1}^{n-1} x_k \ , \ 1 < n \leq N \tag{5.51}$$

units at stage n. This also implies that the transition law is as follows:

$$s_{n+1} = T(n, s, x) = s - x \tag{5.52}$$

Since no more than s_n units can be allocated at stage n, the set of feasible decisions at stage n is as follows:

$$D(n, s) = [0, s] \ , \ n \in \mathbb{N}, s \in S := [0, r] \tag{5.53}$$

Thus, we would set

$$S = \mathbb{D} = [0, r] \tag{5.54}$$

$$S_1 = \{r\} \tag{5.55}$$

Finally, in view of (5.48), we would set opt = max and

$$g(s, x_1, x_2, \dots, x_N) = \sum_{n=1}^{N} \beta^{n-1} x_n^{1/2} \tag{5.56}$$

Next, considering that the objective function is additive, the modified objective functions have this form:

$$g_n(s, x_n, x_{n+1}, \dots, x_N) = \sum_{k=n}^{N} \beta^{k-1} x_k^{1/2} \ , \ n \in \mathbb{N} \tag{5.57}$$

Clearly then, the composition function would be defined thus:

$$\rho(n,s,x,a) = \beta^{n-1}x^{1/2} + a \ , \ a \in \mathbb{R} \tag{5.58}$$

That ρ has this form is due to the fact that by construction

$$g_n(s, x_n, x_{n+1}, \dots, x_N) = \beta^{n-1}x_n^{1/2} + \sum_{k=n+1}^{N} \beta^{k-1}x_k^{1/2} \tag{5.59}$$

$$= \beta^{n-1}x_n^{1/2} + g_{n+1}(s', x_{n+1}, x_{n+2}, \dots, x_N) \tag{5.60}$$

where $s' = T(n,s,x)$.

Also, s and s' are redundant in this case. And so, since $S_1 = \{r\}$, it follows that

$$S_2 = \{T(1,r,x) : x \in D(1,r)\} \tag{5.61}$$
$$= \{T(1,r,x) : 0 \le x \le r\} \tag{5.62}$$
$$= \{r - x : 0 \le x \le r\} \tag{5.63}$$
$$= [0,r] \tag{5.64}$$
$$= S \tag{5.65}$$

and by induction

$$S_n = S = [0,r] \ , \ \forall n \in \mathbb{N} \tag{5.66}$$

Therefore, the functional equation is of the following form:

$$f_N(s) = \max_{x \in D(N,s)} g_N(s,x) \ , \ s \in [0,r] \tag{5.67}$$
$$= \max_{0 \le x \le s} \beta^{N-1}x^{1/2} \tag{5.68}$$

and

$$f_n(s) = \max_{x \in D(n,s)} \left\{ \beta^{n-1}x^{1/2} + f_{n+1}(T(n,s,x)) \right\} \tag{5.69}$$
$$= \max_{0 \le x \le s} \left\{ \beta^{n-1}x^{1/2} + f_{n+1}(s-x) \right\} \ , \ 1 \le n < N, s \in S_N \tag{5.70}$$

Let us examine then an analytic solution of this equation.

Step 1. Initialization. Set $n = N$. Clearly, (5.68) implies that

$$f_N(s) = \beta^{N-1}s^{1/2} \ , \ s \in [0,r] \tag{5.71}$$

Step 2. Stopping rule. Because the functional equation is solved analytically, this step does not apply here. Instead, the values of f_n are searched for by *induction.*

Step 3. Iteration (induction). For $n = N - 1$ (5.70) assumes this form

$$f_{N-1}(s) = \max_{0 \leq x \leq s} \left\{ \beta^{N-2} x^{1/2} + f_N(s-x) \right\} \ , \ 0 \leq s \leq r \tag{5.72}$$

Since (5.71) entails that $f_N(s-x) = \beta^{N-1}(s-x)^{1/2}$, it follows from (5.72) that

$$f_{N-1}(s) = \max_{0 \leq x \leq s} \left\{ \beta^{N-2} x^{1/2} + \beta^{N-1}(s-x)^{1/2} \right\} \ , \ 0 \leq s \leq r \tag{5.73}$$

Observe that the variable s on the right-hand side of (5.73) is treated as a known constant. Thus, from the standpoint of a typical s, the right-hand side of (5.73) boils down to a problem involving the maximization of the following function:

$$L(x) := \beta^{N-2} x^{1/2} + \beta^{N-1}(s-x)^{1/2} \ , \ 0 \leq x \leq s \tag{5.74}$$

Hence, the first derivative of L with respect to x is as follows:

$$L'(x) = \frac{1}{2}\beta^{N-2} \left[x^{-1/2} - \beta(s-x)^{-1/2} \right] \tag{5.75}$$

and the second as follows:

$$L''(x) := -\frac{1}{4}\beta^{N-2} \left[x^{-3/2} + \beta(s-x)^{-3/2} \right] \tag{5.76}$$

Since the second derivative is nonpositive for all $0 \leq x < s$, it follows that L is concave with x on $[0, s]$. Hence, any local maximum is also global. To identify the local maximum the first derivative is equated to zero. This yields $x^{-1/2} - \beta(s-x)^{-1/2} = 0$.

The unique solution to this equation — expressed as a function of the state s — is then as follows:

$$x^*(s) = \frac{s}{1+\beta^2} \ , \ 0 \leq s \leq r \tag{5.77}$$

This implies in turn that

$$\begin{align}
f_{N-1}(s) &= L(x^*(s)) \ , \ 0 \leq s \leq r \tag{5.78} \\
&= L(\frac{s}{1+\beta^2}) \tag{5.79} \\
&= \beta^{N-2} \left[\frac{s}{1+\beta^2} \right]^{1/2} + \beta^{N-1} \left[s - \frac{s}{1+\beta^2} \right]^{1/2} \tag{5.80} \\
&= \beta^{N-2} \left[s(1+\beta^2) \right]^{1/2} \tag{5.81}
\end{align}$$

With these results in hand, the next move is to set $n = N - 2$, whereupon the functional equation takes this form:

$$f_{N-2}(s) = \max_{0 \leq x \leq s} \left\{ \beta^{N-3} x^{1/2} + f_{N-1}(s-x) \right\} \ , \ 0 \leq s \leq r \tag{5.82}$$

Considering that (5.81) entails that

$$f_{N-1}(s-x) = \beta^{N-2}\left[(s-x)(1+\beta^2)\right]^{1/2} \tag{5.83}$$

we now have to solve the following problem:

$$f_{N-2}(s) = \max_{0\le x\le s}\left\{\beta^{N-3}x^{1/2} + \beta^{N-2}\left[(s-x)(1+\beta^2)\right]^{1/2}\right\} \tag{5.84}$$

for $0 \le s \le r$. This calls for a redefinition of L as follows:

$$L(x) = \beta^{N-3}x^{1/2} + \beta^{N-2}\left[(s-x)(1+\beta^2)\right]^{1/2} \ , \ 0 \le x \le s \tag{5.85}$$

Consequently, the first and second derivatives of L are:

$$L'(x) = \frac{1}{2}\beta^{N-3}\left[x^{-1/2} - \beta[(1+\beta^2)]^{1/2}(s-x)^{-1/2}\right] \tag{5.86}$$

and

$$L''(x) := -\frac{1}{4}\beta^{N-3}\left[x^{-3/2} + \beta(1+\beta^2)^{1/2}(s-x)^{-3/2}\right] \tag{5.87}$$

respectively.

Again, the second derivative is nonpositive on $[0, s]$, hence L is concave on $[0, s]$. Equating the first derivative to zero yields the unique solution

$$x^*(s) = \frac{s}{1+\beta^2+\beta^4} \ , \ 0 \le s \le r \tag{5.88}$$

which in turn entails that

$$f_{N-2}(s) = L(x^*(s)) \tag{5.89}$$

$$= \beta^{N-3}\sqrt{\frac{s}{1+\beta^2+\beta^4}} + \beta^{N-2}\sqrt{(1+\beta^2)(s-\frac{s}{1+\beta^2+\beta^4})} \tag{5.90}$$

$$= \beta^{N-3}\sqrt{s(1+\beta^2+\beta^4)} \tag{5.91}$$

Clearly, a pattern is beginning to emerge. The inference to be drawn from this is that it should prove convenient to define

$$b_n := \sum_{k=0}^{N-n}\beta^{2k} \ , \ n = 1, 2, 3, \ldots, N \tag{5.92}$$

whereupon one would be able to appeal to the following *inductive hypothesis*:

$$f_n(s) = \beta^{n-1}\sqrt{s - b_n} \ , \ 1 \le n \le N, 0 \le s \le r \tag{5.93}$$

By inspection it is clear that this hypothesis holds for $n = N$, $n = N - 1$ and $n = N - 2$. Assume then that it holds for $N \ge n \ge k + 1$ as well. In this case, the functional equation pertaining to $n = k$ takes this form

$$f_k(s) = \max_{0\le x\le s}\left\{\beta^{k-1}x^{1/2} + \beta^k f_{k+1}(s-x)\right\} \ , \ 0 \le s \le r \tag{5.94}$$

From (5.93) it follows that

$$f_k(s) = \max_{0\le x\le s}\left\{\beta^{k-1}x^{1/2} + \beta^k\left[(s-x)b_{k+1}\right]^{1/2}\right\} \tag{5.95}$$

Next, define

$$L(x) = \beta^{k-1}x^{1/2} + \beta^k\left[(s-x)b_{k+1}\right]^{1/2}\ ,\ 0\le x\le s \tag{5.96}$$

and consider the first and second derivatives of L, namely

$$L'(x) = \frac{1}{2}\beta^{k-1}\left[x^{-1/2} - \beta b_{k+1}^{1/2}(s-x)^{-1/2}\right] \tag{5.97}$$

and

$$L''(x) = -\frac{1}{4}\beta^{k-1}\left[x^{-3/2} + b_{k+1}^{1/2}(s-x)^{-3/2}\right] \tag{5.98}$$

Again, since β and b_{k+1} are positive, the second derivative is nonpositive, hence L is concave on $[0,s]$. Equating the first derivative to zero yields the unique solution

$$x^*(s) = \frac{s}{1+\beta^2 b_{k+1}}\ ,\ 0\le s\le r \tag{5.99}$$

and since (5.31) implies that $b_k = 1+\beta^2 b_{k+1}$, it follows that

$$x^*(s) = \frac{s}{b_k}\ ,\ 0\le s\le r \tag{5.100}$$

Thus,

$$\begin{aligned}
f_k(s) &= L(x^*(s)) && (5.101)\\
&= \beta^{k-1}\left[\frac{s}{b_k}\right]^{1/2} + \beta^k\left[\left(s-\frac{s}{b_k}\right)b_{k+1}\right]^{1/2} && (5.102)\\
&= \beta^{k-1}\left\{\left[\frac{s}{b_k}\right]^{1/2} + \beta^k\left[\left(s-\frac{s}{b_k}\right)b_{k+1}\right]^{1/2}\right\} && (5.103)\\
&= \beta^{k-1}\left[\frac{s}{b_k}\right]^{1/2}\left\{1+\beta\left[s(b_k-1)\left(\frac{b_{k+1}}{b_k}\right)\right]^{1/2}\right\} && (5.104)
\end{aligned}$$

In short, the inductive hypothesis holds for $n=k$ and therefore it must hold for all $n\in N$. Consequently,

$$\begin{aligned}
f_1(s) &= \beta^0[sb_1]^{1/2}\ ,\ 0\le s\le r && (5.105)\\
&= [sb_1]^{1/2} && (5.106)\\
&= \left[s\sum_{n=0}^{N-1}\beta^{2n}\right]^{1/2} && (5.107)\\
&= \begin{cases}\left[\dfrac{s(1-\beta^{2n})}{1-\beta^2}\right]^{1/2} & ,\ 1\ne\beta>0\\ [sN]^{1/2} & ,\ \beta=1\end{cases} && (5.108)
\end{aligned}$$

In particular, the value of $p(r)$ defined by (5.48) is

$$p(r) = f_1(r) = \begin{cases} \sqrt{\dfrac{r(1-\beta^{2N})}{1-\beta^2}} & , \ 1 \neq \beta > 0 \\ \sqrt{rN} & , \ \beta = 1 \end{cases} \tag{5.109}$$

This confirms the initial observation that $p(r) = \sqrt{r}p(1), r > 0$. □

Having seen how a dynamic programming functional equation would be solved either numerically or analytically, let us now examine a situation where the equation defies a straightforward solution by either one of these conventional approaches. The point to note here is that although the problem in question is eminently amenable to a dynamic programming treatment — in that it yields an unquestionably valid functional equation — the equation proves impervious to standard solution techniques.

5.3.4 Example

Consider the problem

$$p := \max_{(x_1,\dots,x_N)} \sum_{n=1}^{N} w(x_n) \tag{5.110}$$

subject to (5.49)-(5.50), where

$$w(x) = x^{1/2} + e^{-x} + \frac{1+x^2}{1+x^3} + \sin(x) \ , \ 0 \leq x \leq r \tag{5.111}$$

Surely, this problem is similar to the problem studied in *Example 5.3.3,* and yet the objective function in this case induces a functional equation that is exceedingly difficult to solve. The equation in the present case has this form:

$$f_N(s) := \max_{0 \leq x \leq s} \left\{ x^{1/2} + e^{-x} + \frac{1+x^2}{1+x^3} + \sin(x) \right\} \ , \ 0 \leq s \leq r \tag{5.112}$$

and

$$f_n(s) = \max_{0 \leq x \leq s} \left\{ x^{1/2} + e^{-x} + \frac{1+x^2}{1+x^3} + \sin(x) + f_{n+1}(s-x) \right\} \tag{5.113}$$

for $1 \leq n < N$, $0 \leq s \leq r$.

Recall that to initiate *Procedure 5.3.1* we need to have on hand the values of $\{f_N(s)\}$. But this calls for the solution of the following optimization problem:

$$\max_{0 \leq x \leq s} \left\{ x^{1/2} + e^{-x} + \frac{1+x^2}{1+x^3} + \sin(x) \right\} \tag{5.114}$$

Suffice it to say that even if we could somehow solve this problem, it would still be immensely difficult to recover the values of $\{f_n(s) : 0 \leq s \leq r\}$ for $n < N$. $\square$

To round out this general account on solution approaches for truncated dynamic programming functional equations, I wish to point out that, should the sets $\{D(n,s)\}$ be finite and not too large, the functional equation would allow solution by *enumeration* — as was done in *Example 5.3.2.* So in view of the difficulties encountered in *Example 5.3.4*, let us consider the following.

5.3.5 Example

Suppose that the problem featured in *Example 5.3.4* is additionally subject to the constraint

$$x_n \in \{0,1,2,3,\dots\} \ , \ n \in \mathbb{N} \tag{5.115}$$

Also, assume that r is a positive integer. In this case, we would set

$$S = \mathbb{D} = S_n = \{0,1,2,3,\dots,r\} \ , \ n \in \mathbb{N} \tag{5.116}$$
$$D(n,s) = \{0,1,2,\dots,s\} \ , \ n \in \mathbb{N}, s \in S \tag{5.117}$$

Thus, the functional equation would take the form

$$f_N(s) := \max_{x \in \{0,1,2,\dots,s\}} \left\{ x^{1/2} + e^{-x} + \frac{1+x^2}{1+x^3} + \sin(x) \right\} \ , \ s \in S \tag{5.118}$$

and

$$f_n(s) = \max_{x \in \{0,1,2,\dots,s\}} \{w(x) + f_{n+1}(s-x)\} \ , \ 1 \leq n < N, s \in S \tag{5.119}$$

Assuming that N and r are not unduly large, these equations would be readily amenable to solution by enumeration. Namely, the required maximization will be carried out by enumerating the feasible values of x.

To illustrate how this would be done, let us solve $\{f_N(s) : s \in \{0,1,\dots,r\}\}$ for $r = 5$. This will yield the following results:

s	0	1	2	3	4	5
$f_N(s)$	2	2.3679	2.3679	2.3679	2.3679	2.4492

Given these values, we would be able to solve the functional equation pertaining to $n = N - 1$, which is

$$f_{N-1}(s) = \max_{x \in \{0,1,2,\dots,s\}} \{w(x) + f_N(s-x)\}, \ s \in S \tag{5.120}$$

by identifying the feasible values of the decision variable. The results are:

s	0	1	2	3	4	5
$f_{N-1}(s)$	4	2.3679	4.3679	4.7358	4.7358	4.7358

For example, $f_{N-1}(3)$ was evaluated as follows:

$$\begin{aligned}
f_{N-1}(3) &= \max_{x\in\{0,1,2,3\}}\{w(x)+f_N(3-x)\} && (5.121)\\
&= \max\{w(x)+f_N(3-x) : x\in\{0,1,2,3\}\} && (5.122)\\
&= \max\{w(0)+f_N(3), w(1)+f_N(2), w(2)+f_N(1), w(3)+f_N(0)\} && (5.123)\\
&= \max\{4.3679, 4.7358, 4.4730, 4.1390\} && (5.124)\\
&= 4.7358 && (5.125)
\end{aligned}$$

Similarly, once the values of $\{f_{N-1}(s) : s\in S\}$ are known, the same method can be used to compute the values of $\{f_{N-2}(s) : s\in S\}$, and so on for $n = N-3, N-4, \ldots, 1$. Recall that we are interested in the value of $f_1(r)$. □

Before I can proceed to outline the solution techniques for nontruncated equations, I need to call attention to an important point that applies to both truncated and nontruncated functional equations. The point is this. The dynamic programming functional equation can be characterized as a *necessary condition* that is required to be met by the optimal values of the objective functions pertaining to the modified problems. However, as is usually the case with most other necessary optimality conditions, these do not constitute *sufficient* optimality conditions. Namely, they stop short of providing unequivocal assurances that the solution yielded under their terms is indeed the solution sought. The implication is therefore that there may be cases where the functional equation of dynamic programming will generate solutions that are not equal to the functions $\{f_n\}$.

To illustrate this point, assume that the functions $\{f_n\}$ defined by (5.9)-(5.12) satisfy the following dynamic programming functional equation

$$f_n(s) = \underset{x\in D(n,s)}{\text{opt}}\ \rho(n,s,x,f_{n+1}(T(n,s,x)))\ ,\ 1\le n<N, s\in S_n \tag{5.126}$$

What guarantee is there that the functions $\{f_n : n\in\mathbb{N}\}$ recovered by the iterative procedure outlined above are indeed the functions defined by (5.9)-(5.12)? The answer to this question can be formulated as follows:

Lemma 5.3.1 *Assume that N is finite and let*

$$u_N(s) := \underset{x\in D(N,s)}{\text{opt}}\ g_N(s,x)\ ,\ s\in S_N \tag{5.127}$$

Then, if the functional equation

$$u_n(s) = \underset{x\in D(n,s)}{\text{opt}}\ \rho(n,s,x,u_{n+1}(T(n,s,x)))\ ,\ 1\le n<N, s\in S_n \tag{5.128}$$

has a solution, say $u_n^*(s)$, then this solution is UNIQUE. *That is, if there* *are two sequences of real-valued functions $\{u'_n\}$ and $\{u''_n\}$ on $\{S_n\}$ satisfying (5.127)-(5.128), then*

$$u'_n(s) = u''(s)\ ,\ \forall 1 \leq n < N, s \in S_n \tag{5.129}$$

Proof. Let $\{u'_n\}$ and $\{u''\}$ be any two solutions for (5.127)-(5.128). Since for each $s \in S_N$ the right-hand side of (5.127) produces a unique scalar, it follows that $u'_N(s) = u''(s)$, $\forall s \in S_N$. Proceeding by induction, a similar argument for (5.128) yields $u'_n(s) = u''(s)$, $\forall n \in \mathbb{N}$, $s \in S_n$. □

Remark: Note that the uniqueness is with respect to $u_n(s)$, not the decision variable $x \in D(n,s)$.

In other words, if N is finite and the functions defined by (5.9)-(5.12) satisfy the dynamic programming functional equation, then this equation constitutes both a sufficient and a necessary optimality condition for $\{f_n\}$. This is so because with N being finite the functional equation has at most one solution. On the other hand, in cases where N is not finite, the dynamic programming functional equation may have more than one solution. It would therefore be necessary to ascertain that the solution obtained by the method used to solve the functional equation is indeed the desired solution.

5.4 Nontruncated Functional Equations

Using an iterative solution procedure, structured on *Procedure 5.3.1*, to solve nontruncated dynamic programming functional equations cannot even be contemplated because it would simply be impossible to initialize the procedure. Indeed, if N is not finite, solving the modified problems for any given $n \in \mathbb{N}$ would normally be as difficult a task as solving the initial problem.

The solution techniques that are usually employed in such cases, deploy a strategy known as SUCCESSIVE APPROXIMATION. I shall examine this strategy in depth in the next chapter. Here I merely give its general drift.

As a framework for discussion, I shall use the following problem, which is the nontruncated version of the problem featured in *Example 5.3.3.*

5.4.1 Example

Consider the problem

$$p := \max_{(x_1,x_2,\dots)} \sum_{n=1}^{\infty} \beta^{n-1} x_n^{1/2}\ ,\ 0 < \beta < 1 \tag{5.130}$$

$$\sum_{n=1}^{\infty} x_n \leq r\ ,\ r > 0 \tag{5.131}$$

$$x_n \geq 0\ ,\ n \in \mathbb{N} := \{1, 2, 3, \dots\} \tag{5.132}$$

As this is our first encounter with a problem of this type, to begin with, we need to identify the effect that the problem's nontruncated character has on its analysis.

The first thing to note then is that the objective function in this case is stated formally as follows:

$$q(x_1, x_2, \dots) = \sum_{n=1}^{\infty} q_n(x_n) \ , \ x_n \in [0, r] \tag{5.133}$$

where

$$q_n(x_n) := \beta^{n-1} x_n^{1/2} \ , \ 0 < \beta < 1 \tag{5.134}$$

Since $0 < \beta < 1$, it is clear that the values attained by the function q on the set

$$X = \left\{ z \in \mathbb{R}^{\infty} : \sum_{n=1}^{\infty} z_n \leq r, z_n \geq 0 \right\} \tag{5.135}$$

are *bounded.*

Furthermore, given that X is compact and q is continuous on X, we have an assurance that q attains a maximum on X. Namely, the problem under consideration has at least one optimal solution. Secondly, because

$$\sum_{n=1}^{\infty} \beta^{n-1} x_n^{1/2} = x_1^{1/2} + \sum_{n=2}^{\infty} \beta^{n-1} x_n^{1/2} \tag{5.136}$$

$$= x_1^{1/2} + \beta \sum_{n=2}^{\infty} \beta^{n-2} x_n^{1/2} \tag{5.137}$$

it follows that

$$q(x_1, x_2, \dots) = x_1^{1/2} + \beta q(x_2, x_3, \dots) \tag{5.138}$$

$$= x_1^{1/2} + \beta x_2^{1/2} \beta^2 q(x_3, x_4, \dots) \tag{5.139}$$

$$= x_1^{1/2} + \beta x_2^{1/2} + \beta^2 x_3^{1/2} + \beta^3 q(x_4, x_5, \dots) \tag{5.140}$$

$$= x_1^{1/2} + \beta x_2^{1/2} + \beta^2 x_3^{1/2} + \beta^3 x_4^{1/2} + \beta^4 q(x_5, x_6, \dots) \tag{5.141}$$

etc.

As we shall later see, this property of the objective function provides for the formulation of a Markovian decomposition scheme such that all the modified objective functions that it gives rise to are identical to q.

Going back now to our topic of interest.

As in the case of the truncated problem featured in *Example 5.3.3,* we would set the following:

$$S_1 = \{r\} \tag{5.142}$$

$$S = \mathbb{D} = [0, r] \tag{5.143}$$

$$D(n, s) = [0, s] \ , \ n \in \mathbb{N}, s \in S \tag{5.144}$$

$$T(n, s, x) = s - x \ , \ n \in \mathbb{N}, s \in S, x \in D(n, s) \tag{5.145}$$

The objective function would be defined thus:

$$g(s,x_1,x_2,\dots) := \sum_{n=1}^{\infty} \beta^{n-1} x_n^{1/2} \tag{5.146}$$

Except for the set of stages $\mathbb{N}$ now being equal to $\{1,2,3,\dots\}$ rather than to $\{1,2,3,\dots,N\}$, $N<\infty$, the problem under consideration is modeled along the same lines as the truncated problem in *Example 5.3.3.* Thus, in a similar fashion we would set:

$$S_n = S \ , \ \forall n \in \mathbb{N} \tag{5.147}$$

The modified objective functions would be defined as follows:

$$g_n(s,x_n,x_{n+1},\dots) = \sum_{k=n}^{\infty} \beta^{k-1} x_k^{1/2} \tag{5.148}$$

which suggests in turn the following composition function:

$$\rho(n,s,x,a) = \beta^{n-1} x^{1/2} + a \ , \ a \in \mathbb{R} \tag{5.149}$$

In short, if the derivation process yielding the truncated functional equation in *Example 5.3.3* is to be repeated step by step in the case of the nontruncated problem in question, the end product will be the same functional equation, except that now $N=\infty$. In other words, the functional equation will take this form:

$$f_n(s) = \max_{0\le x\le s} \{\beta^{n-1} x^{1/2} + f_{n+1}(s-x)\} \ , \ n \in \{1,2,3,\dots\} \tag{5.150}$$

for $0 \le s \le r$.

The difficulty posed by this equation, as indicated above, is that it will be impossible to initialize *Procedure 5.3.1.* We would therefore have to approach its solution with a different set of ideas in mind.

The first step would be then to transform this equation so that instead of infinitely many functions $\{f_n\}$, we would have only **one** function to contend with. This is accomplished as follows.

1. From the definition of the functions $\{g_n\}$ it follows that

$$\begin{align} g_n(s,x_n,x_{n+1},\dots) &= \sum_{k=n}^{\infty} \beta^{k-1} x_k^{1/2} \tag{5.151}\\ &= \beta^{n-1} \sum_{k=n}^{\infty} \beta^{k-n} x_k^{1/2} \tag{5.152}\\ &= \beta^{n-1} \sum_{k=1}^{\infty} \beta^{k-1} x_{n+k-1}^{1/2} \tag{5.153}\\ &= \beta^{n-1} g(s,x_n,x_{n+1},\dots) \tag{5.154}\end{align}$$

so that

$$g_n = \beta^{n-1} g \ , \ n \in \mathbb{N} \tag{5.155}$$

2. As n has no bearing on the sets $\{D(n,s)\}$, it follows that for every $s' \in S$, all the modified problems $\{Problem\ P(n,s') : n \in \mathbb{N}\}$ have the same set of feasible solutions. Therefore, (5.44) entails that

$$f_n(s) = \beta^{n-1} f(s) \ , \ \forall n \in \mathbb{N}, s \in S \tag{5.156}$$

3. Substituting $\beta^{n-1}f(s)$ for $f_n(s)$ and $\beta^n f(s-x)$ for $f_{n+1}(s-x)$ in (5.150) and then dividing both sides by β^{n-1}, we obtain the following functional equation:

$$f(s) = \max_{0 \leq x \leq s} \{x^{1/2} + \beta f(s-x)\} \ , \ n \in \mathbb{N}, s \in S \tag{5.157}$$

recalling that the function $f = f_1$ is defined as follows:

$$f(s) := \max_{(x_1, x_2, \dots)} \sum_{n=1}^{\infty} \beta^n x_n^{1/2} \ , \ 0 \leq s \leq r \tag{5.158}$$

$$\sum_{n=1}^{\infty} x_n \leq s \tag{5.159}$$

$$x_n \geq 0, n \in \{1, 2, 3, \dots\} \tag{5.160}$$

In view of (5.156), it is sufficient to recover a solution for the function f as the sequence $\{f_n\}$ is readily obtainable from f. This allows treating (5.157) rather than (5.150) as the functional equation of interest.

This clear, let us now examine how the successive approximation method would handle equation (5.157). To be able to highlight the method's main points, we ignore the fact that the equation derives from a dynamic programming model formulation, as in essence this equation sets the following task:

Given the constants $r > 0$ and $0 < \beta < 1$, find a real-valued function u on the interval $[0,r]$ such that

$$u(s) = \max_{0 \leq x \leq s} \left\{ x^{1/2} + \beta u(s-x) \right\}, \ \forall s \in [0,r] \tag{5.161}$$

The successive approximation strategy would be set into motion by postulating an *initial approximation* for u, say

$$u^{(0)}(s) := 0 \ , \ \forall s \in [0,r] \tag{5.162}$$

Then, using the procedure implied by

$$u^{(k+1)}(s) := \max_{0 \leq x \leq s} \{x^{1/2} + \beta u^{(k)}(s-x)\} \ , \ s \in [0,r] \tag{5.163}$$

this approximation would be successively updated. Thus, after assigning $u^{(0)}$ the initial value, we would solve the equation

$$u^{(1)}(s) := \max_{0\leq x\leq s} \{x^{1/2} + \beta u^{(0)}(s-x)\} \ , \ s\in[0,r] \tag{5.164}$$

and evaluate the function $u^{(1)}$.

If $u^{(1)}$ is found to be equal to $u^{(0)}$ the procedure would be stopped as a solution had been recovered. Otherwise, it would continue with

$$u^{(2)}(s) := \max_{0\leq x\leq s} \{x^{1/2} + \beta u^{(1)}(s-x)\} \ , \ s\in[0,r] \tag{5.165}$$

In short, the approximation process will carry on, in this fashion, until $u^{(k+1)} = u^{(k)}$ for some k, or until it becomes evident that the sequence $\{u^{(k)}\}$ unfolds in a *pattern.* If the latter rather than the former will prove to be the case, then it will be necessary to test whether the said sequence converges and whether its limit satisfies the functional equation.

To go back then to *Example 5.3.1,* observe that (5.162)-(5.164) yield

$$u^{(1)}(s) := \max_{0\leq x\leq s} \ x^{1/2} \ , \ s\in[0,r] \tag{5.166}$$

Hence,

$$u^{(1)}(s) = s^{1/2} \ , \ s\in[0,r] \tag{5.167}$$

Thus, applying (5.163) with $k=1$ yields

$$u^{(2)}(s) :\, = \max_{0\leq x\leq s} \{x^{1/2} + \beta u^{(1)}(s-x)\} \tag{5.168}$$

$$= \max_{0\leq x\leq s} \{x^{1/2} + \beta (s-x)^{1/2}\} \ , \ s\in[0,r] \tag{5.169}$$

Note that this problem is similar to the one encountered in the solution of the problem that was analyzed in *Example 5.3.3.* So, equating the first derivative of the function being maximized in (5.169) to zero yields the following optimal solution:

$$x^*(s) := \frac{s}{1+\beta^2} \ , \ s\in[0,r] \tag{5.170}$$

Since the function being maximized in (5.169) is concave, the implication is that this is a global maximum point. Hence,

$$u^{(2)}(s) = [x^*(s)]^{1/2} + \beta\,[s - x^*(s)]^{1/2} \tag{5.171}$$

$$= \left[\frac{s}{1+\beta^2}\right]^{1/2} + \beta\left[s - \frac{s}{1+\beta^2}\right]^{1/2} \tag{5.172}$$

$$= \left[1+\beta^2\right]\left[\frac{s}{1+\beta^2}\right]^{1/2} \tag{5.173}$$

$$= \left[s(1+\beta^2)\right]^{1/2} \ , \ s\in[0,r] \tag{5.174}$$

Thus, for $k = 2$ we obtain

$$u^{(3)}(s) := \max_{0\leq x\leq s} \left\{x^{1/2} + \beta u^{(2)}(s-x)\right\} \ , \ s \in [0,r] \tag{5.175}$$

$$= \max_{0\leq x\leq s} \left\{x^{1/2} + \beta \left[(1+\beta^2)(s-x)\right]^{1/2}\right\} \tag{5.176}$$

Again, note that this problem is also similar to the problem encountered in *Example 5.2.3.* Here as well, the function being maximized is concave. Equating this function's first derivative to zero yields the optimal solution

$$x^*(s) = \frac{s}{1+\beta^2+\beta^4} \ , \ s \in [0,r] \tag{5.177}$$

which in turn yields

$$u^{(3)}(s) = [x^*(s)]^{1/2} + \beta \left[(1+\beta^2)(s - x^*(s))\right]^{1/2} \tag{5.178}$$

$$= \left[s(1+\beta^2+\beta^4)\right]^{1/2} \ , \ s \in [0,r] \tag{5.179}$$

We now proceed by induction using the following inductive hypothesis

$$u^{(m)}(s) = [sb_m]^{1/2} \ , \ m \in \{1,2,\dots\}, s \in [0,r] \tag{5.180}$$

where

$$b_m := \sum_{k=0}^{m-1} \beta^{2k} \ , \ m \in \{1,2,3,\dots\} \tag{5.181}$$

By inspection, it is clear that this inductive hypothesis holds for $1 \leq m \leq 3$. We therefore assume that it is true for $m = 4, 5, \dots, n$ and we consider $m = n+1$. In this case we have

$$u^{(n+1)}(s) := \max_{0\leq x\leq s} \{x^{1/2} + \beta u^{(n)}(s-x)\} \ , \ s \in [0,r] \tag{5.182}$$

$$= \max_{0\leq x\leq s} \left\{x^{1/2} + \beta[b_n(s-x)]^{1/2}\right\} \tag{5.183}$$

Solving this problem in the standard classical fashion we obtain the following optimal solution:

$$x^*(s) = \frac{s}{b_{n+1}} \ , \ s \in [0,r] \tag{5.184}$$

which in turn yields

$$u^{(n+1)}(s) = [x^*(s)]^{1/2} + \beta u^{(n)}(s - x^*(s)) \tag{5.185}$$

$$= \left[\frac{s}{b_{n+1}}\right]^{1/2} + \beta\left[b_n\left(s - \frac{s}{b_{n+1}}\right)\right]^{1/2} \tag{5.186}$$

$$= \left[\frac{s}{b_{n+1}}\right]^{1/2}\left[1 + \beta\left[b_n(b_{n+1} - 1)\right]^{1/2}\right] \tag{5.187}$$

$$= \left[\frac{s}{b_{n+1}}\right]^{1/2}\left[1 + \beta\left[b_n(\beta^2 b_n)\right]^{1/2}\right] \tag{5.188}$$

$$= \left[1 + \beta^2 b_n\right]\left[\frac{s}{b_{n+1}}\right]^{1/2} \tag{5.189}$$

$$= b_{n+1}\left[\frac{s}{b_{n+1}}\right]^{1/2} \tag{5.190}$$

$$= \left[s b_{n+1}\right]^{1/2} \tag{5.191}$$

Note that the above derivation invoked twice the relation $b_{n+1} = 1 + \beta^2 b_n$ which follows directly from the definition of b_n in (5.181).

This implies that the inductive hypothesis holds for $m = n+1$, and hence, that it is true for all $m \geq 1$.

Turning now to the sequence $\{u^{(n)}\}$, we first need to determine whether it converges and, if it does, whether its limit satisfies the functional equation. So, we define

$$u^*(s) := \lim_{n\to\infty} u^{(n)}(s) \tag{5.192}$$

$$= \lim_{n\to\infty} \left[s b_n\right]^{1/2} \tag{5.193}$$

$$= \lim_{n\to\infty} \left[s \sum_{k=0}^{n-1} \beta^{2k}\right]^{1/2} \tag{5.194}$$

$$= \lim_{n\to\infty} \left[\frac{s(1-\beta^{2n})}{1-\beta^2}\right]^{1/2} \tag{5.195}$$

Since $1 > \beta > 0$ it follows that

$$u^*(s) = \left[\frac{s}{1-\beta^2}\right]^{1/2} \ , \ s \in [0, r] \tag{5.196}$$

To ascertain that this function indeed satisfies the functional equation, we define

$$v(s) := \max_{0 \leq x \leq s} \left\{x^{1/2} + \beta u^*(s - x)\right\} \ , \ s \in [0, r] \tag{5.197}$$

As we need to show that $v(s) = u^*(s)$, $\forall s \in [0, r]$, we proceed as follows. From (5.196)-(5.197) it follows that

$$v(s) = \max_{0 \leq x \leq s} \left\{x^{1/2} + \beta\left[\frac{s - x}{1 - \beta^2}\right]^{1/2}\right\} \ , \ s \in [0, r] \tag{5.198}$$

We therefore define

$$L(x) := x^{1/2} + \beta \left[\frac{s-x}{1-\beta^2} \right]^{1/2} , 0 \leq x \leq s \tag{5.199}$$

so that the first and second derivatives of L are

$$L'(x) := \frac{1}{2} \left\{ x^{-1/2} - \frac{\beta}{1-\beta^2} \left[\frac{s-x}{1-\beta^2} \right]^{-1/2} \right\} \tag{5.200}$$

and

$$L''(x) := -\frac{1}{4} \left\{ x^{-3/2} + \frac{\beta}{(1-\beta^2)^2} \left[\frac{s-x}{1-\beta^2} \right]^{-3/2} \right\} \tag{5.201}$$

Because the second derivative of L is nonpositive for all $0 \leq x \leq s$, it follows that for each $0 \leq s \leq r$, the function L is concave on $[0, s]$. Thus, equating the first derivative to zero, we obtain the optimal solution

$$x^*(s) = s(1-\beta^2) \ , \ \ s \in [0, r] \tag{5.202}$$

which in turn yields

$$\begin{aligned} v(s) &= L(x^*(s)) && (5.203) \\ &= [x^*(s)]^{1/2} + \beta u^*(s-x) && (5.204) \\ &= \left[s(1-\beta^2) \right]^{1/2} + \beta \left[\frac{s - x^*(s)}{1-\beta^2} \right]^{1/2} && (5.205) \\ &= \left[s(1-\beta^2) \right]^{1/2} + \beta \left[\frac{s - s(1-\beta^2)}{1-\beta^2} \right]^{1/2} && (5.206) \\ &= \left[s(1-\beta^2) \right]^{1/2} + \beta \left[\frac{s\beta^2}{1-\beta^2} \right]^{1/2} && (5.207) \\ &= \left[\frac{s}{1-\beta^2} \right]^{1/2} \left[(1-\beta^2) + \beta^2 \right] && (5.208) \\ &= \left[\frac{s}{1-\beta^2} \right]^{1/2} && (5.209) \\ &= u^*(s) && (5.210) \end{aligned}$$

We conclude then that the function u^* defined by (5.196) is a solution for the functional equation (5.161). $\square$

An important question remains, however. Much as the function u^* turned out to be a solution to the functional equation in question, namely (5.161), what assurance is there that $u^* = f$? This question arises because there is nothing in the definition of function f, namely in (5.158)-(5.160), to provide for this.

This matter will be taken up in the following chapter. For the moment it suffices to note that one would need to show that the functional equation (5.161) has a *unique* solution. In demonstrating that the equation has only *one* solution, and considering that f is *a* solution to this equation, one would thereby show that the solution obtained is necessarily equal to f.

As we shall see, certain dynamic programming functional equations have more than one solution. In such cases, therefore, it will be necessary to track down the solution that meets the definition of the functions $\{f_n :, n \in \mathbb{N}\}$.

And to go back to the problem featured in *Example 5.4.1,* we may conclude the following. The foregoing analysis suggests that the function f defined by (5.158)-(5.160) is as follows:

$$f(s) = \left[\frac{s}{1-\beta^2}\right]^{1/2} \ , \ s \in [0,r] \tag{5.211}$$

Furthermore, (5.202) suggests that the optimal decisions expressed as a function of the state variable are *independent of the stage variable* and take the form:

$$x^*(s) = s(1-\beta^2) \ , \ 0 \leq s \leq r \tag{5.212}$$

These results will be corroborated in the next chapter where I shall show that the function f defined by (5.158)-(5.160) is indeed the unique solution for the functional equation (5.161). □

5.5 Summary

The discussion in this chapter illustrated that the decisive factor determining how a given dynamic programming functional equation would be solved is whether the equation is truncated or nontruncated.

Truncated equations are normally solved by direct methods that iterate on the stage variable, whereas nontruncated equations are normally solved by means of successive approximation procedures.

As regards the solution obtained for the equation, we saw that in the case of nontruncated problems the solution must be shown to be equal to the functions $\{f_n\}$. In the case of truncated problems, one is assured of this a priori.

It must be appreciated that usually a given functional equation can be solved by more than one method. The most famous case in point is the *shortest path problem.* As we shall see, functional equations associated with problems of this type can be solved by a variety of direct and successive approximation methods.

In the next chapter I take a closer look at the successive approximation strategy, and in *Chapter 7* I show that the solution of the dynamic programming functional equation provides for the recovery of optimal policies.

And as a final note, I should point out that there are, of course, methods that by design do not seek “exact” solutions to the dynamic programming functional equation but settle instead for an approximation (eg. Heidari et al. [1971], Gal [1989], Sutton [1990], Sniedovich and Voß [2006], Powell [2007]). The rationale behind this “heuristic” approach is the forbidding computational requirements that “exact” methods can have in certain cases (see *Chapter 8*).

6

Successive Approximation Methods

6.1 Introduction

In this chapter I examine in greater detail the basic structure of the generic successive approximation method as it is used in the solution of dynamic programming functional equations. I begin with an investigation of those elements of the successive approximation method that are pivotal in the solution of a nontruncated dynamic programming functional equation, namely *convergence* and *uniqueness.* After that I outline in broad terms an alternative method for nontruncated dynamic programming functional equations — which I call *truncation method.* Then, following a brief discussion on the nature of stationary models, I conclude with an examination of the relation between the generic successive approximation method and the truncation method.

The nontruncated dynamic programming functional equation that is under examination here has this form:

$$f_n(s) = \operatorname*{opt}_{x \in D(n,s)} \rho(n, s, x, f_{n+1}(T(n, s, x))) \ , \ n \in \mathbb{N}, s \in S_n \tag{6.1}$$

where the functions $\{f_n\}$ are defined as follows:

$$f_n(s) = \operatorname*{opt}_{(x_1, x_2, \dots)} g_n(s, x_n, x_{n+1}, \dots) \ , \ n \in \mathbb{N}, s \in S_n \tag{6.2}$$

$$x_m \in D(m, s_m) \ , \ m \geq n \tag{6.3}$$

where $s_n = s$, $s_{m+1} = T(m, s_m, x_m)$, and $\mathbb{N} = \{1, 2, 3, \dots\}$.

For the purposes of this discussion it is convenient to incorporate the stage variable as a formal argument of f and rewrite this equation as follows:

$$f(n, s) = \operatorname*{opt}_{x \in D(n,s)} \rho(n, s, x, f(n+1, T(n, s, x))) \ , \ n \in \mathbb{N}, s \in S_n \tag{6.4}$$

In this framework f is regarded an unknown real-valued function on $\{(n, s) : n \in \mathbb{N}, s \in S_n\}$. The objective is to find a solution to this equation, that is a function f that satisfies (6.4).

My investigation will be conducted in the general framework of the following prototype functional equation:

$$u = Au \ , \ u \in U \tag{6.5}$$

where U is a nonempty set and A is a map from U into itself, namely to each $u \in U$ the map A assigns an element in U denoted by Au.

To be precise, I shall study the ramifications of convergence and uniqueness for the solution of a nontruncated dynamic programming functional equation through the medium of equation (6.5). The advantage in this approach is that it will enable couching the presentation in the language of classical *functional analysis* and to appeal to its established results. The terminology and results that are pertinent to this discussion are summed up in *Appendix A*.

Note that the following is an abstraction of (6.4) where u represents f and the operation A represents the operation

$$(Au)(n,s) := \operatorname*{opt}_{x \in D(n,s)} \rho(n,s,x,u(n+1,T(n,s,x))) \ , \ n \in \mathbb{N}, s \in S_n \quad (6.6)$$

So, the question is how would the successive approximation procedure — delineated in *Chapter 5* — handle the equation $Au = u$.

6.2 Motivation

A rough sketch outlining the successive approximation procedure for solving equation (6.5) would run as follows:

6.2.1 Procedure

Step 1. *Initialization*
Set $m = 0$ and select some element $u^{(m)}$ from U.

Step 2. *Iteration*
Set $u^{(m+1)} := Au^{(m)}$.

Step 3. *Stopping rule*
If $u^{(m+1)} = u^{(m)}$ set $u^* = u^{(m)}$ and stop. Otherwise set $m = m+1$ and go to Step 2. □

Now, as can be gathered from *Chapter 5,* this procedure basically has two possible "useful" outcomes:

· The procedure *terminates*, thus yielding a $u^* \in U$ such that $u^* = Au^*$.

· The procedure does not terminate but the sequence $\{u^{(m)}\}$ that it generates *converges* to some $u^* \in U$ such that $u^* = Au^*$.

Contrary to the first case, whose implications are clear, the latter warrants special attention, obviously because of the infinite sequence that it engenders. I shall therefore ignore the first and concentrate entirely on the second. To illustrate the type of issues that the second situation gives rise to, let us consider the following simple example.

6.2.2 Example

Consider the equation

$$q(z) = z^2 + q(\beta z) \ , \ 0 < \beta < 1, z \in Z := [0, \infty) \quad (6.7)$$

where q is an unknown real-valued function on Z.

Our goal is to find a function q satisfying this equation. Let U denote the set of all bounded real-valued functions on Z and define the map A as follows:

$$(Au)(z) := z^2 + u(\beta z)\ ,\ z \in Z \tag{6.8}$$

where $(Au)(z)$ denotes the value assigned to $z \in Z$ by the function Au. Then *Procedure 6.2.1* would yield a sequence $\{u^{(m)}\}$ such that

$$u^{(m+1)}(z) = (Au^{(m)})(z)\ ,\ z \in Z \tag{6.9}$$

$$= z^2 + u^{(m)}(\beta z) \tag{6.10}$$

For simplicity, define the initial approximation, namely $u^{(0)}$, as follows:

$$u^{(0)}(z) := 0\ ,\ \forall z \in Z \tag{6.11}$$

Then the procedure would yield the following sequence of approximations:

$$u^{(1)}(z) := z^2 + u^{(0)}(\beta z)\ ,\ z \in Z \tag{6.12}$$

$$= z^2 + 0 \tag{6.13}$$

$$= z^2 \tag{6.14}$$

$$u^{(2)}(z) := z^2 + u^{(1)}(\beta z)\ ,\ z \in Z \tag{6.15}$$

$$= z^2 + (\beta z)^2 \tag{6.16}$$

$$= z^2(1 + \beta^2) \tag{6.17}$$

$$u^{(3)}(z) := z^2 + u^{(2)}(\beta z)\ ,\ z \in Z \tag{6.18}$$

$$= z^2 + (\beta z)^2(1 + \beta^2) \tag{6.19}$$

$$= z^2(1 + \beta^2 + \beta^4) \tag{6.20}$$

By induction then,

$$u^{(m)}(z) = z^2 \sum_{k=0}^{m-1} \beta^{2k}\ ,\ z \in Z, m = 1, 2, 3, \ldots \tag{6.21}$$

It follows therefore that

$$u^*(z) := \lim_{m\to\infty} u^{(m)}(z)\ ,\ z \in Z \tag{6.22}$$

$$= \lim_{m\to\infty} z^2 \sum_{k=0}^{m-1} \beta^{2k} \tag{6.23}$$

$$= z^2 \lim_{m\to\infty} \frac{1 - \beta^{2m}}{1 - \beta^2} \tag{6.24}$$

$$= \frac{z^2}{1 - \beta^2} \tag{6.25}$$

To ascertain that the function u^* satisfies the equation $u = Au$ we would define $u'' := Au^*$ and aim to show that $u'' = u^*$, namely that $u''(z) = u^*(z)$, $\forall z \in Z$. Notice then that by definition:

$$\begin{aligned} u''(z) &= (Au^*)(z), z \in Z && (6.26)\\ &= z^2 + u^*(\beta z) && (6.27)\\ &= z^2 + \frac{(\beta z)^2}{1-\beta^2} && (6.28)\\ &= \frac{z^2}{1-\beta^2} && (6.29)\\ &= u^*(z) && (6.30)\end{aligned}$$

This means that u^* is indeed a solution to the functional equation

$$u(z) = z^2 + u(\beta z)\ ,\ z \in [0,\infty) \qquad \square \tag{6.31}$$

So, the basic question is this:

> *Under what conditions does the sequence $\{u^{(m)}\}$ generated by Procedure 6.2.1 converge to a $u^* \in U$ such that $u^* = Au^*$?*

Or, to put it differently, the following two separate, yet closely related, questions arise:

· What conditions guarantee the convergence of the sequence $\{u^{(m)}\}$ generated by *Procedure 6.2.1* ?

· What conditions guarantee that the limit of this sequence satisfies the equation $u = Au$?

Keep in mind that these questions about $u = Au$ are of concern to us only insofar as they are immediately applicable to the solution of the nontruncated dynamic programming equation.

Another important question that we shall have to answer in this chapter is the following:

> *What conditions ensure that a solution generated by Procedure 6.2.1 is the desired solution to the given dynamic programming functional equation?*

As I noted in *Chapter 5,* this question arises because a nontruncated dynamic programming functional equation may have more than one solution. We must therefore be able to determine whether the solution obtained is the solution sought.

The next two examples illustrate this issue.

6.2.3 Example

Consider the multistage decision model comprising the following constructs:

$$S = S_1 = \mathbb{D} = [0, r] \ , \ r > 0 \tag{6.32}$$

$$D(n, s) = [0, s], n \in \mathbb{N} \ , \ s \in S \tag{6.33}$$

$$T(n, s, x) = \beta(s - x) \ , \ 0 < \beta < 1, n \in \mathbb{N}, s \in S, x \in D(n, s) \tag{6.34}$$

$$g(s, x_1, x_2, \dots) = \sum_{n=1}^{\infty} x_n^{1/2} \tag{6.35}$$

Since the objective function is additive, to derive a functional equation we invoke an additive decomposition scheme consisting of the following modified objective functions:

$$g_n(s_n, x_n, x_{n+1}, \dots) = \sum_{m=n}^{\infty} x_m^{1/2} \ , \ n \in \mathbb{N} \tag{6.36}$$

and the following additive decomposition function:

$$\rho(n, s, x, a) = x^{1/2} + a \ , \ n \in \mathbb{N} \ , \ s \in S, x \in D(n, s), a \in \mathbb{R} \tag{6.37}$$

The resulting dynamic programming functional equation therefore takes this form:

$$\begin{aligned} f_n(s) &= \underset{x \in D(n,s)}{\text{opt}} \ \rho(n, s, x, f_{n+1}(T(n, s, x))) \ , \ n \in \mathbb{N}, s \in S_n \\ &= \underset{0 \le x \le s}{\text{opt}} \{x^{1/2} + f_{n+1}(\beta(s - x))\} \end{aligned} \tag{6.38}$$

Now, by definition, *Problem* $P(n, s)$ is of the form

$$f_n(s) := \underset{(x_n, x_{n+1}, \dots)}{\text{opt}} \ g_n(s, x_1, x_2, \dots) \tag{6.39}$$

$$= \underset{(x_n, x_{n+1}, \dots)}{\text{opt}} \ \sum_{k=n}^{\infty} x_k^{1/2} \tag{6.40}$$

$$x_k \in [0, s_k] \ , \ k \ge n \tag{6.41}$$

$$s_n = s \tag{6.42}$$

$$s_{k+1} = \beta(s_k - x_k) \ , \ k \ge n \tag{6.43}$$

Clearly, n has no bearing on $f_n(s)$, hence,

$$f_1(s) = f_n(s) \ , \ \forall n \in \mathbb{N}, s \in S \tag{6.44}$$

whereupon (6.38) yields

$$f_1(s) = \underset{0 \le x \le s}{\text{opt}} \{x^{1/2} + f_1(\beta(s - x))\} \ , \ 0 \le s \le r \tag{6.45}$$

Now, set opt = max and $u = f_1$, in which case (6.45) yields

$$u(s) = \max_{0 \leq x \leq s} \{x^{1/2} + u(\beta(s - x))\} \ , \ 0 \leq s \leq r \tag{6.46}$$

Thus, in the $u = Au$ framework, we can rewrite (6.46) as follows:

$$(Au)(s) := \max_{0 \leq x \leq s} \left\{x^{1/2} + u(\beta(s - x))\right\} \ , \ 0 \leq s \leq r, u \in U \tag{6.47}$$

where U denotes the set of all bounded real-valued functions on $[0, r]$. Consider now the function $u'' \in U$ defined as follows:

$$u''(s) := \left[\frac{s}{1 - \beta}\right]^{1/2} \ , \ 0 \leq s \leq r \tag{6.48}$$

To show that u'' is a solution to (6.46) we need to show that

$$u''(s) = (Au'')(s) \ , \ \forall s, 0 \leq s \leq r \tag{6.49}$$

Observe then that from (6.47)

$$(Au'')(s) = \max_{0 \leq x \leq s} \left\{x^{1/2} + u''(\beta(s - x))\right\} \ , \ 0 \leq s \leq r \tag{6.50}$$

$$= \max_{0 \leq x \leq s} \left\{x^{1/2} + \left[\frac{\beta(s - x)}{1 - \beta}\right]^{1/2}\right\} \tag{6.51}$$

Thus, focusing on a given $s \in S$, we define

$$L(x) := x^{1/2} + \left[\frac{\beta(s - x)}{1 - \beta}\right]^{1/2} \ , \ 0 \leq x \leq s \tag{6.52}$$

On equating the first derivative of L to zero we obtain the unique solution

$$x^*(s) = s(1 - \beta) \ , \ 0 \leq s \leq r \tag{6.53}$$

Since L is concave on $[0, s]$, it follows that $x^*(s)$ is the optimal solution to (6.13). Hence,

$$(Au'')(s) = L(x^*(s)) \tag{6.54}$$

$$= [x^*(s)]^{1/2} + \left[\frac{\beta(s - x^*(s))}{1 - \beta}\right]^{1/2} \tag{6.55}$$

$$= [s(1 - \beta)]^{1/2} + \left[\frac{\beta(s - s(1 - \beta))}{1 - \beta}\right]^{1/2} \tag{6.56}$$

$$= s^{1/2}\left[(1 - \beta)^{1/2} + \frac{\beta}{(1 - \beta)^{1/2}}\right] \tag{6.57}$$

$$= \left[\frac{s}{1 - \beta}\right]^{1/2} \tag{6.58}$$

$$= u''(s) \tag{6.59}$$

The conclusion then is that u'' is a solution to the equation $Au = u$. It turns out, however, that for any $a \in \mathbb{R}$, the function

$$u_a(s) : = a + u''(s) \ , \ 0 \leq s \leq r \tag{6.60}$$

$$= a + \left[\frac{s}{1-\beta}\right]^{1/2} \tag{6.61}$$

also constitutes a solution to this equation. That is, because by definition

$$(Au_a)(s) := \max_{0\leq x\leq s} \left\{x^{1/2} + u_a(\beta(s-x))\right\} \ , \ 0 \leq s \leq r \tag{6.62}$$

it follows that

$$(Au_a)(s) = \max_{0\leq s\leq r} \{x^{1/2} + a + u''(\beta(s-x))\} \tag{6.63}$$

$$= \max_{0\leq x\leq s} \left\{x^{1/2} + a + \left[\frac{s}{1-\beta}\right]^{1/2}\right\} \tag{6.64}$$

$$= a + \max_{0\leq x\leq s} \left\{x^{1/2} + \left[\frac{s}{1-\beta}\right]^{1/2}\right\} \tag{6.65}$$

$$= a + \max_{0\leq x\leq s} \left\{x^{1/2} + u''(\beta(s-x))\right\} \tag{6.66}$$

$$= a + (Au'')(s) \tag{6.67}$$

$$= a + u''(s) \tag{6.68}$$

$$= u_a(s) \tag{6.69}$$

In short, the functional equation specified by (6.46) has an infinite number of solutions of the form

$$u_a(s) = a + \left[\frac{s}{1-\beta}\right]^{1/2} \ , \ a \in \mathbb{R}, 0 \leq s \leq r \tag{6.70}$$

The question therefore is: which solution, if any, is the desired solution to (6.45)? Recall that to count as the desired solution, it must comply with the *definition* of f_1 stipulated by (6.39)-(6.43). □

6.2.4 Example

As already shown in *Chapter 4,* a functional equation deriving from a nontruncated final state model would be of this form:

$$f_n(s) = \underset{x\in D(n,s)}{\text{opt}} f_{n+1}(T(n,s,x)) \ , \ n \in \mathbb{N}, s \in S_n \tag{6.71}$$

Thus, let U be the set of all the bounded real-valued functions on the set $Z =: \{(n,s) : n \in \mathbb{N},\ s \in S_n\}$ and define

$$(Au)(n,s) := \underset{x\in D(n,s)}{\text{opt}} u(n+1, T(n,s,x)) \ , \ (n,s) \in Z \tag{6.72}$$

Equation $u = Au$, in this case, has infinitely many solutions in U. That is, for any $r \in \mathbb{R}$ the function

$$u_r(n,s) = r \ , \ \forall (n,s) \in Z \tag{6.73}$$

trivially satisfies the equation $u_r = Au_r$. □

Before I can turn to the investigation of the questions posed above, I must first clarify a few points.

6.3 Preliminaries

Using the equation $u = Au$ as a framework for the study of convergence and uniqueness in the solution of nontruncated dynamic programming functional equations obviously implies that the nontruncated dynamic programming functional equation is viewed as a manifestation of $u = Au$. This means that the object u is considered to be a real-valued function on the set $Z := \{(n,s) : n \in \mathbb{N},\ s \in S_n\}$, so that the equation $u = Au$ is understood to set the following task:

Given a set Z, a set U consisting of all bounded real-valued functions on Z, and a map A from U into itself, find a $u \in U$ such that $u = Au$.

All the same, the two equations namely $Au = u$ and the dynamic programming functional equation, must still be reconciled. This is so because, as you will recall, the nontruncated dynamic programming functional equation under consideration, namely

$$f_n(s) = \underset{x \in D(n,s)}{\text{opt}} \ \rho(n,s,x,f_{n+1}(T(n,s,x))) \ , \ n \in \mathbb{N}, s \in S_n \tag{6.74}$$

involves infinitely many unknown real-valued functions $\{f_n\}$; whereas the equation $u = Au$ involves only one such function. We saw at the outset that the sequence $\{f_n : n \in \mathbb{N}\}$ can be slightly modified so as to allow regarding it a single function rather than a sequence of functions.

Let us quickly repeat this simple exercise, this time in a somewhat more formal manner. First, define

$$Z := \{(n,s) : n \in \mathbb{N}, s \in S_n\} \tag{6.75}$$

and

$$f^*(n,s) := f_n(s), (n,s) \in Z \tag{6.76}$$

This will allow the function f^* to satisfy the equation

$$f^*(n,s) = \underset{x \in D(n,s)}{\text{opt}} \ \rho(n,s,x,f^*(n+1,T(n,s,x))) \ , \ \forall (n,s) \in Z \tag{6.77}$$

In turn, with U being treated as the set of all bounded real-valued functions on Z, the map A will be defined as follows:

$$(Au)(n,s) := \operatorname*{opt}_{x\in D(n,s)} \rho(n,s,x,u(n+1,T(n,s,x))) \tag{6.78}$$

for all $u \in U, (n,s) \in Z$.

But the truth is that dynamic programming functional equations that are amenable to the successive approximation procedure very often lend themselves to a simpler formulation. Indeed, in many cases $f_n = \beta^{n-1} f_1$, $0 < \beta < 1$, and $S_n \subseteq S$, for all $n \in \mathbb{N}$, so that (6.74) yields

$$f_1(s) = \operatorname*{opt}_{x\in D(1,s)} \rho(1,s,x,f_2(T(1,s,x))) \ , \ s \in S_1 \tag{6.79}$$

$$= \operatorname*{opt}_{x\in D(1,s)} \rho(1,s,x,\beta f_1(T(1,s,x))) \tag{6.80}$$

This being so, we can set $Z = S_1$. Given that U denotes the set of all bounded real-valued functions on Z, the map A can be defined as follows:

$$(Au)(s) := \operatorname*{opt}_{x\in D(1,s)} \rho(1,s,x,\beta u(T(1,s,x))) \ , \ u \in U, s \in S_1 \tag{6.81}$$

A case in point is the problem featured in *Example 5.4.1.* We shall come back to this issue later in this chapter, in our analysis of STATIONARY MODELS in *Section 6.7.*

Another point to note is that throughout this discussion, the assumption is that opt = max. Confining the investigation to this case will enable to maintain clarity of presentation without compromising on matters of substance. For opt = min, simply multiply the objective function by -1 while preserving opt = max.

Also, throughout the analysis I shall employ the following metric, denoted Θ, on U:

$$\Theta(u,v) := \sup_{z\in Z} |u(z) - v(z)| \ , \ u,v \in U \tag{6.82}$$

Observe that (U,Θ) is a *complete metric space* (see *Lemma A.2.2* in *Appendix A*). And I shall assume that the map A has the following attractive *contraction* property:

Assumption 6.3.1

$$\Theta(Au,Av) \leq \Theta(u,v) \ , \ \forall u,v \in U \tag{6.83}$$

On the strength of this assumption, *Lemma A.2.3* yields the following.

Theorem 6.3.1 *If Assumption 6.3.1 holds and the sequence $\{u^{(m)}\}$ generated by Procedure 6.2.1 converges to some $u'' \in U$, then $u'' = Au''$.*

PROOF. Suppose that the above conditions hold. Now, consider the sequence $\{v^{(m)}\}$ where $v^{(m)} := Au^{(m)}$, $m = 0, 1, 2, 3, \ldots$ Since (U, Θ) is a complete metric space and the modulus of A is equal to $\alpha = 1$, it follows from *Lemma A.2.3* that the sequence $\{v^{(m)}\}$ converges to some $v'' \in U$ such that $v'' = Av''$. Thus, because by construction $v^{(m)} = Au^{(m)} = u^{(m+1)}$, it follows that $v'' = u''$. Hence, $u'' = Au''$. □

Note that all the nontruncated dynamic programming functional equations that we shall encounter in this book satisfy *Assumption 6.3.1,* and hence *Theorem 6.3.1.* And finally, a brief explanation of the strategy I shall pursue in the investigation of the equation $u = Au$.

As we shall shortly see, this equation's behavior is essentially determined by two types of conditions. Under the first condition it will be referred to as a TYPE ONE EQUATION and under the second as a TYPE TWO EQUATION.

In both cases, I shall proceed in the following fashion. First, I shall conduct a general analysis of the equation $u = Au$, general in the sense that, other than allowing U to be a set of real-valued functions on some set Z, both A and U will be abstracted from any specific context. Then, the results obtained from this analysis will be translated back into the context of a nontruncated dynamic programming functional equation.

6.4 Functional Equations of Type One

Let us consider the following condition which, it should be noted, is stronger than the one required by *Assumption 6.3.1,* observing that here $\alpha < 1$.

Assumption 6.4.1

$$\Theta(Au, Av) \leq \alpha\Theta(u, v) \ , \ \forall u, v \in U \textit{ for some } \alpha \in [0, 1) \tag{6.84}$$

Functional equations satisfying this assumption will henceforth be referred to as TYPE ONE EQUATIONS. In the framework of this assumption *Theorem A.3.1* states the following:

Theorem 6.4.1 *(Fixed Point Theorem)*
If $u = Au$ is a Type One equation then it has a unique solution in U and any sequence generated by Procedure 6.2.1 converges to this solution.

The following example illustrates this theorem.

6.4.1 Example

Recall that the functional equation induced by the problem featured in *Example 5.4.1* was of the following form:

$$f_n(s) = \max_{0\le x\le s} \left\{\beta^{n-1}x^{1/2} + f_{n+1}(s-x)\right\} \ , \ n \in \mathbb{N}, 0 \le s \le r \tag{6.85}$$

On showing that $f_n = \beta^{n-1} f_1$ we deduced that

$$f_1(s) = \max_{0\le x\le s} \left\{x^{1/2} + \beta f_1(s-x)\right\} \ , \ 0 \le s \le r \tag{6.86}$$

to obtain the following solution to the associated equation $u = Au$:

$$u''(s) = \left[\frac{s}{1-\beta^2}\right]^{1/2} \ , \ 0 \le s \le r \tag{6.87}$$

In view of the condition laid down by *Theorem 6.4.1,* to establish that $f_1 = u''$ we need to show that the functional equation (6.86) satisfies *Assumption 6.4.1.* So, let U denote the set of all bounded real-valued functions on $Z = S = [0, r]$ and define the map A as follows:

$$(Au)(s) := \sup_{0\le x\le s} \{x^{1/2} + \beta u(s-x)\} \ , \ 0 \le s \le r \tag{6.88}$$

Note that max is substituted by sup to underscore that u can be any bounded real-valued function on S and that, under these terms, there is no assurance that a maximum is attainable. The replacement itself is a straightforward matter, indeed a mere technicality. The definition of A entails that

$$\Theta(Au, Av) = \sup_{0\le s\le r} \left| \sup_{0\le x\le s} \{x^{1/2} + \beta u(s-x)\} - \sup_{0\le x\le s} \{x^{1/2} + \beta v(s-x)\} \right| \tag{6.89}$$

which, in conjunction with *Lemma A.1.12,* yield

$$\Theta(Au, Av) \le \sup_{0\le s\le r} \left\{ \sup_{0\le x\le s} \left(\left|\{x^{1/2} + \beta u(s-x)\}\right| - \left|\{x^{1/2} + \beta v(s-x)\}\right| \right) \right\} \tag{6.90}$$

$$\le \sup_{0\le s\le r} \left\{ \sup_{0\le x\le s} |\beta u(s-x) - \beta v(s-x)| \right\} \tag{6.91}$$

$$= \sup_{0\le x\le s} \left\{ |\beta \left[u(s-x) - v(s-x)\right]| \right\} \tag{6.92}$$

$$= \sup_{0\le z\le r} \left\{ |\beta \left[u(z) - v(z)\right]| \right\} \ , \ z = s - x \tag{6.93}$$

$$= \sup_{0\le z\le r} |\beta| \left| \left[u(z) - v(z)\right] \right| \tag{6.94}$$

$$= \beta \sup_{0\le z\le r} \left| \left[u(z) - v(z)\right] \right| \tag{6.95}$$

$$= \beta\Theta(u, v) \tag{6.96}$$

But because $0 \leq \beta < 1$, it follows that $\Theta(Au, Av) < \Theta(u, v)$. Hence, *Assumption 6.4.1* holds with $\alpha = \beta$. This means that $f_1 = u''$ is a unique solution to the functional equation in question, so that (5.154) yields

$$f_n(s) = \beta^{n-1} f_1(s) \ , \ n \in \mathbb{N}, 0 \leq s \leq r \tag{6.97}$$

$$= \beta^{n-1} \left[\frac{s}{1-\beta^2} \right]^{1/2} \tag{6.98}$$

Theorem 6.4.1 can now be rephrased as follows.

Theorem 6.4.2 *Suppose that the functions $\{f_n\}$ defined by (6.2)-(6.3) satisfy the dynamic programming functional equation*

$$f_n(s) = \max_{x \in D(n,s)} \rho(n, s, x, f_{n+1}(T(n, s, x))), n \in \mathbb{N} \ , \ s \in S_n \tag{6.99}$$

Next define

$$Z := \{(n, s) : n \in \mathbb{N}, s \in S_n\} \tag{6.100}$$

$$f^*(n, s) := f_n(s), (n, s) \in Z \tag{6.101}$$

and assume that there exists a $\alpha \in [0, 1)$ such that

$$|\rho(n, s, x, a) - \rho(n, s, x, b)| \leq \alpha |a - b| \ , \forall (n, s) \in Z, a, b \in \mathbb{R} \tag{6.102}$$

Now, let $u^{(0)}$ be any bounded real-valued function on Z and consider the sequence of real valued functions $\{u^{(m)}\}$, where

$$u^{(m+1)}(n, s) := \sup_{x \in D(n,s)} \rho(n, s, x, u^{(m)}(n+1, T(n, s, x))) \tag{6.103}$$

for all $(n, s) \in Z$.

Then the sequence $\{u^{(m)}\}$ converges to the function f^* defined above in (6.76).

Proof. Considering the claim made by *Theorem 6.4.1,* it is sufficient to show that the conditions laid down by *Theorem 6.4.2* uphold *Assumption 6.4.1.* Let then U denote the set of all bounded real-valued functions on Z and let A be a map from U into itself defined as follows:

$$(Au)(n, s) := \sup_{x \in D(n,s)} \rho(n, s, x, u(n+1, T(n, s, x))) \ , \ (n, s) \in Z \tag{6.104}$$

As by construction,

$$\Theta(Au, Av) = \sup_{(n,s) \in Z} |(Au)(n, s) - (Av)(n, s)| \ , \ u, v \in U \tag{6.105}$$

$$= \sup_{(n,s) \in Z} b(u, v, n, s) \tag{6.106}$$

where

$$b(u,v,n,s) := \left|(Au)(n,s) - (Av)(n,s)\right| \ , \ u,v \in U, (n,s) \in Z \tag{6.107}$$

Lemma A.1.12, in conjunction with (6.104) and (6.107), imply that

$$b(u,v,n,s) = \sup_{x \in D(n,s)} \big|\rho(n,s,x,u(n+1,s')) - \rho(n,s,x,v(n+1,s'))\big| \tag{6.108}$$

where $s' = T(n,s,x)$, and therefore (6.102) yields

$$\Theta(Au,Av) \leq \alpha \sup_{(n,s)\in Z} \left\{ \sup_{x\in D(n,s)} \big|u(n+1,s') - v(n+1,s')\big| \right\} \tag{6.109}$$

$$\leq \alpha \sup_{\substack{x \in D(n,s)\\(n,s)\in Z}} |u(n+1,T(n,s,x)) - v(n+1,T(n,s,x))| \tag{6.110}$$

Since by definition

$$S_{n+1} = \{T(n,s,x) : s \in S_n, x \in D(n,s)\} \tag{6.111}$$

$$Z = \{(n,s) : n \in \mathbb{N}, s \in S_n\} \tag{6.112}$$

it follows from (6.110) that

$$\Theta(Au,Av) \leq \alpha \sup_{\substack{s\in S_{n+1}\\ n\in\mathbb{N}}} |u(n,s) - v(n,s)| \tag{6.113}$$

$$= \alpha\Theta(u,v) \tag{6.114}$$

The implication is then that *Assumption 6.4.1* holds here. □

6.4.2 Discounted Additive Functional Equations

Consider the case where the composition scheme is of the following additive form:

$$\rho(n,s,x,a) = w(n,s,x) + a\beta(n,s,x) \ , \ a \in \mathbb{R} \tag{6.115}$$

where w and β are bounded real-valued functions on

$$NSD := \{(n,s,x) : n \in \mathbb{N}, s \in S_n, x \in D(n,s)\} \tag{6.116}$$

and

$$0 \leq \beta(n,s,x) < \alpha \leq 1 \ , \ \forall (n,s,x) \in NSD \tag{6.117}$$

Then clearly,

$$|\rho(n,s,x,a) - \rho(n,s,x,b)| = |\beta(n,s,x)(a-b)| \tag{6.118}$$

$$= \beta(n,s,x)|a-b| \tag{6.119}$$

$$< \alpha|a-b| \tag{6.120}$$

The conclusion to be drawn is that any functional equation of this variety is a *Type One Equation.* Take note, though, that when performed analytically, *Procedure 6.2.1* can be adversely affected by β not being constant over NSD. An analytic solution of (6.103) may also prove difficult should w depend on n and s.

6.5 Functional Equations of Type Two

The distinctive feature of equations falling in this group is that they involve a contraction in the *domain space* rather than in the *functional space.* In the context of dynamic programming functional equations this means that a contraction occurs in the *state space* S. This happens, for example, when the transition function T is such that the sequence of states $s_1, s_2, \dots$ converges to some $s' \in S$ which is independent of the decisions $x_1, x_2, \dots$ made.

For instance, in the case of *Example 6.2.3* we have $s' = T(n, s, x) = \beta(s - x)$ where $0 < \beta < 1$ and $0 \leq x \leq s$. It follows then that $s_{n+1} \leq \beta s_n$, hence $s_n \leq \beta^{n-1} s_1$, and therefore as $n \to \infty$ the state variable will converge to $s' = 0$ regardless of the value of the initial state s_1 and the decisions $x_1, x_2, \dots$ made.

The point is that such a contraction in the state space, in conjunction with appropriate additional conditions, induce a contraction in the return space that characterizes functional equations of the type discussed in the preceding section.

The technical details associated with generic functional equations of this type are discussed in *Appendix A.* The following example features a dynamic programming functional equation of this type.

6.5.1 Example

Let us go back to the problem discussed in *Example 6.2.3,* recalling that in this context

$$S = \mathbb{D} = S_1 = [0, r] \ , \ r \geq 0 \tag{6.121}$$

$$D(n, s) = [0, s] \tag{6.122}$$

$$T(n, s, x) = \beta(s - x) \ , \ 0 < \beta < 1 \tag{6.123}$$

$$g(s_1, x_2, x_2, \dots) = \sum_{n=1}^{\infty} x_n^{1/2} \tag{6.124}$$

Thus in view of the modified objective functions

$$g_n(s_n, x_n, x_{n+1}, \dots) = \sum_{k=n}^{\infty} x_k^{1/2} \tag{6.125}$$

and the additive composition function

$$\rho(n,s,x,a) = x^{1/2} + a \ , \ a \in \mathbb{R} \tag{6.126}$$

we deduced the following functional equation (see (6.45)):

$$f_1(s) = \operatorname*{opt}_{0\leq x\leq s} \{x^{1/2} + f_1(\beta(s-x))\} \ , \ 0 \leq s \leq r \tag{6.127}$$

Then, using *Procedure 6.2.1*, for opt = max we obtained the solution

$$u''(s) = \left[\frac{s}{1-\beta^2}\right]^{1/2} \ , \ 0 \leq s \leq r \tag{6.128}$$

As it turned out that for every $a \in \mathbb{R}$ the function

$$u_a(s) := a + u''(s) \ , \ 0 \leq s \leq r \tag{6.129}$$

is also a solution to the equation $u = Au$, where

$$(Au)(s) := \max_{0\leq x\leq s} \{x^{1/2} + u(\beta(s-x))\} \ , \ 0 \leq s \leq r \tag{6.130}$$

we concluded that it ought to be established that $f_1 = u''$.

So in line with the framework outlined in *Appendix A.4,* set $Z = S_1$, $z^* = 0$, $K = r$, $\alpha = \beta$, and $c = 0$, and define

$$\Theta'(z,z') := |zz'| \ , \ z,z' \in Z \tag{6.131}$$

Then clearly,

$$\begin{aligned} \Theta'(z,z^*) &= |z-0| & (6.132)\\ &= |z| \leq K \ , \ \forall z \in Z & (6.133) \end{aligned}$$

Now, by definition

$$\begin{aligned} Z(m) &= \{z \in Z : \Theta'(z,z^*) \leq \alpha^m K\} \ , \ m = 0,1,2,3,\ldots & (6.134)\\ &= \{z \in [0,r] : |z| \leq r\beta^m\} & (6.135)\\ &= [0, r\beta^m] & (6.136) \end{aligned}$$

Hence,

$$\begin{aligned} \Theta_{Z(m)}(Aw,w) &= \sup_{0\leq z\leq r\beta^m} \left\{ \left| \max_{0\leq x\leq z} \left\{ x^{1/2} + w(\beta(z-x)) \right\} - w(z) \right| \right\} & (6.137)\\ &= \sup_{0\leq z\leq r\beta^m} \max_{0\leq x\leq z} \left| x^{1/2} \right| & (6.138)\\ &= \sup_{0\leq z\leq r\beta^m} \left| z^{1/2} \right| & (6.139)\\ &= (r\beta^m)^{1/2} & (6.140) \end{aligned}$$

recalling that $w(z) = c$, $\forall z \in Z$, and that in this case $w(z) = 0$, $\forall z \in Z$. Therefore,

$$\sum_{m=0}^{\infty} \Theta_{Z(m)}(Aw, w) = \sum_{m=0}^{\infty} (\beta^m r)^{1/2} \tag{6.141}$$

$$= r^{1/2} \sum_{m=0}^{\infty} (\beta^{1/2})^m \tag{6.142}$$

$$= \frac{r^{1/2}}{1 - \beta^{1/2}} < \infty \tag{6.143}$$

Clearly, the equation under consideration here is a functional equation of *Type Two*. Since the solution u'' was generated by the initial approximation $u^{(0)} = w$ it follows that u'' is the unique solution that $u = Au$ has in $U(c)$, $c = 0$. Thus, to establish that $f_1 = u''$, it is sufficient to show that f_1 is an element of $U(0)$. In other words, it is sufficient to show that f_1 is continuous at $s^* = z^* = 0$ and equal to $c = 0$ there. Observe then that, by definition,

$$f_1(s) := \max_{(x_1, x_2, \dots)} \sum_{n=1}^{\infty} x_n^{1/2} \tag{6.144}$$

$$x_n \in D(n, s_n) \ , \ n \in \mathbb{N} := \{1, 2, 3, \dots\} \tag{6.145}$$

$$s_1 = s \tag{6.146}$$

$$s_{n+1} = T(n, s_n, x_n) \ , \ n \in \mathbb{N} \tag{6.147}$$

Because $D(n, s) = [0, s]$ and $T(n, s, x) = \beta(s - x)$, the only feasible solution to this problem for $s = 0$ is the sequence $y = (0, 0, 0, \dots)$. Therefore, it is clear that $f_1(0) = 0$. This means that we have to demonstrate that f_1 is continuous at $s^* = 0$. Note then that

$$f_1(s) \geq 0 \ , \ \forall s \in S_1 \tag{6.148}$$

and that $x_n \in D(n, s_n)$ implies that

$$0 \leq x_n \leq s_n \ , \ \forall n \in \mathbb{N} \tag{6.149}$$

Hence,

$$0 \leq \sum_{n=1}^{\infty} x_n^{1/2} \leq \sum_{n=1}^{\infty} s_n^{1/2} \tag{6.150}$$

Let us then determine the largest feasible value that s_n can take, the initial state being s_1. Since

$$s_2 = T(1, s_1, x_1) = \beta(s_1 - x_1) \tag{6.151}$$

and

$$x_1 \in D(1, s_1) = [0, s_1] \tag{6.152}$$

it follows that

$$0 \leq s_2 \leq \beta s_1 \tag{6.153}$$

and

$$x_2 \in D(2, s_2) = [0, s_2] \tag{6.154}$$

so that

$$0 \leq s_3 \leq \beta^2 s_1 \tag{6.155}$$

and by induction

$$0 \leq s_n \leq \beta^{n-1} s_1 \ , \ \forall n \in \mathbb{N} \tag{6.156}$$

Hence,

$$\begin{align} 0 \leq f_1(s_1) &\leq \sum_{n=1}^{\infty} s_n^{1/2} \tag{6.157}\\ &\leq \sum_{n=1}^{\infty} (\beta^{n-1} s_1)^{1/2} \tag{6.158}\\ &= s_1^{1/2} \sum_{n=1}^{\infty} (\beta^{n-1})^{1/2} \tag{6.159}\\ &= s_1^{1/2} \sum_{n=0}^{\infty} (\beta^{1/2})^n \tag{6.160}\\ &= \left[\frac{s_1}{1-\beta^{1/2}}\right]^{1/2} \tag{6.161} \end{align}$$

Given that the upper bound of $f_1(s)$ converges to its lower bound as s tends to zero, it follows that f_1 is continuous at $s_1 = 0$. Because this entails that $f_1 \in U(c)$, $c = 0$, we conclude that

$$f_1(s) = \left[\frac{s}{1-\beta^{1/2}}\right]^{1/2} \ , \ s \in S_1 \tag{6.162}$$

is the sought solution. Therefore, (6.44) entails that

$$\begin{align} f_n(s) &= f_1(s) \ , \ n \in \mathbb{N}, s \in S_n \tag{6.163}\\ &= \left[\frac{s}{1-\beta^{1/2}}\right]^{1/2} \tag{6.164} \end{align}$$

It should be observed that it is plainly no coincidence that $Z(n) = S_n$ here. □

The following theorem summarizes the discussion on functional equations of Type Two.

Theorem 6.5.1 *Consider the case of the stationary dynamic programming functional equation*

$$f(s) = \max_{x\in D(s)} \rho(s,x,f(T(s,x))) \ , \ \forall s \in S \tag{6.165}$$

and assume that the following conditions hold:

1. *There exist a metric Θ'' on S, a state $s^* \in S$ and a constant $K \geq 0$ such that*

$$\Theta''(s,s^*) \leq K < \infty \ , \ \forall s \in S \tag{6.166}$$

2. *There exists an $\alpha \in [0,1)$ such that*

$$\Theta''(T(s,x),s^*) \leq \alpha\Theta''(s,s^*) \ , \ \forall s \in S, x \in D(s,x) \tag{6.167}$$

3. *Let (s,x) be any pair such that $s \in S$ and $x \in D(s)$. Then,*

$$|\rho(s,x,a) - \rho(s,x,b)| \leq |a-b| \ , \ \forall a,b \in \mathbb{R} \tag{6.168}$$

4. *Define,*

$$S(m) := \{s \in S : \Theta''(s,s^*) \leq \alpha K\} \ , \ m = 0,1,2,3,\ldots \tag{6.169}$$

Then,

$$\sum_{n=0}^{\infty} \sup_{s\in S(m)} \left\{ \sup_{x\in D(s)} |\rho(s,x,c) - c| \right\} < \infty \ , \ c = f(s) \tag{6.170}$$

5. *f is continuous at $s = s^*$.*

Now let U denote the set of all the bounded real-valued functions on S and let A be the map from U into itself defined as follows:

$$(Au)(s) := \sup_{x\in D(s)} \rho(s,x,u(T(s,x))) \ , \ s \in S \tag{6.171}$$

Then,

1. *f is the unique solution of $u = Au$ in $U(c)$, where $U(c)$ denotes the set of all the bounded real-valued functions on S that are continuous at $s = s^*$ and equal to c there.*
2. *If $u^{(0)}$ is an element of $U(c)$ then the sequence $\{u^{(m)}\}$ generated by Procedure 6.2.1 converges to the function f.*

PROOF. Set $Z = S$ and $z^* = s^*$. Then clearly (6.166) implies that *Assumption 6.4.2* holds true for $\Theta' = \Theta''$. Thus, to prove the theorem, it is sufficient to show that *Assumption 6.4.3* is also true. Let then s be any element of S, let x be any element of $D(s)$, let (u,v) be any element of $U \times U$, and set $s' = T(s,x)$. Since (6.168) implies that

$$|\rho(s,x,u(s')) - \rho(s,x,v(s'))| \leq |u(s') - v(s')| \tag{6.172}$$

it follows that

$$\sup_{x\in D(s)} |\rho(s,x,u(s'))| - \rho(s,x,v(s'))| \leq \sup_{x\in D(s)} |u(s') - v(s')| \tag{6.173}$$

so that an appeal to *Lemma A.4.1* yields

$$|(Au)(s) - (Av)(s)| \leq \sup_{x\in D(s)} |u(T(s,x)) - v(T(s,x))| \tag{6.174}$$

This implies that

$$\Theta_Y(Au, Av) \leq \sup_{s\in Y} \left\{ \sup_{x\in D(s)} |u(T(s,x)) - v(T(s,x))| \right\} \tag{6.175}$$

for any subset Y of S. In particular,

$$\Theta_{S(m)}(Au, Av) \leq \sup_{s\in S(m)} \left\{ \sup_{x\in D(s)} |u(T(s,x)) - v(T(s,x))| \right\} \tag{6.176}$$

for all $m \geq 0$. Therefore,

$$\Theta_{S(m)}(Au, Av) \leq \sup_{\substack{x\in D(s)\\ s\in S(m)}} |u(T(s,x)) - v(T(s,x))| \tag{6.177}$$

$$= \sup_{s'\in S'(m)} |u(s') - v(s')| \tag{6.178}$$

where

$$S'(m) := \{s' \in S(m) : s' = T(s,x), x \in D(s)\} \tag{6.179}$$

Since (6.167), in conjunction with (6.169), entail that

$$S'(m) \subseteq S(m+1) \ , \ \forall m \geq 0 \tag{6.180}$$

it follows from (6.178) that

$$\Theta_{S(m)}(Au, Av) \leq \Theta_{S(m+1)}(u, v) \tag{6.181}$$

Thus, (A.52) is true. What remains to be shown is that (A.57) is true. Observe then that since

$$\sup_{x\in D(s)} |\rho(s,x,c) - c| \leq \left| \sup_{x\in D(s)} \rho(s,x,c) - c \right| \tag{6.182}$$

it follows that

$$\sum_{m=0}^{\infty} \left\{ \sup_{s\in S(m)} |(Aw)(s) - w(s)| \right\} \leq \sum_{m=0}^{\infty} \sup_{s\in S(m)} \left\{ \sup_{x\in D(s)} |\rho(s,x,c) - c| \right\} \tag{6.183}$$

recalling that $w(s) = c$, $\forall s \in S$. Hence,

$$\sum_{m=0}^{\infty} \Theta_{S(m)}(Aw, w) \leq \sum_{m=0}^{\infty} \sup_{s \in S(m)} \left\{ \sup_{x \in D(s)} |\rho(s, x, c) - c| \right\} \tag{6.184}$$

so that (6.170) implies that

$$\sum_{m=0}^{\infty} \Theta_{S(m)}(Aw, w) < \infty \tag{6.185}$$

Since by construction $Z(m) = S(m)$, it follows that (A.57) is true, or in other words, that the functional equation under consideration is a *Type Two* Equation. This means that this equation has a unique solution in $U(c)$. Since f is continuous at $s^* = s$ and $f(s^*) = c$, it follows that f is the only solution to $u = Au$ in $U(c)$.

Moreover, if $u^{(0)} \in U(c)$, then the sequence $\{u^{(m)}\}$ generated by *Procedure 6.2.1* will converge to f. □

6.6 Truncation Method

In this section I describe the main points of an alternative solution method for nontruncated dynamic programming functional equations. As implied by the title — truncation — the method's basic proposition is to solve a nontruncated problem by treating it as though one were solving a sequence of truncated problems so as to generate a solution at the "limit".

The proposition to approach nontruncated problems in this fashion is motivated by the idea that the objective function g of a nontruncated multistage decision problem can in fact be explained as being the limit of a sequence of truncated objective functions $\{g^{(m)}\}$ where $g^{(m)}$ is a real-valued function on $S \times \mathbb{D}^m$. And to illustrate, consider the following additive nontruncated objective function

$$g(s_1, x_1, x_2, \dots) = \sum_{n=1}^{\infty} w_n(s_n, x_n) \tag{6.186}$$

where $s_2 = T(1, s_1, x_1)$, $s_3 = T(2, s_2, x_2)$, etc. Because by definition

$$\sum_{n=1}^{\infty} w_n(s_n, x_n) = \lim_{N \to \infty} \sum_{n=1}^{N} w_n(s_n, x_n) \tag{6.187}$$

we can write

$$g^{(m)}(s_1, x_1, x_2, \dots, x_m) := \sum_{n=1}^{m} w_n(s_n, x_n) \ , \ m = 1, 2, 3, \dots \tag{6.188}$$

whereupon, by construction

$$g(s_1, x_1, x_2, \dots) = \lim_{m\to\infty} g^{(m)}(s_1, x_1, x_2, \dots, x_m) \tag{6.189}$$

Thus, for each m, $1 \le m < \infty$, the function $g^{(m)}$ is taken to be a truncated objective function, to wit an objective function associated with a truncated multistage decision process consisting of m stages.

In this vein, the truncation method argues as follows. Consider the non-truncated multistage decision problem defined thus:

Problem $P(s),\ s \in S_1$:

$$f(s) := \operatorname*{opt}_{(x_1,x_2,\dots)} g(s, x_1, x_2, \dots) \tag{6.190}$$

$$x_n \in D(n, s_n)\ ,\ n \in \mathbb{N} := \{1, 2, 3, \dots\} \tag{6.191}$$

$$s_1 = s \tag{6.192}$$

$$s_{n+1} = T(n, s_n, x_n)\ ,\ n \in \mathbb{N} \qquad \square \tag{6.193}$$

Suppose that the objective function g is the limit of a sequence of functions $\{g^{(m)}\}$ such that $g^{(m)} = g^{(m)}(s, x_1, x_2, \dots, x_m)$, and *Problem P(s)* is treated as the "limit" of the following sequence of problems:

Problem $P^{(m)}(s),\ s \in S_1,\ m \in \mathbb{N}$:

$$f^{(m)}(s) := \operatorname*{opt}_{(x_1,x_2,\dots,x_m)} g^{(m)}(s, x_1, x_2, \dots, x_m) \tag{6.194}$$

$$x_n \in D(n, s_n)\ ,\ n = 1, 2, 3, \dots, m \tag{6.195}$$

$$s_1 = s \tag{6.196}$$

$$s_{n+1} = T(n, s_n, x_n) \qquad \square \tag{6.197}$$

Then, if

$$f(s) = \lim_{m\to\infty} f^{(m)}(s)\ ,\ \forall s \in S_1 \tag{6.198}$$

one would solve *Problem P(s)* by solving the sequence of truncated problems $\{$*Problem* $P^{(m)}(s) : m \in \mathbb{N},\ s \in S_m\}$.

Because the validity of this argument is contingent on (6.198), we need to determine under what conditions (6.198) holds — assuming that (6.189) is valid.

Observe then that by definition, if $s \in S_1$ and $(x_1^*, x_2^*, \dots)$ is an optimal solution for *Problem* $P(s)$ then

$$g(s, x_1^*, x_2^*, \dots) = f(s) \tag{6.199}$$

Now, assume that opt = max. Because for each $m \in \mathbb{N}$ the sub-sequence $(x_1^*, x_2^*, \dots, x_m^*)$ is a feasible — but not necessarily optimal — solution to *Problem* $P^{(m)}(s)$, it follows that

$$f^{(m)}(s) \ge g^{(m)}(s, x_1^*, x_2^*, \dots)\ ,\ \forall m \in \mathbb{N} \tag{6.200}$$

and therefore

$$\lim_{m\to\infty} f^{(m)}(s) \geq \lim_{m\to\infty} g^{(m)}(s, x_1^*, x_2^*, \dots) \tag{6.201}$$

provided that these limits exist. Thus, if (6.189) is true, then (6.201) yields

$$\lim_{m\to\infty} f^{(m)}(s) \geq f(s) \ , \ \forall s \in S_1 \tag{6.202}$$

From this it follows that for (6.198) to also be true, it is sufficient that

$$f^{(m)}(s) \leq f(s) \ , \ \forall m \in \mathbb{N}, s \in S_1 \tag{6.203}$$

Notice then that this is indeed the case if

$$g^{(m)}(s, x_1, x_2, \dots, x_m) \leq g(s, x_1, x_2, \dots) \tag{6.204}$$

for all $(x_1, x_2, \dots) \in X(s)$ and $m \in \mathbb{N}$, where $X(s)$ denotes the set of feasible solutions for *Problem P(s).*

For example, the additive objective function defined by (6.186) can be stated thus:

$$\begin{aligned} g(s_1, x_1, x_2, \dots) &= \sum_{n=1}^{\infty} w_n(s_n, x_n) && (6.205) \\ &= \sum_{n=1}^{m} w_n(s_n, x_n) + \sum_{n=m+1}^{\infty} w_n(s_n, x_n) && (6.206) \\ &= g^{(m)}(s_1, x_1, x_2, \dots, x_m) + \sum_{n=m+1}^{\infty} w_n(s_n, x_n) && (6.207) \end{aligned}$$

where $g^{(m)}$ is the function defined by (6.188).

Hence, it is clear that (6.204) — and therefore (6.198) — hold if the functions $\{w_n\}$ are *non-negative,* namely if $w_n(s, x) \geq 0$, $\forall n \in \mathbb{N}$, $s \in S_n$, $x \in D(n, s)$.

The following example illustrates the truncation method in action.

6.6.1 Example

Consider the truncated problem studied in *Example 5.3.3.* You will recall that it was formulated as follows:

$$\begin{aligned} p := \max_{(x_1,\dots,x_N)} & \sum_{n=1}^{N} \beta^{n-1} x_n^{1/2} \ , \ 0 < \beta < 1 && (6.208) \\ & \sum_{n=1}^{N} x_n \leq r \ , \ r > 0 && (6.209) \\ & x_n \geq 0 \ , \ n = 1, 2, 3, \dots, N && (6.210) \end{aligned}$$

The nontruncated version of this problem is then

$$p' := \max_{(x_1,x_2,\dots)} \sum_{n=1}^{\infty} \beta^{n-1} x_n^{1/2} \tag{6.211}$$

$$\sum_{n=1}^{\infty} x_n \leq r \ , \ r > 0 \tag{6.212}$$

$$x_n \geq 0 \ , \ n = 1, 2, 3, \dots \tag{6.213}$$

We can therefore set

$$S = S_1 = \mathbb{D} = [0, r] \tag{6.214}$$

$$D(n,s) = [0, s] \ , \ n \in \mathbb{N}, s \in S \tag{6.215}$$

$$T(n,s,x) = s - x \ , \ n \in \mathbb{N}, s \in S, x \in D(n,s) \tag{6.216}$$

$$g(s_1, x_1, x_2, \dots) = \sum_{n=1}^{\infty} \beta^{n-1} x_n^{1/2} \tag{6.217}$$

Observe that because $1 > \beta > 0$, it is clear that g is bounded above. Next, we truncate g as follows:

$$g^{(m)}(s_1, x_1, x_2, \dots, x_m) = \sum_{n=1}^{m} \beta^{n-1} x_n^{1/2} \quad , \ m \in N \tag{6.218}$$

Clearly, (6.189) holds in this case. Furthermore, as opt = max and the functions

$$w_n(s,x) := \beta^{n-1} x_n^{1/2} \tag{6.219}$$

are non-negative, it is obvious that the problem concerned satisfies (6.204) and therefore (6.198).

Thus, since *Problem* $P^{(m)}(s)$ is identical to the truncated problem in *Example 5.3.3* (with $N = m$), we conclude that

$$f^{(m)}(s) = \left[\frac{s(1-\beta^{2m})}{1-\beta^2}\right]^{1/2} \ , \ m \in \mathbb{N}, 0 \leq s \leq r \tag{6.220}$$

Thus,

$$f(s) = \lim_{m\to\infty} f^{(m)}(s) \ , \ 0 \leq s \leq r \tag{6.221}$$

$$= \lim_{m\to\infty} \left[\frac{s(1-\beta^{2m})}{1-\beta^2}\right]^{1/2} \tag{6.222}$$

$$= \left[\frac{s}{1-\beta^2}\right]^{1/2} \tag{6.223}$$

In short,

$$p' = f(r) = \left[\frac{r}{1-\beta^2}\right]^{1/2} \tag{6.224}$$

This result accords with the one obtained in *Example 6.4.1* by the successive approximations procedure. □

In *Section 6.7* I examine the relation between the truncation method and the successive approximation method. But before I can do this, I need to introduce the concept *Stationary* into the discussion. This will allow a significant simplification of the notation in the discussion that follows.

6.7 Stationary Models

Any object in a multistage decision model that is insensitive to the stage variable n is said to be *stationary.* For example, the transition function

$$T(n, s, x) = x - s \tag{6.225}$$

and the composition function

$$\rho(n, s, x, a) = \alpha(s - x)^2 + a \tag{6.226}$$

are stationary. Similarly, the Markovian policy δ defined thus

$$\delta(n, s) = s(1 - \beta)\ ,\ \ s \in S := [0, r] \tag{6.227}$$

is stationary.

If all the elements that are distinctive to a dynamic programming model are stationary, the model as a whole is rendered stationary. To set the scene for the definition of a stationary nontruncated dynamic programming model, let us first consider the definition of a stationary nontruncated multistage decision model.

Definition 6.7.1 A STATIONARY *nontruncated multistage decision model is a collection* (S, D, T, g) *where*

1. *S is a nonempty set called the* STATE SPACE.
2. *D is a function on S such that to each $s \in S$ it assigns a nonempty subset of some set $\mathbb{D}$. We refer to $D(s)$ as the* SET OF FEASIBLE DECISIONS *pertaining to state s. The set $\mathbb{D}$ is called the* DECISION SPACE.
3. *T is a function on $S \times \mathbb{D}$ with values in S called the* TRANSITION FUNCTION.
4. *g is a real-valued function on $S \times \mathbb{D}^\infty$ called the* OBJECTIVE FUNCTION.

With the above definition as background, consider now the following family of optimization problems.

Problem $P(s), s \in S_1$:

$$f(s) := \underset{(x_1,x_2,\dots)}{\text{opt}} \; g(s, x_1, x_2, \dots) \tag{6.228}$$

$$x_n \in D(s_n) \ , \ n \in \mathbb{N} := \{1, 2, 3, \dots\} \tag{6.229}$$

$$s_1 = s \tag{6.230}$$

$$s_{n+1} = T(s_n, x_n) \ , \ n \in \mathbb{N} \tag{6.231}$$

For each $s \in S$, let $X(s)$ denote the set of feasible solutions for *Problem P(s),* and let $X^*(s)$ denote the set of optimal solutions for *Problem P(s).* □

The definition of a stationary nontruncated dynamic programming model would therefore read as follows:

Definition 6.7.2 *A* STATIONARY *nontruncated dynamic programming model is a collection* (S, D, T, g, ρ) *such that*

1. (S, D, T, g) *is a* STATIONARY *multistage decision model.*
2. ρ *is a real-valued function on* $S \times \mathbb{D} \times \mathbb{R}$*, called the* COMPOSITION FUNCTION*, satisfying the following condition:*

$$g(s, x, z) = \rho(s, x, g(T(s, x), z)) \ , \ \forall s \in S, x \in D(s), z \in X(s) \tag{6.232}$$

3. *The function* f *defined by (6.228)-(6.231) satisfies the equation*

$$f(s) = \underset{x \in D(s)}{opt} \; \rho(s, x, f(T(s, x))) \ , \ \forall s \in S \tag{6.233}$$

In a word, a nontruncated dynamic programming model is rendered stationary by D, T and ρ being stationary and all its modified objective functions being *identical.*

6.7.1 Example

Consider the problem

$$p := \max_{(x_1,x_2,\dots)} \sum_{n=1}^{\infty} \beta^{n-1} v(x_n) \ , \ 0 < \beta < 1 \tag{6.234}$$

$$\sum_{n=1}^{\infty} x_n \leq r \ , \ r > 0 \tag{6.235}$$

$$x_n \geq 0 \ , \ n \in \mathbb{N} := \{1, 2, 3, \dots\} \tag{6.236}$$

where v is a real-valued function on $[0, r]$ such that $|v(x)| < b$ for some $b < \infty$. To frame a stationary dynamic programming model for this problem set

$$S = \mathbb{D} = [0, r] \tag{6.237}$$

$$D(s) = [0, s] \ , \ \ s \in S \tag{6.238}$$

$$T(s, x) = s - x \ , \ \ s \in S, x \in D(s) \tag{6.239}$$

$$g(s_1, x_1, x_2, \dots) = \sum_{n=1}^{\infty} \beta^{n-1} v(x_n) \tag{6.240}$$

By construction then,

$$g(s_1, x_1, x_2, \dots) = \beta^0 v(x_1) + \sum_{n=2}^{\infty} \beta^{n-1} v(x_n) \tag{6.241}$$

$$= v(x_1) + \beta \sum_{n=2}^{\infty} \beta^{n-2} v(x_n) \tag{6.242}$$

$$= v(x_1) + \beta \sum_{n=1}^{\infty} \beta^{n-1} v(x_{n+1}) \tag{6.243}$$

$$= v(x_1) + \beta g(s', x_2, x_3, \dots) \ , \ \ s' = T(s_1, x_1) \tag{6.244}$$

Thus, if we set

$$\rho(s, x, a) = v(x) + \beta a \ , \ \ s \in S, x \in D(s), a \in \mathbb{R} \tag{6.245}$$

then ρ satisfies (6.232). Finally, to show that the functional equation (6.233) is valid, note that by definition

$$X(s) := \{(x, z) : x \in D(s), z \in X(T(s, x))\} \tag{6.246}$$

so that

$$f(s) = \underset{(x,z) \in X(s)}{\text{opt}} \ g(s, x, z) \tag{6.247}$$

$$= \underset{\substack{x \in D(s) \\ z \in X(s') \\ s' = T(s,x)}}{\text{opt}} \ g(s, x, z) \tag{6.248}$$

Hence, applying the *Principle of Conditional Optimization* to (6.248) yields the following:

$$f(s) = \underset{x \in D(s)}{\text{opt}} \left\{ \underset{\substack{s' = T(s,x) \\ z \in X(s')}}{\text{opt}} \ g(s, x, z) \right\} \tag{6.249}$$

$$= \underset{x \in D(s)}{\text{opt}} \left\{ \underset{\substack{s' = T(s,x) \\ z \in X(s')}}{\text{opt}} \ v(x) + \beta g(s', z) \right\} \ , \ \ 0 < \beta < 1 \tag{6.250}$$

$$= \operatorname*{opt}_{x\in D(s)} \Big\{ v(x) + \operatorname*{opt}_{\substack{s'=T(s,x)\\ z\in X(s')}} \beta g(s',z) \Big\} \tag{6.251}$$

$$= \operatorname*{opt}_{x\in D(s)} \Big\{ v(x) + \beta \operatorname*{opt}_{\substack{s'=T(s,x)\\ z\in X(s')}} g(s',z) \Big\} \tag{6.252}$$

$$= \operatorname*{opt}_{x\in D(s)} \Big\{ v(x) + \beta f(s') \Big\} \ , \ s' = T(s,x) \tag{6.253}$$

$$= \operatorname*{opt}_{x\in D(s)} \Big\{ v(x) + \beta f(T(s,x)) \Big\} \qquad \square \tag{6.254}$$

I remind the reader that the property *stationary* applies to an object only insofar as it is a component of the model, and to the model itself. But it cannot be said to characterize the *problem* as such, as the optimization problem is neither stationary nor nonstationary. Indeed, the same problem can assume formulations that are stationary and others that are nonstationary.

So, when we say that a problem is *stationary/nonstationary* we mean that the problem will lend itself to formulation in terms of a *stationary/nonstationary* model. The reader may wish to ascertain that the trivial dynamic programming model in *Lemma 4.5.1* can be cast as a stationary model.

And before winding up this section I also call attention to these points. First, the above definitions can be modified so as to encompass truncated models as well. I put off the examination of stationary truncated models to *Chapter 10.*

Second, often a dynamic programming model is only partially stationary in that certain, but not all, of its constituent elements are stationary. In this case the definition of the object concerned can be modified accordingly. To be precise, as the stage variable n is irrelevant from the standpoint of a stationary object, it can be omitted from the object's formulation altogether. For example, instead of writing

$$D(n,s) = [0,s] \ , \ 0 \le s \le r, n \in \mathbb{N} \tag{6.255}$$

we can write

$$D(s) = [0,s] \ , \ 0 \le s \le r \tag{6.256}$$

in which case D would be a function on S rather than on $\mathbb{N} \times S$.

By the same token, should an object prove to be insensitive to the state variable s it will also be expedient to omit the latter from the definition of the object concerned. Thus, I shall write

$$\rho(x,a) = x^{1/2} + a \ , \ 0 \le x, a \in \mathbb{R} \tag{6.257}$$

instead of

$$\rho(n,s,x,a) = x^{1/2} + a \ , \ 1 \le n < \mathbb{N}, s \in S_n, 0 \le x, a \in \mathbb{R} \tag{6.258}$$

and

$$g(x_1, x_2, \dots, x_N) = \sum_{n=1}^{N} x_n^{1/2} \tag{6.259}$$

instead of

$$g(s_1, x_1, x_2, \dots, x_N) = \sum_{n=1}^{N} x_n^{1/2} \tag{6.260}$$

without explicitly stating that ρ is defined on $[0, \infty) \times \mathbb{R}$ rather than on $\mathbb{N} \times S \times \mathbb{R}$ and that g is defined on $\mathbb{D}^N$ rather than on $S_1 \times \mathbb{D}^N$, etc.

6.8 Truncation and Successive Approximation

Whatever the technical and conceptual differences between the truncation and successive approximation methods, in many instances they amount to the same thing. In what follows I explain this point.

The truncation method proceeds from a sequence of modified objective functions $\{g^{(m)}\}$, each of the following form:

$$g^{(m+1)}(s, x_1, x_2, \dots, x_{m+1}) = \mu(s, x_1, g^{(m)}(s', x_2, x_3, \dots, x_{m+1})) \tag{6.261}$$

where μ is some real−valued function on $S \times \mathbb{D} \times \mathbb{R}$.

Furthermore, the sequence $\{f^{(m)}\}$ defined by (6.194)-(6.197) normally satisfies the following functional equation:

$$f^{(m+1)}(s) = \operatorname*{opt}_{x \in D(s)} \mu(s, x, f^{(m)}(T(s, x))) \ , \ \forall s \in S, m \in \mathbb{N} \tag{6.262}$$

Thus, in the case of opt = max *and* the model being stationary, the truncation method is targeted on the following functional equation:

$$f^{(m+1)}(s) = \max_{x \in D(s)} \mu(s, x, f^{(m)}(T(s, x))) \ , \ m \in \mathbb{N}, s \in S \tag{6.263}$$

Turning now to the successive approximation method, in the case of stationary formulations the functional equation that the successive approximation procedure seeks to solve is of the form

$$u^{(m+1)}(s) = \sup_{x \in D(s)} \rho(s, x, u^{(m)}(T(s, x))) \ , \ m \in \mathbb{N}, s \in S \tag{6.264}$$

where $u^{(0)}$ can be any real-valued function on S, and ρ is the composition function of the pertinent decomposition scheme.

If we disregard the technicality of sup and max, collating (6.263) and

(6.264) reveals the following. In cases where the functions μ and ρ are identical, if the initial approximation $u^{(0)}$ is set so as to be identical to $f^{(1)}$, that is, if

$$u^{(0)}(s) = f^{(1)}(s) \ , \ \forall s \in S \tag{6.265}$$

then the successive approximation procedure and the truncation method come down to the same thing in the sense that

$$f^{(m+1)}(s) = u^{(m)}(s) \ , \ \forall s \in S \tag{6.266}$$

holds for all $m \geq 1$, and the functional equations (6.263)-(6.264) are essentially identical.

The relation between the two methods can be accounted for by a still more elegant explanation. Considering that by definition

$$f^{(1)}(s) := \max_{x \in D(s)} g^{(1)}(s,x) \ , \ s \in S \tag{6.267}$$

and

$$u^{(1)}(s) := \sup_{x \in D(s)} \rho(s,x,u^{(0)}(T(s,x))) \ , \ s \in S \tag{6.268}$$

to bring out the relation between the two methods we need to identify that particular initial approximation $u^{(0)}$ that would satisfy the equation

$$g^{(1)}(s,x) = \rho(s,x,u^{(0)}(T(s,x))) \ , \ \forall s \in S, x \in D(s) \tag{6.269}$$

Clearly, in this case we would have $f^{(1)}(s) = u^{(1)}(s)$, $\forall s \in S$, which if assuming that $\mu = \rho$, would entail that

$$f^{(m)}(s) = u^{(m)}(s) \ , \ \forall s \in S, m \geq 1 \tag{6.270}$$

The question is then how would we track down such an initial approximation. I shall do this through an appeal to the concept *identity element.* This concept is invoked again in *Chapter 10* in our discussion on decomposition schemes. For our present purposes a brief explanation will suffice.

If B is a set and $\oplus$ is a binary function on $B \times B$ then $a \in B$ is said to be the *identity element* of $\oplus$ if, and only if,

$$a \oplus b = b \ , \ \forall b \in B \tag{6.271}$$

$$b \oplus a = b \ , \ \forall b \in B \tag{6.272}$$

For example, the identity element of the function $+$ is zero and the identity element of the function $\times$ is one.

Going back then to where we left off, suppose that the objective function is of the following additive form:

$$g(s_1, x_1, x_2, \dots) = \sum_{n=1}^{\infty} \beta^{n-1} v(s_n, x_n) \ , \ 0 < \beta < 1 \tag{6.273}$$

Then in the case of the truncation method, the sequence of modified objective functions is of the form

$$g^{(m)}(s_1, x_1, x_2, \ldots, x_m) = \sum_{n=1}^{m} \beta^{n-1} v(s_n, x_n) \tag{6.274}$$

Thus, by construction

$$\begin{aligned}
g^{(m+1)}(s_1, x_1, x_2, \ldots, x_{m+1}) &= \sum_{n=1}^{m+1} \beta^{n-1} v(s_n, x_n) && (6.275)\\
&= v(s_1, x_1) + \sum_{n=2}^{m+1} \beta^{n-1} v(s_n, x_n) && (6.276)\\
&= v(s_1, x_1) + \beta \sum_{k=1}^{m} \beta^{k-1} v(s_{k+1}, x_{k+1}) && (6.277)\\
&= v(s_1, x_1) + \beta g^{(m)}(s_2, x_2, x_3, \ldots, x_{m+1}) && (6.278)\\
&= \mu(s_1, x_1, g^{(m)}(s_2, x_2, x_3, \ldots, x_{m+1})) && (6.279)
\end{aligned}$$

where

$$\mu(s, x, a) = v(s, x) + \beta a \ , \ \ s \in S, x \in D(s), a \in \mathbb{R} \tag{6.280}$$

On the other hand, in the case of the successive approximation method, the sequence of modified objective functions is of the following form:

$$g_n(s_n, x_n, x_{n+1}, \ldots) = \sum_{k=n}^{\infty} \beta^{k-n} v(s_k, x_k) \tag{6.281}$$

By construction then,

$$\begin{aligned}
g_n(s_n, x_n, x_{n+1}, \ldots) &= v(s_n, x_n) + \beta \sum_{k=n+1}^{\infty} \beta^{k-n} v(s_k, x_k) && (6.282)\\
&= v(s_n, x_n) + \beta \sum_{k=n+1}^{\infty} \beta^{k-(n+1)} v(s_k, x_k) && (6.283)\\
&= v(s_n, x_n) + \beta g_{n+1}(s_{n+1}, x_{n+1}, x_{n+2}, \ldots) && (6.284)\\
&= \rho(s_n, x_n, g_{n+1}(s_{n+1}, x_{n+1}, x_{n+2}, \ldots)) && (6.285)
\end{aligned}$$

where

$$\rho(s, x, a) = v(s, x) + \beta a \ , \ \ s \in S, x \in D(s), a \in \mathbb{R} \tag{6.286}$$

Now, clearly

$$\rho(s, x, a) = \mu(s, x, a) = v(s, x) \oplus \beta a \tag{6.287}$$

with $\oplus = +$.

Since the identity element of $+$ is zero, setting $u^{(0)}(s) = 0$, $\forall s \in S$ has the effect of yielding

$$\begin{aligned} u^{(1)}(s) &= \sup_{x\in D(s)} \rho(s,x,u^{(0)}(T(s,x)))\ ,\ \ s \in S && (6.288)\\ &= \sup_{x\in D(s)} \{v(s,x) + 0\} && (6.289)\\ &= \sup_{x\in D(s)} v(s,x) && (6.290)\\ &= \sup_{x\in D(s)} g^{(1)}(s,x) && (6.291)\\ &= f^{(1)}(s) && (6.292)\end{aligned}$$

Alternatively, the function $u^{(0)}$ satisfying (6.269) can be identified as follows. By construction

$$g^{(1)}(s,x) = v(s,x) \tag{6.293}$$

and

$$\rho(s,x,u^{(0)}(T(s,x))) = v(s,x) + \beta u^{(0)}(T(s,x)) \tag{6.294}$$

Thus, (6.269) reduces to

$$v(s,x) = v(s,x) + \beta u^{(0)}(T(s,x)) \tag{6.295}$$

which in turn yields

$$u^{(0)}(s) = 0\ ,\ \forall s \in S \tag{6.296}$$

In short, if the objective function relevant to the truncation procedure is additive and the initial approximation relevant to the successive approximation procedure is $u^{(0)}(s) = 0$, $\forall s \in S$, then the two procedures are identical. More generally, assume that

$$\rho(s,x,a) = \mu(s,x,a) = v(s,x) \oplus a\ ,\ \forall s \in S, x \in D(s), a \in \mathbb{R} \tag{6.297}$$

and let $I_\oplus$ denote the identity element of $\oplus$. Then, using the initial approximation

$$u^{(0)}(s) = I_\oplus\ ,\ \forall s \in S \tag{6.298}$$

the successive approximation procedure and the truncation method amount to the same thing.

It should also be noted that the identity element concept can be employed as a device for simplifying the initialization of the iterative procedure for truncated dynamic programming functional equations discussed in *Chapter 5.*

That is, if $\mathbb{N}$ is finite we can define

$$f_{N+1}(s) := I_{\oplus} \ , \ s \in S_{N+1} \tag{6.299}$$

in which case, the truncated dynamic programming functional equation can be written thus:

$$f_n(s) = \underset{x \in D(n,s)}{\text{opt}} \ \rho(n, s, x, f_{n+1}(T(n, s, x))) \ , \ 1 \leq n \leq N, s \in S_n \tag{6.300}$$

where

$$\rho(N, s, x, a) = g_N(s, x) \oplus I_{\oplus} \ , \ s \in S_N, x \in D(N, s), a \in \mathbb{R} \tag{6.301}$$

For example, if the objective function is additive, the initialization step would involve setting

$$f_{N+1}(s) = 0 \ , \ \forall s \in S_N \tag{6.302}$$

rather than solving the optimization problems

$$f_N(s) = \underset{x \in D(N,s)}{\text{opt}} \ g_N(s, x) \ , \ s \in S_N \tag{6.303}$$

This modification, of course, has no impact whatsoever on the computational or analytic side of the iterative procedures of direct methods. It does, however, make for a more uniform and thus more elegant formulation of these procedures.

6.9 Summary

We saw that the nontruncated dynamic programming functional equation is amenable to the techniques of the standard successive approximation method. We also saw that one of the most salient differences between the successive approximation method and the direct method employed to solve truncated functional equations is that the former usually necessitates proving that the equation has a unique solution.

As we shall see at a later stage, successive approximation methods can also be used to solve truncated functional equation. Indeed, one of the most famous algorithms in computer science and operations research, namely *Dijkstra's Algorithm* for the shortest path problem is a successive approximation method that can be applied to truncated and non-truncated functional equations. I examine this algorithm in *Chapter 15.*

To conclude our discussion on the mathematical idiom of dynamic programming I shall have to show that the policies generated by the solution of the dynamic programming functional equation are optimal. And, I shall have to look into the issue of the computational requirements of dynamic programming algorithms.

6.10 Bibliographic Notes

The essentials of how to use successive approximation techniques in the solution of nontruncated dynamic programming functional equations go back to Bellman's early papers and his first book on dynamic programming (1957a). Bellman's analysis was confined, however, to additive objective functions and it was not explicitly couched in the contraction mappings terminology.

The general approach to what I labeled here Type One functional equations, based in the language and results of classical functional analysis and contraction mapping, was formulated by Denardo [1965, 1968].

The treatment of the functional equation of Type Two outlined in this chapter synthesizes Bellman's treatment of the additive case and Denardo's treatment of functional equations of Type One.

Needless to say, there are several other approaches to successive approximation that can be used effectively to solve nontruncated and, it might be added, truncated dynamic programming equations. To mention but a few, Bellman [1957a, 1971], Denardo [1982], Hartley et al. [1986], Howard [1960], White [1963, 1978], Zuo and Wu [1989].

For "heuristic" successive approximation methods, that is successive approximation methods that do not aim to find exact solutions to the dynamic programming functional equation, see Heidari et al. [1971], Gal [1989], Sniedovich and Voß [2006], Powell [2007].

7

Optimal Policies

7.1 Introduction

To be sure, obtaining an optimal solution for the dynamic programming functional equation is the driving force behind the effort to obtain an optimal solution for the optimization problem considered. So, having charted the solution techniques that are available for this equation, I could have presumably deemed the analysis of dynamic programming's treatment of an optimization problem to be complete. But, we are reminded that:

> The problem is not considered completely solved in the mathematical sense until the structure of the optimal policy is understood.
>
> Bellman [1957a, p. ix]

The implication is then that a full appreciation of dynamic programming's handling of a problem requires a good grasp of the policy underlying the optimal solution of the problem. I therefore devote this chapter to an examination of the polices generated by the dynamic programming functional equation, focusing on two key issues:

· Their basic features.

· The conditions assuring their optimality.

To do this I shall draw on the results obtained in the preceding three chapters.

7.2 Preliminary Analysis

Let us begin then with a quick review of what we have seen thus far about the topic of policies. In *Section 3.3,* I described four types of policies that are suitable for the multistage decision model. Of the four types, the *Markovian* policy was singled out as the most natural for dynamic programming. I also observed that, ideally, a single Markovian policy would be optimal for all the initial problems. I demonstrated, however, that under the terms of the multistage decision model there are no a priori guarantees that such a policy always exists. I concluded therefore that added measures are necessary to secure its existence. Subsequently, discussions on the Markovian property in ensuing chapters have shed considerable light on what sort of provisions would ensure the existence of such an optimal Markovian policy.

With this as background, I begin this chapter by first updating the definition of an optimal policy. This is called for in view of the introduction into the discussion in *Chapter 4* of the notion *modified problems.*

Recall then that Δ denotes the set of Markovian policies and that $X^*(n, s)$

denotes the set of optimal solutions for the modified problem at (n, s), namely *Problem P(n,s).*

Definition 7.2.1 *Let (n, s, δ) be any triplet such that $n \in N$, $s \in S_n$ and $\delta \in \Delta$. Then δ is said to be* OPTIMAL WITH RESPECT TO *(n,s) if its application at (n, s) generates a sequence of decisions $(x_n, \ldots, x_N)$ such that $(x_n, \ldots, x_N) \in X^*(n, s)$. Similarly, a policy $\delta \in \Delta$ is said to be* OPTIMAL *if it is optimal for all pairs $\{(n, s)\}$ such that $n \in \mathbb{N}$ and $s \in S_n$. Let Δ^* denote the set of optimal Markovian policies.*

In this rendering, a Markovian policy would be optimal if it proves optimal for all the modified problems induced by the decomposition scheme under consideration.

It is important to note, therefore, that with an optimal policy in hand, working out the optimal solution is a straightforward matter. This is so because the sequence of decisions $(x_n, \ldots, x_N)$ obtained by applying a policy $\delta \in \Delta$ to a pair (n, s) is determined as follows: $x_n = \delta(n, s)$ and

$$x_{m+1} = \delta(m + 1, s_{m+1}) \ , \ n \leq m < N \tag{7.1}$$

where $s_n = s$ and

$$s_{m+1} = T(m, s_m, x_m) \ , \ n \leq m \leq N \tag{7.2}$$

Obviously, of the policies comprising the Markovian set Δ, of concern to us here are only those that are recoverable from the solution to the dynamic programming functional equation. So, henceforth I shall refer to these policies as *dynamic programming policies*. Their formal definition reads as follows.

Definition 7.2.2 *For each pair (n, s) such that $n \in \mathbb{N}$ and $s \in S_n$, let $D^\circ(n, s)$ denote the subset of $D(n, s)$ whose elements are solutions to the dynamic programming functional equation*

$$f_n(s) = \operatorname*{opt}_{x \in D(n,s)} \rho(n, s, x, f_{n+1}(T(n, s, x))) \ , \ 1 \leq n < N, s \in S_n \tag{7.3}$$

Namely, for each pair (n, s) such that $1 \leq n < N$ and $s \in S_n$ define

$$D^\circ(n, s) := \{x \in D(n, s) : \rho(n, s, x, f_{n+1}(T(n, s, x))) = f_n(s)\} \tag{7.4}$$

and if N is finite define

$$D^\circ(N, s) := \{x \in D(N, s) : g_N(s, x) = f_N(s)\} \ , \ s \in S_N \tag{7.5}$$

Now, define

$$\Delta^\circ := \{\delta \in \Delta : \delta(n, s) \in D^\circ(n, s), \forall n \in \mathbb{N}, s \in S_n\} \tag{7.6}$$

We shall refer to Δ° as the set of DYNAMIC PROGRAMMING POLICIES.

The next task is to examine the relation between the set of optimal Markovian policies and the set of dynamic programming policies. But before I do this, I need to introduce the following notation.

For any Markovian policy $\delta \in \Delta$, let $x(n,s,\delta)$ denote the sequence of decisions $(x_n,\dots,x_N)$ generated by applying δ to (n,s), as described above by (7.1)-(7.2). Also, define

$$g(s,\delta) := g(s,x(1,s,\delta)) \tag{7.7}$$

and

$$g_n(s,\delta) := g_n(s,x(n,s,\delta)) \tag{7.8}$$

Thus, the expression $g(s,\delta)$ designates the return obtained by applying the objective function g to the initial state s and the decisions generated by δ and s. Likewise, $g_n(s,\delta)$ designates the return from the modified objective function g_n, given state s at stage n, and the sequence of decisions generated by (n,s) and δ as described above.

Now, let (ρ,G) be any decomposition scheme, δ be any Markovian policy, and (n,s) any stage-state pair. Then from the definition of (ρ,G) it follows that

$$g_n(s,x_n,x_{n+1},\dots,x_N) = \rho(n,s,x_n,g_{n+1}(s',x_{n+1},x_{n+2},\dots,x_N)) \tag{7.9}$$

where $s' = T(n,s,x_n)$.

Also, since by construction

$$x(n,s,\delta) = (\delta(n,s),x(n+1,s',\delta))\ ,\ s' = T(n,s,\delta(n,s)) \tag{7.10}$$

it follows from (7.9)-(7.10) that

$$g_n(s,\delta) = \rho(n,s,\delta(n,s),g_{n+1}(s',\delta))\ ,\ s' = T(n,s,\delta(n,s)) \tag{7.11}$$

In brief, the above notation permits incorporating δ in the expression stipulating the relation between g_n and g_{n+1}. And now consider the following.

Theorem 7.2.1 *If Δ^* is not empty, then the dynamic programming functional equation is valid and $\Delta^* \subseteq \Delta^\circ$. That is, if an optimal policy exists, then any Markovian optimal policy is a dynamic programming policy.*

Proof. Assume that Δ^* is not empty and let δ be any element of Δ^*. I shall first show that this implies that the functional equation is valid, and on these grounds that $\delta \in \Delta^\circ$.

Observe then that as δ is assumed to be an optimal Markovian policy, it follows that for all $1 \le n < N$, and $s \in S_n$ we have

$$\begin{aligned} f_n(s) &= g_n(s,\delta) & (7.12)\\ &= \rho(n,s,x,g_{n+1}(T(n,s,x),\delta)) & (7.13)\\ &= \rho(n,s,x,f_{n+1}(T(n,s,x))) & (7.14)\end{aligned}$$

Now, define

$$a_n(s) := \underset{x \in D(n,s)}{opt} \rho(n,s,x,f_{n+1}(T(n,s,x))) \ , \ 1 \leq n < N, s \in S_n \tag{7.15}$$

To show that the functional equation is valid, we have to show that

$$f_n(s) = a_n(s) \ , \ \forall 1 \leq n < N, s \in S_n \tag{7.16}$$

Case 1. opt = max .

Since $\delta(n,s) \in D(n,s)$ for all $n \in N$ and $s \in S_n$, it follows from (7.14)-(7.15) that $a_n(s) \geq f_n(s)$, $\forall 1 \leq n < N$, $s \in S_n$. Now, assume that contrary to (7.16), there exists a pair (m,z) such that $1 \leq m < N$, $z \in S_m$ and $a_m(z) > f_m(z)$. This implies in turn the existence of some $x^* \in D(m,z)$ such that

$$\rho(m,z,x^*,f_{m+1}(T(m,z,x^*))) > f_m(z) = g_m(z,\delta) \tag{7.17}$$

Next, consider the policy δ^* specified as follows:

$$\delta^*(n,s) = \begin{cases} x^* & , \quad n = m, s = z \\ \delta(n,s) & , \quad \text{otherwise} \end{cases} \tag{7.18}$$

By construction then,

$$g_{n+1}(s,\delta^*) = g_{n+1}(s,\delta) \ , \ \forall s \in S_{n+1} \tag{7.19}$$
$$= f_{n+1}(s) \tag{7.20}$$

so that (7.17) entails that

$$\rho(m,z,x^*,g_{m+1}(T(m,z,x^*),\delta^*)) > g_m(z,\delta) \tag{7.21}$$

As $\delta^*(m,z) = x^*$, it follows that

$$\rho(m,z,x^*,g_{m+1}(T(m,z,x^*),\delta^*)) = g_m(z,\delta^*) \tag{7.22}$$

so that (7.21)-(7.22) yield

$$g_m(z,\delta^*) > g_m(z,\delta) \tag{7.23}$$

However, since δ^* is feasible and opt = max, (7.23) contradicts the assertion that δ is an optimal policy. We conclude then that (7.16) holds, and hence, that the functional equation for the case opt = max is valid.

Case 2. opt = min .

Reverse the above inequalities.

What remains to be shown then is that δ being an optimal Markovian policy entails that it must be a dynamic programming policy. In other words, it needs to be shown that if $\delta \in \Delta^*$ then

$$\delta(n,s) \in D^\circ(n,s) \ , \ \forall n \in \mathbb{N}, s \in S_n \tag{7.24}$$

This follows immediately from (7.14) and (7.4). Because, if δ is optimal it must also be feasible, hence $\delta(n,s) \in D(n,s)$, $\forall 1 \leq n \leq N$, $s \in S_n$. $\square$

This clear, I can now proceed to identify conditions ensuring the optimality of the dynamic programming policies and to describe their general structure. Prior to this, I need to call attention to two points. First, it is important to keep in mind that the same problem can have more than one set of optimal policies. This is so because, as we have seen, the same optimization problem can assume various valid dynamic programming formulations. The inference is therefore that each formulation would have a corresponding set of optimal policies.

Second, a *stationary* functional equation will give rise to stationary dynamic programming policies. Note that a stationary policy is a function on S with values in $\mathbb{D}$.

7.3 Truncated Functional Equations

Within the framework of truncated functional equations, to show that a Markovian policy δ is optimal one has to demonstrate that

$$g_n(s,\delta) = f_n(s) \ , \ \forall 1 \leq n \leq N, s \in S_n \tag{7.25}$$

I shall prove this by induction on n. Assume then that the truncated dynamic programming functional equation is valid, and let δ be any dynamic programming policy. Considering that, by definition, $\delta(N,s) \in D^{\circ}(N,s)$ for all $s \in S_N$ and $g_N(s,x) = f_N(s)$ for all $s \in S_N$ and $x \in D^{\circ}(N,s)$, it follows that $g_N(s,\delta) = f_N(s)$ for all $s \in S_N$. Hence, the inductive hypothesis holds for $n = N$. This being so, assume next that it holds for all $N \geq n \geq m+1$ as well. In this case, since

$$f_{m+1}(s) = g_{m+1}(s,\delta) \ , \ \forall s \in S_{m+1} \tag{7.26}$$

then (7.3) implies that

$$f_m(s) = \underset{x \in D(m,s)}{\text{opt}} \ \rho(m,s,x,g_{m+1}(T(n,s,x),\delta)), \forall s \in S_m \tag{7.27}$$

so that (7.4), in conjunction with (7.25)-(7.27), entail that

$$f_m(s) = \rho(m,s,x,g_{m+1}(T(m,s,x),\delta)) \ , \ \forall s \in S_m, x \in D^{\circ}(m,s) \tag{7.28}$$

Furthermore, δ being a dynamic programming policy implies that $\delta(m,s) \in D^{\circ}(m,s)$, $\forall s \in S_m$, so that (7.28) implies that

$$f_m(s) = \rho(m,s,x,g_{m+1}(T(m,s,x),\delta)) \ , \ x = \delta(m,s) \tag{7.29}$$

for all $s \in S_m$. Next, (ρ, G) being a decomposition scheme entails that,

$$g_m(s,\delta) = \rho(m,s,x,g_{m+1}(T(m,s,x),\delta)) \ , \ x = \delta(m,s) \tag{7.30}$$

for all $s \in S_m$, so that (7.29) yields

$$f_m(s) = g_m(s,\delta) \ , \ \forall s \in S_m \tag{7.31}$$

Hence, the inductive hypothesis holds for $n = m$. We conclude then that δ is optimal. And to phrase this verbally:

Theorem 7.3.1 *If N is finite and the dynamic programming functional equation is valid, then* ALL *the dynamic programming policies are optimal, in other words $\Delta^\circ \subseteq \Delta^*$.*

Thus, *Theorem 7.2.1,* in conjunction with *Theorem 7.3.1,* imply the following.

Corollary 7.3.1 *If N is finite and the dynamic programming functional equation is valid then $\Delta^\circ = \Delta^*$.*

Let us now illustrate this by means of a concrete example.

7.3.1 Example

Consider the following problem:

$$p := \min_{(x_1,\dots,x_N)} \sum_{n=1}^{N} (x_n - a)^2 \tag{7.32}$$

$$\sum_{n=1}^{N} x_n = a \ , \ a \geq 0 \tag{7.33}$$

$$x_n \geq 0 \ , \ n \in \{1,2,3,\dots,N\} \tag{7.34}$$

In line with the *Problem $P(s)$* format, we set:

$$S = \mathbb{D} = S_1 = [0,a] \tag{7.35}$$

$$D(n,s) = \begin{cases} [0,s] & , \quad 1 \leq n < N \\ \{s\} & , \quad n = N \end{cases} \ , \ s \in S \tag{7.36}$$

$$T(s,x) = s - x \ , \ s \in S, x \in D(s) \tag{7.37}$$

$$g(x_1,x_2,\dots,x_N) = \sum_{n=1}^{N} (x_n - a)^2 \tag{7.38}$$

I note parenthetically that T is stationary and that the objective function g is insensitive to the initial state. Also, the conditional form of (7.36) is due to the constraint (7.33) involving an equality. The reader may wish to ascertain that (7.33)-(7.34) is equivalent to

$$x_n \in D(n, s_n) \ , \ n = 1, 2, 3, \dots, N \tag{7.39}$$

where $s_1 = a$ and $s_{n+1} = T(s_n, x_n)$.

Back to where we left off. Since the objective function is additive, we use an additive decomposition scheme consisting of the modified objective functions

$$g_n(x_n, \dots, x_N) = \sum_{m=n}^{N} (x_m - a)^2 \tag{7.40}$$

and the composition function

$$\rho(x, r) = (x - a)^2 + r \ , \ \ x \in \mathbb{D}, \ r \in \mathbb{R} \tag{7.41}$$

Note that ρ is affected neither by n nor by s.

Thus, the functional equation induced by this decomposition scheme is as follows:

$$f_n(s) = \min_{x \in D(n,s)} \{(x - a)^2 + f_{n+1}(s - x)\} \ , \ \ 1 \le n < N, s \in S \tag{7.42}$$

$$= \min_{0 \le x \le s} \{(x - a)^2 + f_{n+1}(s - x)\} \ , \ \ 1 \le n < N, 0 \le s \le a \tag{7.43}$$

where

$$f_N(s) := \min_{x \in D(N,s)} (x - a)^2 \ , \ \ s \in S \tag{7.44}$$

$$= \min_{x \in \{s\}} (x - a)^2 \ , \ \ 0 \le s \le a \tag{7.45}$$

$$= (s - a)^2, \ \ 0 \le s \le a \tag{7.46}$$

Now, granted (7.46), and because clearly

$$D^{\circ}(N, s) = D(N, s) = \{s\} \ , \ \ \forall 0 \le s \le a \tag{7.47}$$

we can start directly with $n = N - 1$, in which case (7.43)-(7.46) yield

$$f_{N-1}(s) = \min_{0 \le x \le s} \{(x - a)^2 + (s - x - a)^2\} \ , \ \ 0 \le s \le a \tag{7.48}$$

$$= \min_{0 \le x \le s} \{2x^2 - 2sx + (2a^2 + s^2 - 2as)\} \tag{7.49}$$

As the right hand side of (7.49) requires the minimization of a convex function, we equate the first derivative of the function being minimized to zero. This yields the equation $4x - 2s = 0$ which in turn implies that the optimal solution is of the form

$$x^*(s) = \frac{s}{2} \ , \ \ 0 \le s \le a \tag{7.50}$$

We therefore set

$$D^{\circ}(N-1,s) = \left\{\frac{s}{2}\right\} \ , \ 0 \le s \le a \tag{7.51}$$

and

$$f_{N-1}(s) = [x^*(s) - a]^2 + [s - x^*(s) - a]^2 , \ 0 \le s \le a \tag{7.52}$$

$$= \left[\frac{s}{2} - a\right]^2 + \left[s - \frac{s}{2} - a\right]^2 \tag{7.53}$$

$$= 2\left[\frac{s}{2} - a\right]^2 \tag{7.54}$$

Thus, for $n = N-2$ the functional equation takes the following form:

$$f_{N-2}(s) = \min_{0 \le x \le s} \left\{ (x-a)^2 + 2\left[\frac{s-x}{2} - a\right]^2 \right\} \ , \ 0 \le s \le a \tag{7.55}$$

Again, the right-hand side calls for the minimization of a convex function. The equation rendered by equating the first derivative of the function being minimized to zero is

$$2(x-a) + 4\left[\frac{s-x}{2} - a\right](-1/2) = 0 \tag{7.56}$$

The unique solution to this equation, expressed as a function of s, is

$$x^*(s) = \frac{s}{3} \ , \ 0 \le s \le a \tag{7.57}$$

This yields in turn

$$f_{N-2}(s) = [x^*(s) - a]^2 + 2\left\{\left[\frac{s - x^*(s)}{2}\right] - a\right\}^2 \ , \ 0 \le s \le a \tag{7.58}$$

$$= \left[\frac{s}{3} - a\right]^2 + 2\left\{\left[\frac{s - s/3}{2}\right] - a\right\}^2 \tag{7.59}$$

$$= 3\left[\frac{s}{3} - a\right]^2 \tag{7.60}$$

As a definite pattern can now be discerned, we can proceed by induction, appealing to the following inductive hypothesis:

$$D^{\circ}(n,s) = \left\{\frac{s}{N-n+1}\right\} \ , \ 1 \le n \le N, 0 \le s \le a \tag{7.61}$$

and

$$f_n(s) = [N-n+1]\left[\frac{s}{N-n+1} - a\right]^2 \ , \ 1 \le n \le N, 0 \le s \le a \tag{7.62}$$

Thus, for $1 \leq m < N$, we have

$$f_m(s) = \min_{0\leq x\leq s} \left\{(x-a)^2 + f_{m+1}(s-x)\right\}, 0 \leq s \leq a \tag{7.63}$$

$$= \min_{0\leq x\leq s} \left\{(x-a)^2 + (N-m)\left[\frac{s-x}{N-m} - a\right]^2\right\} \tag{7.64}$$

Equating the first derivative of the convex function being minimized to zero yields the following equation:

$$2(x-a) + 2(N-m)\left[\frac{s-x}{N-m} - a\right]\left[-\frac{1}{N-m}\right] = 0 \tag{7.65}$$

Since the unique solution is of the form

$$x^*(s) = \frac{s}{N-m+1}\ ,\ 0 \leq s \leq a \tag{7.66}$$

it follows that (7.61) holds for $n = m$.

Finally, to ascertain that (7.62) holds as well, note that

$$f_m(s) = [x^*(s) - a]^2 + (N-m)\left[\left[\frac{s - x^*(s)}{N-m}\right] - a\right]^2 \tag{7.67}$$

$$= \left[\frac{s}{N-m+1} - a\right]^2 + (N-m)\left[\left[\frac{s - \dfrac{s}{N-m+1}}{N-m}\right] - a\right]^2 \tag{7.68}$$

$$= (N-m+1)\left[\frac{s}{N-m+1} - a\right]^2 \tag{7.69}$$

The inference is then that Δ°, and therefore Δ^*, consist of a single Markovian policy specified as follows:

$$\delta(n,s) = \frac{s}{N-n+1}\ ,\ 1 \leq n \leq N, 0 \leq s \leq a \tag{7.70}$$

Let us now elucidate the meaning of this policy. To this end, let $m = N-n+1$. Hence, $\delta(n,s) = s/m$. By construction, m stipulates the number of remaining decisions left to be made when we are at stage n. The implication is then that this policy dictates that the quantity s_n be divided equally between the remaining stages of the process. When applied at the initial state $s_1 = a$, this policy would generate the following sequence of states and decisions:

$$x_1 = \delta(1, s_1) = \frac{a}{N-1+1} = \frac{a}{N} \tag{7.71}$$

$$s_2 = T(s_1, x_1) = s_1 - x_1 = a - \frac{s}{N} = \frac{a(N-1)}{N} \tag{7.72}$$

$$x_2 = \delta(2, s_2) = \frac{s_2}{N-2+1} = \frac{s_2}{N-1} = \frac{a}{N} \tag{7.73}$$

$$s_3 = T(s_2, x_2) = s_2 - x_2 = \left[\frac{a(N-1)}{N}\right] - \frac{a}{N} = \frac{a(N-2)}{N} \tag{7.74}$$

$$x_3 = \delta(3, s_3) = \frac{s_3}{N-3+1} = \frac{s_3}{N-2} = \frac{a}{N} \tag{7.75}$$

so that by induction,

$$x_n = \frac{a}{N} \ , \ n = 1, \dots, N \tag{7.76}$$

$$s_n = \frac{a(N-n+1)}{N} \ , \ n = 1, \dots, N \tag{7.77}$$

It should be noted that the above assurances notwithstanding, it is always good practice to verify that the policy concerned is feasible, and that it satisfies the optimality condition $g_n(s,\delta) = f_n(s)$, $\forall 1 \leq n \leq N$, $s \in S_n$. In the case under consideration, the feasibility conditions require that the decisions be nonnegative and that their sum be equal to a. That these conditions are met by the above sequence is immediately apparent. That $g_n(s,\delta) = f_n(s)$ for all $1 \leq n \leq N$ and $s \in S$, is also met, can be ascertained as follows. By definition,

$$\begin{align} g_n(s_n,\delta) &= \sum_{m=n}^{N} [\delta(m,s_m) - a]^2 \tag{7.78}\\ &= \sum_{m=n}^{N} \left[\frac{s_m}{N-m+1} - a\right]^2 \tag{7.79}\end{align}$$

Applying δ to (n, s_n) yields the states

$$\begin{align} s_{n+1} &= T(n, s_n, \delta(n,s_n)) \tag{7.80}\\ &= s_n - \delta(n,s_n) \tag{7.81}\\ &= s_n - \frac{s_n}{N-n+1} \tag{7.82}\\ &= \frac{N-n}{N-n+1} s_n \tag{7.83}\\ s_{n+2} &= T(n+1, s_{n+1}, \delta(n+1, s_{n+1})) \tag{7.84}\\ &= s_{n+1} - \delta(n+1, s_{n+1}) \tag{7.85}\\ &= s_{n+1} - \frac{s_{n+1}}{N-n} \tag{7.86}\\ &= \frac{N-n-1}{N-n} s_{n+1} \tag{7.87}\end{align}$$

so by induction,

$$s_m = \frac{N-m+1}{N-n+1} s_n \ , \ n \leq m \leq N \tag{7.88}$$

Thus, (7.79)-(7.88) yield

$$g_n(s_n,\delta) = \sum_{m=n}^{N} \left[\frac{s_m}{N-m+1} - a\right]^2 \tag{7.89}$$

$$= \sum_{m=n}^{N} \left[\frac{s_n}{N-n+1} - a\right]^2 \tag{7.90}$$

$$= (N-n+1)\left[\frac{s_n}{N-n+1} - a\right]^2 \tag{7.91}$$

$$= f_n(s) \tag{7.92}$$

In particular,

$$g_1(a,\delta) = f_1(a) = N\left[\frac{a(N-1)}{N}\right]^2 = \frac{a^2(N-1)^2}{N} \qquad \square \tag{7.93}$$

As a final note, observe that the policies obtained in the solution of truncated functional equations are usually non-stationary. This is a consequence of the boundary conditions imposed at the final stage, $n = N$.

7.4 Nontruncated Functional Equations

Consider the functional equation

$$f_n(s) = \max_{x\in D(n,s)} \rho(n,s,x,f_{n+1}(T(n,s,x))) \tag{7.94}$$

for $n \in \mathbb{N} := \{0,1,2,\dots\}, s \in S_n$, or equivalently,

$$f^*(n,s) = \max_{x\in D(n,s)} \rho(n,s,x,f^*(n+1,T(n,s,x))) \ , \ (n,s)\in Z \tag{7.95}$$

where

$$Z := \{(n,s) : n \in N, s \in S_n\} \tag{7.96}$$

$$f^*(n,s) := f_n(s) \ , \ (n,s) \in Z \tag{7.97}$$

You will recall that the successive approximation method fixes on two objects: the set U which consists of all the bounded real-valued functions on Z, and a map A on U into itself defined as follows:

$$(Au)(n,s) := \sup_{x\in D(n,s)} \rho(n,s,x,u(n+1,T(n,s,x))) \ , \ (n,s)\in Z \tag{7.98}$$

Now, let (ρ, G) be any decomposition scheme such that (7.95) holds, and for each Markovian policy $\delta \in \Delta$ assume that A_δ is a map on U into itself defined thus

$$(A_\delta u)(n,s) := \rho(n,s,x,u(n+1,T(n,s,x))) \ , \ x=\delta(n,s), (n,s)\in Z \tag{7.99}$$

In view of (ρ, G) being a decomposition scheme, we have

$$g_n(s,\delta) = \rho(n,s,x,g_{n+1}(T(n,s,x),\delta)) \ , \ x = \delta(n,s) \tag{7.100}$$

so that defining

$$g_\delta(n,s) := g_n(s,\delta) \ , \ (n,s) \in Z, \delta \in \Delta \tag{7.101}$$

yields

$$g_\delta(n,s) = \rho(n,s,x,g_\delta(n+1,T(n,s,x))) \ , \ x = \delta(n,s) \tag{7.102}$$

Consequently, if we define

$$g_\delta(s) := g(s,\delta) \ , \ s \in S, \delta \in \Delta \tag{7.103}$$

taking g_δ to be — for each $\delta \in \Delta$ — a real-valued function on Z, then (7.99), in conjunction with (7.102), yield the following:

$$g_\delta = A_\delta g_\delta \ , \ \forall \delta \in \Delta \tag{7.104}$$

Finally, by construction, a dynamic programming policy has the characteristic that

$$f^* = A_\delta f^*, \ \forall \delta \in \Delta^\circ \tag{7.105}$$

as by definition, $\delta \in \Delta^\circ$ entails that $\delta(n,s) \in D^\circ(n,s)$ for all $(n,s) \in Z$ and that

$$\rho(n,s,x,f_{n+1}(T(n,s,x))) = f_n(s) \ , \ \forall (n,s) \in Z, x \in D^\circ(n,s) \tag{7.106}$$

Now, the object is to identify the conditions assuring the optimality of the policy described above. To this end, one needs to show that for each $\delta \in \Delta$ the equation $u = A_\delta u$ has a *unique* solution in U, as this will imply that $f^* = g_\delta$, $\forall \delta \in \Delta^\circ$. Consider then the following.

Theorem 7.4.1 *Assume that there exists an $\alpha \in [0,1)$ such that*

$$|\rho(n,s,x,a) - \rho(n,s,x,b)| \le \alpha |a-b| \tag{7.107}$$

for all $(n,s) \in Z$, $x \in D(n,s)$ and $(a,b) \in \mathbb{R}^2$. Then, for each $\delta \in \Delta$, the equation $u = A_\delta u$ has a UNIQUE *solution in U.*

PROOF. Let δ be any element of Δ. Granted (7.104), it follows that the equation $u = A_\delta u$ has at least one solution in U. Now, suppose that this equation has two solutions, say, u and v, that is, assume that there is a pair $(u,v) \in U^2$, such that $u = A_\delta u$ and $v = A_\delta v$. We need to show then that $u = v$. Since $u = A_\delta u$ and $v = A_\delta v$, then clearly

$$\begin{aligned} |u(n,s) - v(n,s)| &= |(A_\delta u)(n,s) - A_\delta(n,s)| \ , \ \forall (n,s) \in Z & (7.108)\\ &= |\rho(n,s,x,u(n+1,s')) - \rho(n,s,x,v(n+1,s')| & (7.109)\\ &\le \alpha |u(n+1,s') - v(n+1,s')| & (7.110)\end{aligned}$$

for some $0 \leq \alpha < 1$, where $x = \delta(n,s)$, and $s' = T(n,s,x)$. Hence,

$$\sup_{(n,s)\in Z} |u(n,s) - v(n,s)| \leq \alpha \sup_{(n,s)\in Z} |u(n+1,s') - v(n+1,s')| \quad (7.111)$$

Next, because $(n,s) \in Z$ entails that $(n+1,s') \in Z$, it follows that

$$\sup_{(n,s)\in Z} |u(n,s) - v(n,s)| \leq \alpha \sup_{(n,s)\in Z} |u(n,s) - v(n,s)| \quad (7.112)$$

Finally, considering that $0 \leq \alpha < 1$, we may conclude that

$$\sup_{(n,s)\in Z} |u(n,s) - v(n,s)| = 0 \quad (7.113)$$

the implication being that $u(n,s) = v(n,s)$, $\forall (n,s) \in Z$, that is, $u = v$. $\square$

I call attention to the fact that (7.107) is in effect the condition laid down by *Theorem 6.4.2* to ensure that the successive approximation procedure yields the function f^*. This means then that *Theorem 7.4.1* covers Type-One dynamic programming functional equations. To illustrate this, let us go back to the functional equation studied in *Example 6.4.3.*

7.4.1 Example

Consider then the following functional equation:

$$f(s) = \max_{0\leq x\leq s} \left\{x^{1/2} + \beta f(s-x)\right\} \ , \ 0 \leq \beta < 1, 0 \leq s \leq r \quad (7.114)$$

You will recall that its solution was found to be

$$f(s) = \left[\frac{s}{1-\beta^2}\right]^{1/2} \ , \ 0 \leq s \leq r \quad (7.115)$$

Since the composition function in this case is of the form

$$\rho(x,a) = x^{1/2} + \beta a \ , \ a \in \mathbb{R} \quad (7.116)$$

it follows that

$$\begin{aligned} |\rho(x,a) - \rho(x,b)| &= |x^{1/2} + \beta a - x^{1/2} - \beta b| && (7.117)\\ &= |\beta(a-b)| && (7.118)\\ &= \beta|a-b| && (7.119)\end{aligned}$$

Hence, (7.107) proves true with $\alpha = \beta$.

Of course, in view of this formulation being stationary, the dynamic programming policies will be stationary as well.

Let us now attempt to establish the characteristics of the dynamic programming policy in question. This involves the construction of the sets $\{D^\circ(s)\}, s \in S$, which in this case, are defined as follows

$$D^\circ(s) := \{x \in [0,s] : \rho(s,x,\beta f(s-x)) = f(s)\} \ , \ 0 \le s \le r \tag{7.120}$$
$$= \left\{ x \in [0,s] : \ x^{1/2} + \beta \left[\frac{s-x}{1-\beta^2}\right]^{1/2} = \left[\frac{s}{1-\beta^2}\right]^{1/2} \right\} \tag{7.121}$$

Suppose then that we pick any arbitrary $s \in [0,r]$. Then the equation defining $D^\circ(s)$ would assume the following form:

$$\left[\frac{s}{1-\beta^2}\right]^{1/2} = x^{1/2} + \beta \left[\frac{s-x}{1-\beta^2}\right]^{1/2} \tag{7.122}$$

so that

$$s^{1/2} = \left[x(1-\beta^2)\right]^{1/2} + \beta(s-x)^{1/2} \tag{7.123}$$

Thus,

$$\left[x(1-\beta^2)\right]^{1/2} = s^{1/2} - \beta(s-x)^{1/2} \tag{7.124}$$

hence,

$$x(1-\beta^2) = \left[s^{1/2} - \beta(s-x)^{1/2}\right]^2 \tag{7.125}$$
$$= s - 2\beta s^{1/2}(s-x)^{1/2} + \beta^2(s-x) \tag{7.126}$$

and consequently

$$2\beta s^{1/2}(s-x)^{1/2} = s + \beta^2(s-x) - x(1-\beta^2) \tag{7.127}$$
$$= s(1+\beta^2) - x \tag{7.128}$$

which in turn implies that

$$4\beta^2 s(s-x) = \left[s(1+\beta^2) - x\right]^2 \tag{7.129}$$
$$= s^2(1+\beta^2)^2 - 2xs(1+\beta^2) + x^2 \tag{7.130}$$

This yields the quadratic equation

$$x^2 - 2s(1-\beta^2)x + s^2(1-\beta^2)^2 = 0 \tag{7.131}$$

or equivalently,

$$\left[x - s(1-\beta^2)\right]^2 = 0 \tag{7.132}$$

Because this equation has a unique solution of the form

$$x(s) = s(1-\beta^2) \tag{7.133}$$

it follows that

$$D^{\circ}(s) = \left\{s(1-\beta^2)\right\} \ , \ 0 \leq s \leq r \tag{7.134}$$

and

$$\Delta^{\circ} = \{\delta : \delta(s) = s(1-\beta^2) \ , \ 0 \leq s \leq r\} \tag{7.135}$$

In other words, the set of dynamic programming policies is a singleton composed of a stationary Markovian policy. Notice that this result can be obtained by solving the right-hand side of the functional equation, as was done in (5.202).

Next, let us confirm that the obtained dynamic programming policy is optimal. First, as $\delta(s) \in D(s) = [0, r]$, it follows that δ is feasible. We now need to ascertain that $f_1(s) = g_\delta(s)$ for all $0 \leq s \leq r$. Thus, because

$$g(x_1, x_2, \dots) = \sum_{n=1}^{\infty} \beta^{n-1} x_n^{1/2} \tag{7.136}$$

it follows that

$$g_\delta(s_1) = \sum_{n=1}^{\infty} \beta^{n-1} \left[\delta(s_n)\right]^{1/2} \ , \ s_1 \in [0, r] \tag{7.137}$$

$$= \sum_{n=1}^{\infty} \beta^{n-1} \left[s_n(1-\beta^2)\right]^{1/2} \tag{7.138}$$

To evaluate the right-hand side of (7.138), let us identify the sequence of states generated by applying δ to some initial state s_1. By construction then,

$$s_2 = T(s_1, \delta(s_1)) = s_1 - \delta(s_1) = s_1 - s_1(1-\beta^2) \tag{7.139}$$

$$= s_1\beta^2 \tag{7.140}$$

$$s_3 = T(s_2, \delta(s_2)) = s_2 - s_2(1-\beta^2) = s_2\beta^2 \tag{7.141}$$

$$= s_1\beta^4 \tag{7.142}$$

$$s_4 = T(s_3, \delta(s_3)) = s_3 - s_3(1-\beta^2) = s_3(1-\beta^2) \tag{7.143}$$

$$= s_1\beta^6 \tag{7.144}$$

so that by induction

$$s_n = s_1\beta^{2(n-1)} \ , \ n = 1, 2, 3, \dots \tag{7.145}$$

Thus, (7.138)-(7.145) yield

$$g_\delta(s_1) = \sum_{n=1}^{\infty} \beta^{n-1} \left[s_1\beta^{2(n-1)}(1-\beta^2)\right]^{1/2} \ , \ s_1 \in [0, r] \tag{7.146}$$

$$= \left[s_1(1-\beta^2)\right]^{1/2} \sum_{n=1}^{\infty} \beta^{2(n-1)} \tag{7.147}$$

$$= \frac{\left[s_1(1-\beta^2)\right]^{1/2}}{1-\beta^2} \tag{7.148}$$

$$= \left[\frac{s_1}{1-\beta^2}\right]^{1/2} \tag{7.149}$$

$$= f(s_1) \qquad \square \tag{7.150}$$

I now turn to the conditions assuring the optimality of dynamic programming policies associated with stationary functional equations of *Type Two.*

Theorem 7.4.2 *Assume that the stationary functional equation*

$$f(s) = \max_{x \in D(s)} \rho(s, x, f(T(s,x))) \ , \ s \in S \tag{7.151}$$

is valid. Also, assume that the dynamic programming model in question satisfies the following conditions:

1. *There exist a metric Θ' on S, a state $s^* \in S$ and a real number K such that*

$$\Theta'(s, s^*) \le K < \infty \ , \ \forall s \in S \tag{7.152}$$

2. *There exists an $\alpha \in [0,1)$ such that*

$$\Theta'(T(s,x), s^*) \le \alpha \Theta'(s, s^*) \ , \ \forall s \in S, x \in D(s) \tag{7.153}$$

3. *For any $(a,b) \in \mathbb{R}^2$, we have*

$$|\rho(s,x,a) - \rho(s,x,b)| \le |a-b| \ , \ \forall s \in S, x \in D(s)) \tag{7.154}$$

4. *Let $U(c)$ denote the set of all the bounded real-valued functions on S that are continuous at $s = s^*$ and equal to c there. Then $f \in U(c)$ and $g_\delta \in U(c), \forall \delta \in \Delta$ where $c = f(s^*)$ by construction.*

Then, all the dynamic programming policies concerned are optimal.

Proof. In view of (7.104) and (7.105), it is sufficient to show that, under the above conditions, for each dynamic programming policy δ, the equation $u = A_\delta u$ has a unique solution in $U(c)$. Let then δ be any dynamic programming policy associated with the functional equation (7.151) and assume that there is a pair $(u,v) \in U^2(c)$ such that $u = A_\delta u$ and $v = A_\delta v$. Then clearly, for any subset Y of S, we have

$$\Theta_Y(u,v) := \sup_{s \in Y} |u(s) - v(s)| \tag{7.155}$$

$$= \Theta_Y(A_\delta u, A_\delta v) \tag{7.156}$$

$$= \sup_{s \in Y} |\rho(s,x,u(T(s,x))) - \rho(s,x,v(T(s,x)))| \ , \ x = \delta(s)$$

$$\le \sup_{s \in Y} |u(T(s,x)) - v(T(s,x))| \tag{7.157}$$

In particular, (7.157) implies that for $Y = S$, we have

$$\begin{aligned}\Theta_S(u,v) &\leq \sup_{s\in S} |u(T(s,x)) - v(T(s,x))|, x = \delta(s)\\ &\leq \sup_{s\in S(1)} |u(s) - v(s)|\\ &= \Theta_{S(1)}(u,v) \qquad (7.158)\end{aligned}$$

where

$$S(m) := \{s : s \in S, \Theta'(s,s^*) \leq \alpha^m K\} \ , \ m = 0,1,2,3,\ldots \qquad (7.159)$$

as (7.153) entails that

$$\{T(s,x) : s \in S, x \in D(s)\} \subset S(1) \qquad (7.160)$$

Similarly, considering that $u = A_\delta u$ and $v = A_\delta v$, it follows that

$$\begin{aligned}\Theta_{S(m)}(u,v) &= \Theta_{S(m)}(A_\delta u, A_\delta v) && (7.161)\\ &\leq \sup_{s\in S(m)} |u(T(s,x)) - v(T(s,x))| \ , \ x = \delta(s) && (7.162)\\ &\leq \sup_{s\in S(m+1)} |u(s) - v(s)| && (7.163)\\ &= \Theta_{S(m+1)}(u,v) && (7.164)\end{aligned}$$

because (7.159) implies that

$$\{T(s,x) : s \in S(m), x \in D(s)\} \subset S(m+1) \ , \ \forall m = 0,1,2,3,\ldots \qquad (7.165)$$

Hence,

$$\Theta_S(u,v) \leq \Theta_{S(m)}(u,v) \ , \ \forall m = 0,1,2,3,\ldots \qquad (7.166)$$

Now, observe that $s^* \in S(m)$ for all $m \geq 0$ and that the sequence $\{S(m)\}$ converges to $\{s^*\}$. Since u and v are continuous at $s = s^*$ and equal there, it follows that the right-hand side of (7.166) converges to zero. We conclude then that $\Theta_S(u,v) = 0$, which in turn implies that $u = v$.

In short, for each $\delta \in \Delta$, the equation $u = A_\delta u$ has a unique solution in $U(c)$ and therefore $g_\delta = f$, $\forall \delta \in \Delta^\circ$. □

7.4.2 Example

Consider again the following functional equation studied in *Example 6.1.3,*

$$f(s) = \max_{0\leq x\leq s} \{x^{1/2} + f(\beta(s-x))\} \ , \ 0 \leq \beta < 1, 0 \leq s \leq r \qquad (7.167)$$

Recall that the solution we obtained for it had this form

$$f(s) = \left[\frac{s}{1-\beta}\right]^{1/2} \ , \ 0 \leq s \leq r \qquad (7.168)$$

As I demonstrated already that the first three conditions postulated by *Theorem 7.4.2* are met with $s^* = 0$, $K = r$, $\alpha = \beta$ and $\Theta'(s, s') = |s - s'|$, and that

$$0 \leq f(s) \leq \frac{s^{1/2}}{1 - \beta^{1/2}} \ , \ 0 \leq s \leq r \tag{7.169}$$

I now need to show that $g_\delta \in U(c)$, $\forall \delta \in \Delta$.

However, since it is clear that $f(s) \geq g_\delta(s)$, $\forall s \in S$, $\delta \in \Delta$, it is sufficient to show that $g_\delta(s) \geq 0$, $\forall s \in S$, $\delta \in \Delta$.

Thus, given that the objective function under consideration is of the form

$$g(x_1, x_2, \dots) = \sum_{n=1}^{\infty} x_n^{1/2} \tag{7.170}$$

it follows that $g_\delta(s) \geq 0$, $\forall s \in S$, $\delta \in \Delta$. Hence, the fourth condition required by *Theorem 7.4.2* is met as well.

To find out what structure the dynamic programming policies would have in this case, we again need to determine the makeup of the sets $\{D^\circ(s)\}$. And so, we set

$$D(s) = [0, s] \tag{7.171}$$
$$T(s, x) = \beta(s - x) \tag{7.172}$$
$$\rho(x, a) = x^{1/2} + a \tag{7.173}$$

hence by definition

$$D^\circ(s) := \{x : 0 \leq x \leq s, \rho(s, x, f(\beta(s - x))) = f(s)\}, 0 \leq s \leq r \tag{7.174}$$
$$= \left\{ x \in [0, s] : x^{1/2} + \left[\frac{\beta(s - x)}{1 - \beta}\right]^{1/2} = \left[\frac{s}{1 - \beta}\right]^{1/2} \right\} \tag{7.175}$$

Choosing any $0 \leq s \leq r$, the equation

$$x^{1/2} + \left[\frac{\beta(s - x)}{1 - \beta}\right]^{1/2} = \left[\frac{s}{1 - \beta}\right]^{1/2} \tag{7.176}$$

or equivalently the following,

$$s^{1/2} = [x(1 - \beta)]^{1/2} + [\beta(s - x)]^{1/2} \tag{7.177}$$

yields

$$s = \left[[x(1 - \beta)]^{1/2} + [\beta(s - x)]^{1/2} \right]^2 \tag{7.178}$$
$$= x(1 - \beta) + 2\left[x(1 - \beta)\beta(s - x)\right]^{1/2} + \beta(s - x) \tag{7.179}$$

Hence,

$$2\left[x(1-\beta)\beta(s-x)\right]^{1/2} = s - x(1-\beta)\beta(s-x) \tag{7.180}$$
$$= s(1-\beta) - x(1-2\beta) \tag{7.181}$$

so that

$$4x(1-\beta)\beta(s-x) = [s(1-\beta)-x(1-2\beta)]^2 \tag{7.182}$$
$$= s^2(1-\beta)^2 - 2s(1-\beta)x(1-2\beta) + x^2(1-2\beta)^2 \tag{7.183}$$

Rearranging the terms of this equation gives the following quadratic equation:

$$x^2 - 2s(1-\beta)x + s^2(1-\beta)^2 = 0 \tag{7.184}$$

which can be rewritten as

$$\big[x - s(1-\beta)\big]^2 = 0 \tag{7.185}$$

Clearly, the unique solution is $x(s) = s(1-\beta)$.

It follows therefore that for each $s \in S$, the set $D^\circ(s)$ consists of the single element $s(1-\beta)$, which in turn implies that the set Δ° is composed of the single stationary Markovian policy of the following structure:

$$\delta(s) = s(1-\beta)\ ,\ 0 \le s \le r \tag{7.186}$$

To ascertain that $g_\delta(s) = f(s)$, $\forall 0 \le s \le r$, note that

$$g_\delta(s_1) = \sum_{n=1}^{\infty} x_n^{1/2}\ ,\ x_n = \delta(s_n) \tag{7.187}$$

To evaluate this function, we generate the sequence $(x_1, \dots, x_N)$ thus:

$$x_1 = \delta(s_1) = s_1(1-\beta) \tag{7.188}$$
$$s_2 = T(s_1, x_1) = \beta(s_1 - x_1) = \beta(s_1 - s_1(1-\beta)) = s_1\beta^2 \tag{7.189}$$
$$x_2 = \delta(s_2) = s_2(1-\beta) = s_1\beta^2(1-\beta) \tag{7.190}$$
$$s_3 = T(s_2, x_2) = \beta(s_2 - x_2) = \beta(s_1\beta^2 - s_1\beta^2(1-\beta)) = s_1\beta^4 \tag{7.191}$$
$$x_3 = \delta(s_3) = s_3(1-\beta) = s_1\beta^4(1-\beta) \tag{7.192}$$

so by induction

$$s_n = s_1\beta^{2(n-1)}\ ,\ n = 1, 2, \dots \tag{7.193}$$
$$x_n = s_1\beta^{2(n-1)}(1-\beta)\ ,\ n = 1, 2, \dots \tag{7.194}$$

Hence,

$$g_\delta(s_1) = \sum_{n=1}^{\infty}\left[s_1\beta^{2(n-1)}(1-\beta)\right]^{1/2} \tag{7.195}$$

$$= [s(1-\beta)]^{1/2}\sum_{n=1}^{\infty}\beta^{n-1} \tag{7.196}$$

$$= \frac{[s(1-\beta)]^{1/2}}{1-\beta} \tag{7.197}$$

$$= \left[\frac{s}{1-\beta}\right]^{1/2} \tag{7.198}$$

$$= f(s) \tag{7.199}$$

Obviously, this was a reconstruction of what in practice would be rendered in the solution of the functional equation. For instance, the policy δ specified by (7.69) would be the result (6.53), yielded by the successive approximation procedure. □

Finally, I wish to point out that the optimal policy can also be obtained by means of a successive approximation in the *policy space*. In this concluding section I give a concise illustration of such a procedure highlighting those points that require attention.

7.5 Successive Approximation in the Policy Space

Let U denote the set of all the bounded real-valued functions on $Z = \{(n,s) : n \in \mathbb{N}, s \in S_n\}$ and, for each $u \in U$ define

$$\Delta(u) := \{\delta \in \Delta : \delta(n,s) \in D_u^\circ(n,s), \forall (n,s) \in Z\} \tag{7.200}$$

where for each $z = (n,s) \in Z$ and $u \in U$ set

$$D_u^\circ(z) := \{y \in D(z) : \rho(z,y,u(T^+(z,y))) = (Bu)(z)\} \tag{7.201}$$

such that B is a map from U into itself defined as follows

$$(Bu)(z) := \operatorname*{opt}_{x\in D(z)} \rho(z,x,u(T^+(z,x))) \tag{7.202}$$

where

$$T^+(z,x) := (n+1, T(n,s,x))\ ,\ x \in D(n,s), z = (n,s) \tag{7.203}$$

In other words, for any $u \in U$ let the set $\Delta(u)$ consist of all the Markovian policies satisfying the condition

$$\rho(n,s,\delta(n,s),u(n+1,T(n,s,\delta(n,s)))) = \sup_{x\in D(n,s)} \rho(n,s,x,u(n+1,T(n,s,x))) \quad (7.204)$$

for all $(n,s)\in Z$. Thus, by construction, if $u\in U$, $\delta\in\Delta(u)$ and $(n,s)\in Z$, then $x^*=\delta(n,s)$ is a solution to the optimization problem

$$\sup_{x\in D(n,s)} \rho(n,s,x,u(n+1,T(n,s,x))) \quad (7.205)$$

The short-hand $u=g_\delta$ is used instead of

$$u(n,s)=g_n(s,\delta) \ , \ \forall n\in N, s\in S_n \quad (7.206)$$

where $g_n(s,\delta)$ is defined by (7.8). Consider now the following.

7.5.1 Procedure

Step 1. Select a policy δ from Δ and set $u^{(0)}=g_\delta$, $m=0$ and $\delta^{(0)}=\delta$.
Step 2. Select a police $\delta^{(m+1)}$ from $\Delta(u^{(m)})$.
Step 3. If $\delta^{(m+1)}=\delta^{(m)}$, set $\delta^*=\delta^{(m)}$ and $u^*=g_{\delta^*}$ and stop. Otherwise, set $u^{(m+1)}=g_\delta$, $z\in Z$, $\delta=\delta^{(m+1)}$, $m=m+1$ and go to Step 2. □

Any procedure of this type is understood to perform a *successive approximation in the policy space.* The following observations should be made with regard to this procedure:

1. If the procedure terminates, then clearly, $u^*=Bu^*$. Thus, to ensure that $u^*=f^*$ and consequently that $\delta^*\in\Delta^*$, it is sufficient to demonstrate that the equation $u=Bu$ has a unique solution in U. Also, to ensure the convergence of the sequence $\{\delta^{(m)}\}$ to some $\delta^*\in\Delta^*$, it is necessary to impose regularity conditions on the functional equation.
2. The weak point in this procedure is the difficulty involved in determining the function $u^{(m+1)}=g_\delta \ , \ \delta=\delta^{(m+1)}$.
3. This procedure is viable provided that each optimization problem engendered in the process has an optimal solution. That is, to determine $\delta^{(m+1)}$, for each $(n,s)\in Z$ the problem

$$\operatorname*{opt}_{x\in D(n,s)} \rho(n,s,x,u^{(m)}(n+1,T(n,s,x))) \quad (7.207)$$

must attain an optimal solution. □

I shall not discuss this procedure any further, except to indicate the following.

Theorem 7.5.1 *Let δ be any element of Δ and consider the sequence $\{u^{(m)}\}$, where $u^{(0)} = g_\delta$, $u^{(m+1)} = Au^{(m)}$ and A is the map defined in (7.98). Also, assume that* $\text{opt} = \max$ *and that the composition function is monotone nondecreasing with respect to its last argument, namely, assume that*

$$a \geq b \Rightarrow \rho(n,s,x,a) \geq \rho(n,s,x,b) \ , \ \forall (n,s) \in Z, x \in D(n,s) \tag{7.208}$$

Then the sequence $\{u^{(m)}\}$ is monotone nondecreasing. That is,

$$u^{(m+1)}(n,s) \geq u^{(m)}(n,s) \ , \ \forall m \geq 0, (n,s) \in Z \tag{7.209}$$

Proof. Since by construction, $u^{(m+1)} = Au^{(m)}$, and by definition,

$$(Au)(n,s) = \sup_{x \in D(n,s)} \rho(n,s,x,u(n+1,T(n,s,x))) \tag{7.210}$$

it follows that

$$u^{(m+1)}(n,s) = \sup_{x \in D(n,s)} \rho(n,s,x,u^{(m)}(n+1,T(n,s,x))) \ , \ (n,s) \in Z \tag{7.211}$$

Thus, as ρ is taken to be monotone, it is sufficient to show that

$$u^{(1)}(n,s) \geq u^{(0)}(n,s) \ , \ \forall (n,s) \in Z \tag{7.212}$$

because, by induction, it will follow that $u^{(m+1)} \geq u\,(^m)$ holds for all $m \geq 0$. Let then (n,s) be any element of Z, and let δ be any element of Δ. Next, set $u^{(0)} = g_\delta$. Hence, by construction,

$$\begin{aligned} u^{(1)}(n,s) &= (Au^{(0)})(n,s) && (7.213)\\ &= \sup_{x \in D(n,s)} \rho(n,s,x,u^{(0)}(n+1,T(n,s,x))) && (7.214)\\ &= \sup_{x \in D(n,s)} \rho(n,s,x,g_\delta(n+1,T(n,s,x))) && (7.215)\\ &\geq \rho(n,s,x,g_\delta(n+1,T(n,s,x))) \ , \ \forall x \in D(n,s) && (7.216) \end{aligned}$$

therefore,

$$\begin{aligned} u^{(1)}(n,s) &\geq \rho(n,s,\delta(n,s),g_\delta(n+1,T(n,s,\delta(n,s)))) && (7.217)\\ &= g_\delta(n,s) && (7.218)\\ &= u^{(0)}(n,s) \qquad \square && (7.219) \end{aligned}$$

To sum up, in *Chapter 6* I demonstrated that the sequence $\{u^{(m)}\}$ generated by the successive approximation procedure converges geometrically both in the case of *Type One* and in that of *Type Two* functional equations. *Theorem 7.4.2* adds to this that the convergence is monotone if ρ is monotone with respect to its last argument and the initial approximation is of the form $u^{(0)} = g_\delta$, $\delta \in \Delta$. As we shall see in *Chapter 10,* the composition function ρ would invariably satisfy the above monotonicity condition.

7.6 Summary

The optimal policies for a multistage decision problem are obtained in the solution of the dynamic programming functional equation. We saw that both in the case of truncated and in that of nontruncated *Type One* and *Type Two*) equations, all the pertinent dynamic programming policies are optimal. We also saw that a decomposition scheme satisfying the Markovian condition yields an equation such that all the optimal policies stemming from it are dynamic programming policies.

7.7 Bibliographic Notes

More on the questions of the existence and structure of optimal dynamic programming policies can be found in Balder [1981], Bellman [1957a, 1971], Blackwell [1964], Boltyanskii [1966], Denardo [1965, 1968, 1982], Denardo and Mitten [1967], Hinderer [1970], Howard [1960], Karlin [1955], Kreps [1977a, 1977b], Porteus [1982], Ross [1974, 1983], Sniedovich [1979], Sobel [1975b], Veinott [1966], Walters and Templeman [1979], Yakowitz [1969].

8

The Curse of Dimensionality

8.1 Introduction

Having described the techniques used to solve the dynamic programming functional equation, I now take up a difficulty that often impedes a successful solution of this equation. This difficulty, which has proved to be the bane of many dynamic programming applications, was summed up by Bellman [1957a p. xii] in his now classic phrase: *The Curse of Dimensionality*.

Very broadly, in the context of dynamic programming, this graphic phrase indicates that the volume of computation required to solve the dynamic programming functional equation often increases very rapidly (exponentially?) with the size of a problem. Indeed, the amount of computation can assume such magnitude that solving the equation becomes a practical impossibility.

Of course, the problem of dimensionality is not an ill that afflicts only dynamic programming. Still, it is of particular significance for dynamic programming because it hits hardest at what one might well argue is dynamic programming's forte, namely *combinatorial optimization problems*, or what I shall call here *discrete optimization problems*.

I therefore devote this chapter to a brief analysis of the *Curse of Dimensionality* and to an assessment of the prospects for its elimination.

8.2 Motivation

Dynamic programming can, no doubt, handle problems involving non-denumarable decision sets $\{D(n,s)\}$, and to be sure, it is being used successfully for this purpose. But the point to note here is that in such cases the functional equation is solved by means of classical methods, with the solution itself often taking a closed form representation. In fact, problems of this type are ordinarily amenable to treatment with classical optimization methods to begin with, which means that dynamic programming can be bypassed altogether. For instance, consider the following continuous optimization problem.

8.2.1 Example

$$p := \max_{(x_1,\dots,x_N)} \sum_{n=1}^{N} \beta^{n-1} x_n^{1/2} \ , \ \beta > 0 \tag{8.1}$$

$$\sum_{n=1}^{N} x_n \leq r \ , \ r \geq 0 \tag{8.2}$$

$$x_n \geq 0 \ , \ n = 1, 2, \dots, N \tag{8.3}$$

Since the objective function is strictly increasing with respect to each decision variables, and the decision variables are continuous and allowed to take only non-negative values, (8.2) can be replaced by

$$\sum_{n=1}^{N} x_n = r \tag{8.4}$$

The problem thus turns out to be an equality-constrained optimization problem. Furthermore, as the objective function is *concave* and the constraint function is linear, the implication is that it can be solved for a global optimum with the classical *Lagrangian* method. The affiliated Lagrangian problem would have this form

$$p(\lambda) := \max_{(x_1,\dots,x_N)} \left\{ \sum_{n=1}^{N} \beta^{n-1} x_n^{1/2} + \lambda \left[\sum_{n=1}^{N} x_n - r \right] \right\} \ , \ \lambda \in \mathbb{R} \tag{8.5}$$

subject to (8.3). Thus, define

$$L(\lambda, x_1, \dots, x_N) := \sum_{n=1}^{N} \beta^{n-1} x_n^{1/2} + \lambda \left[\sum_{n=1}^{N} x_n - r \right] \ , \ \lambda \in \mathbb{R} \tag{8.6}$$

Equating the first partial derivative of L with respect to each one of its arguments to zero yields the following system of equations:

$$\frac{1}{2}\beta^{n-1} x_n^{-1/2} = -\lambda \ , \ n = 1, 2, 3, \dots, N \tag{8.7}$$

$$\sum_{n=1}^{N} x_n = r \tag{8.8}$$

Considering that (8.7) yields

$$x_n = \frac{\beta^{2(n-1)}}{4\lambda^2} \ , \ n = 1, 2, 3, \dots, N \tag{8.9}$$

it follows from (8.8) that

$$\sum_{n=1}^{N} \frac{\beta^{2(n-1)}}{4\lambda^2} = r \tag{8.10}$$

which in turn entails that

$$\lambda^2 = \frac{1}{r} \sum_{n=1}^{N} \beta^{2(n-1)} \tag{8.11}$$

Thus, (8.10)-(8.11) yield

$$x_n = \frac{r\beta^{2(n-1)}}{\displaystyle\sum_{k=1}^{N} \beta^{2(k-1)}} \tag{8.12}$$

so that

$$p = \sum_{n=1}^{N} \beta^{n-1} \left[x_n^*\right]^{1/2} \tag{8.13}$$

$$= \sum_{n=1}^{N} \beta^{n-1} \sqrt{\frac{r\beta^{2(n-1)}}{\displaystyle\sum_{k=1}^{N} \beta^{2(k-1)}}} \tag{8.14}$$

$$= \frac{(r^{1/2}) \displaystyle\sum_{n=1}^{N} \beta^{2(n-1)}}{\left[\displaystyle\sum_{n=1}^{N} \beta^{2(n-1)}\right]^{1/2}} \tag{8.15}$$

$$= \sqrt{r \sum_{n=1}^{N} \beta^{2(n-1)}} \tag{8.16}$$

$$= \begin{cases} \sqrt{rN} & , \beta = 1 \\ \sqrt{\dfrac{r(1-\beta^{2N})}{1-\beta^2}} & , 0 < \beta \neq 1 \end{cases} \tag{8.17}$$

You will recall that the same result was obtained in (5.109) by solving the dynamic programming functional equation associated with this problem. □

In short, although dynamic programming has the wherewithal to handle optimization problems with continuous decision variables, in this area it usually faces stiff competition from other (classical) optimization methods.

My objective in putting this matter this bluntly is obviously not to discourage the use of dynamic programming to solve problems of this kind.

Rather, my objective is to emphasize that the *point* about dynamic programming *is not* that its solution plan is well-suited for continuous problems, but that it is particularly suitable for combinatorial problems, namely problems with finite solution sets. Indeed, dynamic programming often turns out to be the only viable method that is able to address the particular requirements of such problems.

And yet, ironically, it is precisely here that dynamic programming proves so vulnerable. For in spite of the fact that its solution strategy yields a valid functional equation, frequently this equation proves thoroughly intractable due to dimensionality.

The strong grip that the *Curse of Dimensionality* continues to have on dynamic programming, and this in spite of the tremendous advances in computer hardware and software in recent years, raises an important question: Is the *Curse of Dimensionality* a fundamental problem, or is it plausible that new developments in computer technology, and/or the area of combinatorial optimization, will succeed to stamp it out, or at least mitigate its effect, so that it will cease to be a factor?

Let us examine this question in a more concrete setting.

8.3 Discrete Problems

Consider the case where *Problem P* is phrased as follows:

Problem P :

$$p := \underset{(x_1,\dots,x_M)}{\text{opt}} \; q(x_1,\dots,x_M) \ , \ X \subset X' = \overset{M}{\underset{m=1}{\times}} X_m \tag{8.18}$$

such that M is finite and for each $1 \leq m \leq M$, the set X_m is finite, so that X is also finite.

You will recall that we saw in *Chapter 4* that any such problem admits of a dynamic programming formulation, meaning that it can be brought to a functional equation of the following form:

$$f_n(s) = \underset{x\in D(n,s)}{\text{opt}} \; \rho(n,s,x,f_{n+1}(T(n,s,x))) \ , \ 1 \leq n \leq N, s \in S_n \tag{8.19}$$

where

$$f_N(s) := \underset{x\in D(n,s)}{\text{opt}} \; g_N(s,x) \ , \ s \in S_N \tag{8.20}$$

and the sets $\{S_n\}$ and $\{D(n,s)\}$ are finite.

Thus, (8.19)-(8.20) can be solved by *complete enumeration,* namely by generating all the triplets (n,s,x) such that $n \in \mathbb{N}, s \in S_n, x \in D(n,s)$ — in this order. Schematically, the procedure can be described as follows

8.3.1 Procedure

Step 1. Initialization.
Set $n = N$, enumerate the set S, and for each $s \in S$ enumerate the set $D(N,s)$ to determine the value of $f(s)$ in (8.20).

Step 2. Iteration.
For $n = N-1, N-2, \dots, 3, 2, 1$ — in this order — enumerate the set S and for each $s \in S_n$ enumerate the set $D(n,s)$ to determine the value of $f(s)$ in (8.19). □

Computer codes based on this procedure would therefore consist of three "loops". An outer "stage loop" that runs through the stages from N to 1; a middle "state loop" that for each $1 \leq n \leq N$ enumerates the set S_n and an inner "decision loop" that goes through the set $D(n,s)$ relative to each stage-state pair enumerated by the outer and middle loops, respectively.

Clearly then, a key indicator of the effect that the size of the sets $\{S_n\}$ and $\{D(n,s)\}$ has on the computations would be the number of times that the inner loop is executed in the procedure. So, to be able to establish of what order this factor can be, we need a counter to keep track of it, defined as follows:

$$K := \sum_{n\in\mathbb{N}} \sum_{s\in S_n} |D(n,s)| \tag{8.21}$$

where $|A|$ denotes the cardinality of set A.

The idea is to determine K's order of magnitude in the context of a large problem. And, to impress on the reader what a large problem is from a dynamic programming perspective, it is instructive to investigate this question in the framework of a problem with real-world content.

8.3.2 Example

Consider the following problem:

$$p := \min_{(x_1,..,x_N)} \sum_{n=1}^{N} C(Z_n, x_n) \tag{8.22}$$

$$\{x_1,\dots,x_N\} = \{1,2,\dots,N\} \tag{8.23}$$

$$Z_1 = \varnothing := \text{empty set} \tag{8.24}$$

$$Z_n = \{x_1,\dots,x_{n-1}\}\ ,\ n = 2,3,4,\dots,N \tag{8.25}$$

To impute real-world meaning to this problem we shall understand it to represent the following situation. On the agenda are N projects. The cost of implementing a project depends on certain characteristics of its predecessors. The task is to determine the order in which the projects should be implemented so as to minimize the overall implementation cost. Read in these terms the function C and the sets $\{Z_n\}$ have the following meaning:

$Z_n :=$	The set consisting of the first $(n-1)$ projects to be implemented.
$C(Z,j) :=$	The cost of implementing project j given that it is implemented immediately following the projects in Z.

It is important to note that the constraint (8.23) requires that each project be implemented once, and once only . Thus, it simply states that a feasible solution to the problem $(x_1,\dots,x_n)$ is a *permutation* of the n projects.

Now, viewed as a multistage decision problem this problem can be described by the following model:

$$S = \{s : s \subseteq \mathbf{J}\}, \mathbf{J} := \{1,2,\dots,N\} \tag{8.26}$$

$$\mathbb{D} = \mathbf{J} \tag{8.27}$$

$$S_1 = \{\varnothing\}, (\varnothing = empty\ set) \tag{8.28}$$

$$D(s) = \mathbf{J}\backslash s := \{j : j \in \mathbf{J},\ j \notin s\}\ ,\ s \in S \tag{8.29}$$

$$T(s,x) = s \cup \{x\}\ ,\ s \in S, x \in D(s) \tag{8.30}$$

$$g(s, x_1, \dots, x_N) = \sum_{n=1}^{N} C(s_n, x_n)\ ,\ s_n = T(s_{n-1}, x_{n-1})\ , 2 \leq n \leq N \tag{8.31}$$

Since the objective function is additive, the functional equation has this form:

$$f_n(s) = \min_{x \in D(s)} \{C(s,x) + f_{n+1}(T(s,x))\}\ ,\ 1 \leq n < N, s \in S_n \tag{8.32}$$

with

$$f_N(s) := \min_{x \in D(s)} C(s,x)\ ,\ s \in S_N \tag{8.33}$$

To establish then of what magnitude K can be in the case of this equation we do as follows. Observe that since $D(s) = \mathbf{J}\backslash s$ and $s \subset \mathbf{J}$, it follows that

$$\begin{align} |D(s)| &= |\mathbf{J} - \mathbf{s}| \tag{8.34} \\ &= N - |s| \tag{8.35} \end{align}$$

We therefore need to determine the composition of the sets $\{S_n\}$. Since by construction $S_1 = \{\varnothing\}$ it follows that

$$|s| = 0\ ,\ \forall s \in S_1 \tag{8.36}$$

More generally, because

$$\begin{align} S_{n+1} &= \{T(s,x) : s \in S_n, x \in D(s)\} \tag{8.37} \\ &= \{s \cup \{x\} : s \in S_n, x \in \mathbf{J}\backslash s\} \tag{8.38} \end{align}$$

and since $x \notin s, \forall x \in D(s)$, it follows that each element of S_{n+1} is the union of two disjoint sets such that one is a singleton and the other is an element of S_n. Therefore,

$$|s'| = 1 + |s|\ ,\ \forall s \in S_{n+1}, s \in S_n \tag{8.39}$$

As we have already established that $|s| = 0, \forall s \in S_1$, it follows that

$$|s| = n - 1\ ,\ \forall s \in S_n, n = 2, \dots, N \tag{8.40}$$

Hence,

$$\begin{align} |D(s)| &= N - n + 1\ ,\ \forall s \in S_n, n = 2, \dots, N \tag{8.41} \\ &= N - n + 1 \tag{8.42} \end{align}$$

Now, observe that for $n > 1$ the set S_n consists of all the subsets of $\mathbf{J}$ with each comprising exactly $n-1$ elements. Therefore,

$$|S| = \binom{N}{n-1} \quad , \quad \binom{m}{k} := \frac{m!}{(m-k)!k!} \tag{8.43}$$

Thus,

$$\sum_{s \in S_n} |D(s)| = (N-n+1)\binom{N}{n-1} \tag{8.44}$$

so that,

$$\begin{aligned}
K &= \sum_{n=1}^{N} (N-n+1)\binom{N}{n-1} && (8.45)\\
&= \sum_{n=1}^{N} N\binom{N-1}{n-1} && (8.46)\\
&= N\sum_{n=1}^{N} \binom{N-1}{n-1} && (8.47)\\
&= N\sum_{n=0}^{N-1} \binom{N-1}{n} && (8.48)\\
&= N\sum_{n=0}^{N-1} \binom{N-1}{n} && (8.49)\\
&= 2^{N-1}N && (8.50)\\
&\geq 2^N \ , \quad \textit{for } N \geq 2 \qquad \square && (8.51)
\end{aligned}$$

It is important to note that in the context of scheduling and sequencing problems, K would often be of the order of 2^N. As this figure is a frequently referred to indicator in combinatorial optimization, it is vital that we have a clear picture of what it can mean in real terms. To do this consider the values given in the following table.

N	10	20	50	100	200
2^N	1024	1.04×10^6	1.12×10^{15}	1.27×10^{30}	1.61×10^{60}

To appreciate the full force of what these figures tell us about the computational requirements involved in the solution of the dynamic programming functional equation, we need to examine them from the point of view of the computational capabilities of present-day computers. For simplicity, let us assume that our computer can execute 10^{10} inner loops per second. This means that such a computer will need 10^{40} seconds to execute 10^{50} inner loops. In other words, for $N = 50$ it will take about 10^{32} years to solve the problem in question.

But what is more, even if we assume that the next generation computer will be 10^{10} times faster than the present-day computer, it will still take about 10^{22} years to solve the functional equation with $K = 10^{50}$.

A far easier way to gauge the computational complexity of the dynamic programming functional equation would be to use a device that would keep track of the number of times the state-loop is executed. In this case K would be defined thus:

$$K := \sum_{n=1}^{N} |S_n| \tag{8.52}$$

Admittedly, the sets $\{D(n,s) : 1 \leq n \leq N, s \in S_n\}$ can vary quite significantly in size depending on the values of n and s. All the same, the value of K as given in (8.51)would still serve this purpose well.

Continuing with *Example 8.2.2,* from (8.43) and (8.51) it follows that

$$\begin{aligned}
K &= \sum_{n=1}^{N} \binom{N-1}{n-1} && (8.53)\\
&= \sum_{n=0}^{N-1} \binom{N-1}{n} && (8.54)\\
&= \sum_{n=0}^{N} \binom{N-1}{n} - \binom{N}{N} && (8.55)\\
&= 2^N - 1 && (8.56)\\
&\approx 2^N \ , \ N >>>> 1 \qquad \square && (8.57)
\end{aligned}$$

In short, the state space of dynamic programming problems can be gigantic.

8.3.3 Example

Consider the following nonlinear integer programming problem

$$p := \max_{x_{i,j}} \sum_{j=1}^{n} C_j\left(\sum_{i=1}^{m} c_{i,j}x_{i,j}\right) \tag{8.58}$$

$$\sum_{j=1}^{n} a_{i,j}x_{i,j} \leq b_i \ , \ i = 1,2,3,\dots,m \tag{8.59}$$

$$x_{i,j} \in \{0,1,2,3,\dots\} \ , \ i = 1,2,\dots,m; \ j = 1,2,\dots,n \tag{8.60}$$

where $\{a_{i,j}\}, \{c_{i,j}\}$ and $\{b_i\}$ are positive integers and $\{C_j\}$ are nonlinear real-valued functions, for instance

$$C_j\left(\sum_{i=1}^{m} c_{i,j}x_{i,j}\right) = \beta^{j-1} \exp\left(\sum_{i=1}^{m} c_{i,j}x_{i,j}\right) \ , \ j = 1,2,\dots,n \tag{8.61}$$

where $0 < \beta < 1$.

The modeling issues associated with problems of this type will be studied in *Chapter 11.* For now all we need to know is that the dynamic programming formulation of this problem would give rise to a state space of the form

$$S = \big\{s \in \mathbb{R}^m : s(i) \in \{0, 1, 2, \dots, b_i\}, i = 1, 2, 3, \dots, m\big\} \tag{8.62}$$

That is, a state would be a vector consisting of m non-negative integers. Thus,

$$|S| = \prod_{i=1}^{m}(1 + b_i) \tag{8.63}$$

If, for simplicity, we assume that $b_i = k - 1$ for all i, and furthermore that $S_n = S$, then (8.62) would yield

$$K = \sum_{j=1}^{n} S_n = n|S| = nk^m \tag{8.64}$$

Suffice it to say that, in practice, it is not uncommon to be presented with problems where $n = k = 100$ and $m = 20$, hence $|S| = 100 \times 100^{20} = 10^{42}$. In such cases the solution of the resulting dynamic programming functional equation would call for the solution of 10^{42} simple optimization problems!

The point made plain then by the above examples is that the nerve of the problem of dimensionality is that a profound disparity exists between a problem's size and the amount of computation engendered by it. Thus, even an equation stemming from a problem of moderate size can trigger a rapid build up in computations to quickly cause the dynamic programming algorithm to come to grief.

The inevitable conclusion therefore seems to be that the *Curse of Dimensionality* needs to be treated as a *pathologic problem* which, it would be safe to assume, even further advances in computer technology — as we know it today — and/or developments in the area of numeric methods, will be unable to root out.

Still, to keep things in perspective, we need to compare the computational complexity of dynamic programming algorithms with that of complete enumeration algorithms.

8.4 Special Cases

Prior to this I should point out that some combinatorial optimization problems whose general cases are notorious for subjecting dynamic programming equations to the *Curse of Dimensionality*, do have special cases that yield dynamic programming equations be can be managed quite easily.

For example, the well-known *Traveling Salesman Problem* (TSP) requires dynamic programming models whose state spaces increase rapidly ($|S| > 2^n$) with the number of cities (n). But there are cases where the topology of the locations of the cities is such that most of the feasible states of the dynamic programming model can be discarded a priori because they cannot be generated by an optimal tour (see for instance the problems studied by Balas [1999] and Balas and Simonetti [2002]).

Indeed, this fact suggests that special cases of this kind be used as approximations of large dynamic programming models. The dynamic programming inspired meta-heuristic *Corridor Method* (Sniedovich and Voß [2006]) is based on this idea (see *Appendix E*).

8.5 Complete Enumeration

A complete enumeration algorithm will simply go through **all** the feasible solutions of a problem with a view to identify the solution that optimizes the objective function.

Continuing with *Example 8.2.2,* suppose that we were to solve the problem under consideration by means of complete enumeration. Since a feasible solution to the problem is a *permutation* of the projects, it follows that

$$X = \{(x_1, \dots, x_N) : \{x_1, \dots, x_N\} = \{1, 2, 3, \dots, N\}\} \tag{8.65}$$

hence,

$$|X| = N! \tag{8.66}$$

In other words, the problem in question has $N!$ feasible solutions.

Again, to bring home the likely order of magnitude of (8.66), let us consider the following table.

N	10	20	50	100	200
2^N	1024	1.04×10^6	1.12×10^{15}	1.27×10^{30}	1.61×10^{60}
$N!$	3.6×10^7	2.4×10^{18}	3×10^{64}	9.3×10^{157}	7.8×10^{374}

Clearly, however profound the effect of the *Curse of Dimensionality* on dynamic programming's functional equation for this class of problems, dynamic programming is still fundamentally more efficient then complete enumeration in terms of *execution time.* On the other hand, complete enumeration has an advantage in terms of *memory* in that, contrary to dynamic programming's procedure requiring storage of a list of size 2^N, complete enumeration requires only a list of size N.

8.6 Conclusions

Notwithstanding the sound solution scheme that dynamic programming sets out for optimization problems — notably combinatorial problems — its use as a practical tool is often impeded owing to the functional equation's vulnerability to the *Curse of Dimensionality*.

Of course, todays computers enable solving problems that in the early days of dynamic programming were totally intractable due to size. All the same, despite the advances in computer technology, the *Curse of Dimensionality* continues to to be an obstacle, and once its pernicious character is fully comprehended the inevitable conclusion seems to be that there is very little reason to believe that it can be eliminated. It ought to be pointed out, though, in dynamic programming's defense, that many combinatorial optimization problems that are natural candidates for a dynamic programming treatment are classified as 'hard' to begin with (see Garey and Johnson [1979]).

It is also important to avoid concluding from all this "gloomy talk" that the *Curse of Dimensionality* has dealt dynamic programming a paralyzing blow. First, several types of problems, for example many graph and network problems, are immune to it. Second, the effort to find ways of coping with it is ongoing, and, as we shall see, these can be found. In *Chapter 12* I describe two methods that are designed to deal with dimensionality caused by non-separable objective functions.

Very broadly, my position on how to deal with the problem of dimensionality in dynamic programming is that it is best to adopt a non-confrontational or evasive approach. That is, the idea should be to approach it on a case by case basis with the view to identify structural features in the dimensionality prone problem which will allow working out a functional equation that will manage to circumvent dimensionality. My experience suggests that a general-purpose approach has a slim chance against the *Curse of Dimensionality*.

Lastly, a comment on a reoccurring misinterpretation of the term *dimensionality*. It is most unfortunate that this term is sometimes misconstrued to mean: *dimension of the state space*, namely the number of components comprising the state s.

The point to note here is that, although the number of dimensions that the state variable has can indeed be an important factor in the computational requirements of a dynamic programming algorithm, as such this has no bearing on the *size* of the state space.

And to illustrate, suppose that

$$S := \{0, 1\}^{10} \tag{8.67}$$

Then,

$$|S| = 2^{10} = 1024 \tag{8.68}$$

Here the state variables are impressively, 10-dimensional, yet the state space is extremely small.

In contrast, consider

$$S := \{1, 2, 3, \dots, 2^k\} \tag{8.69}$$

in which case $|S| = 2^k$. Here, even for a relatively small k, say $k = 50$, the state space S will be extremely large even though the dimension of the state is very small, in fact s is a scalar.

The inference therefore is that the problem of dimensionality cannot be expected to be alleviated simply by transforming a k-dimensional state space into a one-dimensional state space through a one-to-one mapping. This will only alter the representation of the states but not the cardinality of the state space.

And to end, it is important to note that the preceding discussion addresses the question of the *Curse of Dimensionality* from the standpoint of "exact" methods. Namely, from the standpoint where the objective is to find an exact solution to the dynamic programming functional equation concerned.

The *Curse of Dimensionality* is often tackled through the use of *heuristic* methods. The underlying objective of these methods is to find solutions to the dynamic programming functional equation that a priori settle for "good approximations" of the exact solutions (e.g. Heidari et al. [1971], Gal [1989], Sutton [1990], Sniedovich and Voß [2006], Powell [2007]).

But, the trouble here is that, in the absence of exact solutions, often it is difficult, indeed even impossible, to determine how good the heuristic solutions are.

For this reason it is inappropriate to claim that such methods in effect "resolve" the problem of *Curse of Dimensionality.* In fact, the opposite seems to be the case. The insurmountable difficulties encountered while attempting to determine the quality of the solutions generated by these heuristic approximation methods vividly illustrate that despite the unrelenting effort, the *Curse of Dimensionality* remains ... unbeaten!

In short, despite the enormous progress in this area over the past fifty years, the *Curse of Dimensionality* remains as elusive as ever!

9

The Rest Is Mathematics and Experience

9.1 Introduction

My investigation of the question *"what is dynamic programming?"* has now reached its half-way mark. It is appropriate therefore to interrupt it at this point in order to reflect on what I have done so far and to set the stage for the forthcoming discussions. In particular, I wish to explain in greater detail my decision to confine the discussion to the deterministic case, and to justify the reasoning behind my decision to forgo a detailed description and analysis of specialized solution schemes for the functional equation of dynamic programming, and of dynamic programming applications.

9.2 Choice of Model

You will recall that I prefaced the investigation with a brief discussion explaining that it would be conducted entirely within the perimeters of a simple deterministic model. I now come back to this point to reiterate that a correct interpretation of this matter is crucial for a correct understanding of my portrayal of dynamic programming's scope of operation, indeed its very nature.

It is important to keep in mind that my method of presentation does not in any way reflect on the extent of dynamic programming's scope and capabilities. Of course, dynamic programming is a highly versatile method, with an inherent ability to tackle a wide range of optimization problems. And this, as we have seen, is equivalent to saying that dynamic programming has an intrinsic capability to supply a wide range of models.

As I indicated at the outset, I adopted this model as a medium of discourse for two separate, yet intimately related, reasons. First, to present as clear and coherent a picture as possible of dynamic programming's idiom and techniques which, I argued, can best be done in a fixed framework of discussion — one that allows averting a constant accounting for of the idiosyncrasies of specific models.

Second, to impress on the reader the fundamental and universal character of this model which, I argued, is in the *conceptual core* provided by this model. This *conceptual core*, I claimed, can be extended in a number of directions, so as to enable the formulation of problems that are different from the one postulated by this model — stochastic problems, multiobjective problems, and so on.

I substantiate the validity of this position in the ensuing sections.

9.3 Dynamic Programming Models

Had I given the simple multistage decision model which, has served as my medium of discussion the title that it rightly deserves, I would have dubbed it a "deterministic, discrete-time, fixed-horizon, serial, single-objective model." This is another way of pointing at its prototype status.

The point I highlight in this chapter is that, as a prototype, this simple model provides a framework capable of being modified so as to meet the requirements of problems where certain characteristics need to be treated in a different manner than in the case of the problem underlying the prototype.

Thus, adjustments in this model would allow the formulation of problems in whose case the characteristic in question requires to be treated as stochastic rather than deterministic, continuous-time rather than discrete-time, non-serial rather than serial, multiple objective rather than single-objective, and of an unspecified horizon rather than a fixed horizon.

Naturally, these modifications would impact on the derivation of the respective functional equations, as in each case one would have to apply measures that are commensurate with the particular requirements of the formulation concerned. All the same, these would be more like variations on a theme.

On the whole, the basic dynamic programming approach would be present in every one of these derivation processes. The initial problem would be treated as being embedded in a so called family of modified problems, so that with the aid of a decomposition scheme the optimal solutions to these problems would be related to one another in a fashion that this relation would be reflected in the functional equation.

9.3.1 Stochastic Models

The deterministic character of our multistage decision model consists in the transition function T stipulating that the next state of the process, namely s_{n+1}, is determined entirely by the current stage, state and decision, that is, the triplet (n, s_n, x_n). Hence the expression, $s_{n+1} = T(n, s_n, x_n)$.

In contrast, in a *stochastic model*, the state s_{n+1} would be treated as a *random variable* whose (conditional) probability distribution function is uniquely determined by the triplet (n, s_n, x_n) for each s_n and $x_n \in D(n, s_n)$. The problem would then be to optimize the *expected value* of the return function — or some other measure of the return — in a manner that would expressly take account of the fact that the return is a random variable. Since in the stochastic case s_{n+1} is not uniquely determined by s_1 and $(x_1, \dots, x_n)$, the objective function would normally refer overtly to all the state and decision variables, namely $g = g(s_1, x_1, s_2, x_2, \dots, s_N, x_N, s_{N+1})$.

So, in view of the similarity of stochastic dynamic programming models to deterministic dynamic programming models, I do not go into the stochastic case, except for touching on it briefly in *Chapter 11*, *Chapter 14* and *Appendix D.* The following is a foretaste.

This is a typical dynamic programming functional equation of a stochastic problem:

$$f(s) = 1 + \min_{\substack{1 \leq x \leq s/2 \\ integer}} \left\{ \frac{2x}{s} f(x) + \frac{s-2x}{s} f(s-2x) \right\} \ , \ s = 2, \ldots, K \tag{9.1}$$

with $f(1) = 0$.

The story behind it is the famous *Counterfeit Coin Problem* (Sniedovich [2003]). Given K coins, where all, except one, have the same weight — the odd (counterfeit) coin is slightly lighter than the other coins — the task is to identify the counterfeit coin in the minimum expected number of weighings, using a balance beam.

In the above formulation s denotes the number of coins yet to be inspected, x denotes the number of coins on each side of the scale, and $f(s)$ the expected value of the number of weighing required by the optimal policy to identify the counterfeit coin, given that there are s coins to inspect.

Considering s and x, two outcomes are possible insofar as the next state is concerned. The counterfeit coin will be on the balance beam with probability $2x/s$. The counterfeit coin will be off the balance beam with probability $(s - 2x)/s$. The 1 preceding the min in (9.1) represents the "cost" of the current weighings.

For detailed discussions on stochastic dynamic programming see Bellman [1975a], Nemhauser [1966], Yakowitz [1969], White [1969,1983], Hinderer [1970], Bertsekas and Shreve [1978], Denardo [1982], Ross [1983]. For deterministic-like formulations of stochastic problems, see Denardo and Mitten [1967, p. 109].

9.3.2 Continuous-time Models

If the stage variable, n, is understood to denote time, then our model is a *discrete-time* model in that it depicts a situation comprising denumerable instances in which decisions are made. Continuous-time models, on the other hand, provide a framework for the formulation of optimization problems embodying situations where the process occurs over a *continuous* time interval, say $[t_1, t_2]$, $t_1 < t_2$.

In such models the objective function is normally an *integral* and the transition function a differential equation. The objective function would therefore be additive in that an integral is separable and additive over its domain.

Continuous-time problems are sometimes called *optimal control* problems, although there are also discrete-time optimal control problems. Also, many calculus of variation problems fall in the category of continuous-time problems.

Studies of dynamic programming's treatment of continuous-time problems can be found in Bellman [1957a, 1961, 1970], Dreyfus [1965], Nemhauser

[1966], Beckman [1968], Larson [1968], White [1969], Boudarel et al. [1971], Kamien and Schwartz [1981], Larson and Casti [1982] and Whittle [1983].

In *Chapter 13* I briefly discuss *Pontryagin's Maximum Principle* in the context of a multistage decision model.

9.3.3 Non-serial Models

The distinctive feature of our model is that it involves a single set of stages. Furthermore, the stages stand in a *serial* relation to one another, that is, the process is understood to always progress from stage n to stage $n+1$. *Non-serial* models, on the other hand, provide for situations involving a number of serial subprocesses that are linked in a variety of combinations.

For example, a non-serial model can consist of a main serial process and two serial subprocesses branching out of it and then converging back into it.

The basic characteristic of non-serial models is that the transition function T yields a set of states rather than a single state. That is, in such models $T(n, s, x)$ is a subset of S rather than an element of S. The objective function is modified accordingly to account for this feature of the underlying process.

There are four generic configurations to depict the interaction between the branches generated by non-serial problems:

· Diverging branches
· Converging branches
· Feed-forward branches
· Feed-backward branches

The most common — and most simple — case is that where all the branches diverge. Stochastic dynamic programming models are instances of this generic case (see a short discussion on this topic in §*14.11*). However, there are also deterministic models of this type. For example, the dynamic programming model for the *Towers of Hanoi* problem (Sniedovich [2002]) that I discuss in *Chapter 16* is a deterministic diverging non-serial model.

The strategy deployed by dynamic programming in the more general case is to break up the non-serial model into a number of serial models, analyze each model separately in the usual dynamic programming fashion, and then rejoin them back together again. Depending on the architecture of the non-serial model, the introduction of additional state and decision variables may be required to facilitate the rejoining of the subprocesses.

In *Chapter 11* I give a terse description of a simple non-serial dynamic programming model by illustrating the solution of the famous *Chained Matrix Product* problem.

Discussions on the formulation and analysis of non-serial dynamic programming models can be found in Nemhauser [1966], Bertele and Brioschi [1972], Esogbue and Marks [1972, 1974, 1977], Pope et al. [1982].

9.3.4 Preference Order Models

Our basic multistage decision model can also be adapted to allow for the treatment of preference order optimization problems: problems where the goal is to decide between alternative courses of action and where the desirability of the alternatives is dictated not by a real-valued objective function, but by a more general preference relation.

The most important preference model is no doubt the one consisting of a multiple-criteria objective function and a standard vector dominance criterion in $\mathbb{R}^k$.

The solution to the dynamic programming functional equation deriving from such a model yields the set of nondominated (Pareto) solutions to the multiobjective problem. Similarly, if the objectives can be ranked in a *lexicographic* order of importance, then solving the dynamic programming functional equation will yield the optimal lexicographic solution.

Analyses of general preference order dynamic programming models can be found in Ellis [1955], Brown and Strauch [1965], Mitten [1974], Clement [1975], and Sobel [1975]; and, of vector optimization in Tauxe et al. [1979], Furukawa [1980], White [1980, 1982a, 1982b], Daellenbach and De Kluyver [1980], Villarreal and Karwan [1982], Henig [1983], Nollau [1985].

9.3.5 Others

Obviously this is an incomplete list. The basic dynamic programming model can be modified in other directions as well. For instance, it can be reformulated so as to embody **fuzzy sets problems**. On this subject see Bellman and Zadeh [1970], Bellman and Lee [1978, pp. 11-13], Baldwin and Pilsworth [1982], Esogbue [1986].

To conclude this section I should point out though that in the second half of the book I devote a whole chapter — *Chapter 14* — to a discussion on the *forward dynamic programming* model which, on the logic of my argument need not have been included in this book as, it is also a natural extension of the simple model that I set out in the first part. The reasons for my going into it at length are discussed in *Chapter 14*. So, to whet the reader's appetite as to what is in store I give a concise formulation of this model below.

9.4 Forward Decomposition Models

The decomposition scheme that we employed thus far in the derivation of the dynamic programming functional equation is a *backward scheme*. Backward, in the sense that the modified objective functions are related to one another in a backward fashion:

$$g_n(s, x_n, x_{n+1}, \ldots, x_N) = \rho(n, s, x_n, g_{n+1}(s', x_{n+1}, x_{n+2}, \ldots, x_N))) \quad (9.2)$$

for $n = 1, 2, \ldots, N-1$, where $s' = T(n, s, x_n)$. That is, g_n is expressed in terms of g_{n+1}.

The distinctive characteristic of the backward decomposition is then that the arguments of the modified objective functions are expanded in a backward manner, namely one appends x_n to the sequence $(x_{n+1}, x_{n+2}, \ldots, x_N)$ to obtain the sequence of decisions pertaining to the modified objective function associated with stage n.

So, dynamic programming models invoking such decomposition schemes are called *backward* models and the functional equations deriving from such models are called *backward* functional equations.

The distinguishing characteristic of a *forward* model, is that here the modified objective function pertaining to stage n is expressed in terms of the modified objective function associated with stage $n-1$. Here, one appends x_n to the sequence $(x_1, x_2, \ldots, x_{n-1})$ to obtain the sequence of decisions pertaining to the modified objective function associated with stage $n+1$.

The following is a typical forward dynamic programming functional equation. It is the forward counterpart of the backward functional equation derived for the knapsack problem in *Example 5.2.2.*

$$f_{n+1}(s) = \max_{\substack{0 \le x \le s/b_n \\ x \textit{ integer}}} \{x a_n + f_n(s + x b_n)\} \quad (9.3)$$

for $n = 2, 3, \ldots, N, s = 0, 1, 2, \ldots, v$, with $f_1(s) = 0, \forall s$.

Recall that the backward functional equation for the same problem is as follows

$$f_n(s) = \max_{\substack{0 \le x \le s/b_n \\ x \textit{ integer}}} \{x a_n + f_{n+1}(s - x b_n)\} \quad (9.4)$$

for $n = 1, 2, 3, \ldots, N-1, s = 0, 1, 2, \ldots, v$, with $f_{N+1}(s) = 0, \forall s$.

In *Chapter 14* I explain the relationship between backward and forward dynamic programming models.

9.5 Practice What You Preach!

Having given a rough sketch of the types of models that, as we saw, are natural outgrowths of the simple deterministic model, I now proceed to justify what, to certain readers may no doubt, appear as strangely missing from this book.

One imagines that the general expectation amongst potential readers of a text aiming to expound dynamic programming would be that such a text would dwell at length on the description and analysis of schemes designed expressly to improve the efficiency of dynamic programming algorithms, and on accounts outlining how dynamic programming is employed to solve certain types of optimization problems, say sequencing and scheduling, networks, inventory, and so on. Such an expectation seems to be suggested by the fact that these topics dominate the dynamic programming literature. It is necessary therefore that I justify my position in this regard because, in this case as well, it is important to prevent the wrong conclusions from being drawn.

Again, my reasons are similar to those behind my decision not to extend the analysis to dynamic programming's treatment of stochastic problems, multiobjective problems, and so on.

My position is that no study, however thorough, comprehensive and elegant, of this or that dynamic programming application, or of this or that computational scheme for an efficient solution of this or that functional equation, will furnish a clear idea of the *basic nature of dynamic programming.* And by this I mean — as I have been arguing all the while — a clear idea of dynamic programming's understanding of an optimization problem, the strategy that it puts forward to tackle it, and the means that it supplies for this purpose. To achieve this end one *has to* focus on precisely these issues. I have already done this to a considerable extent in the first part of the book. In the second part I go into these matters more deeply by concentrating on the meaning and role of the *Principle of Optimality* and the *state variable* in the method, and by delineating an approach to problem modeling and formulation that is particularly tailored for the needs of dynamic programming.

Still, I think it my duty to refer the reader to the relevant publications on the topics of computational schemes and applications. I therefore provide lists of references for these two topics in the next two sections, and I comment very briefly on the backgrounds against which these literatures evolved.

9.6 Computational Schemes

In sharp contrast to the early days of dynamic programming, when the bulk of the literature on the solution of the functional equation was devoted to analytic methods and simple numeric methods, the last fifty years have seen a steady stream of publications all proposing techniques, algorithms and even plain tricks, for a numeric solution of the equation, with the emphasis being placed on *efficiency.* The driving force behind this effort has been the growing recognition of dynamic programming particular suitability for the treatment of combinatorial problems, countervailed by the increasing appreciation of the equation's vulnerability to the *Curse of Dimensionality.*

It is interesting to note that, from the start, Bellman pointed out that dynamic programming offers a solution strategy that is uniquely appropriate for combinatorial problems. And while remaining fully aware of the magnitude of the problem of dimensionality, he nevertheless expressed the hope that the growing capabilities of the computer would increasingly facilitate the solution of problems of substance.

In later years, however, as it became clear that despite the huge strides in computer hardware and software, the *Curse of dimensionality* remained a serious impediment, the push to find ways and means to efficiently solve the functional equation started gaining momentum.

The discussion in *Chapter 12* on hybrid algorithms based on collaboration schemes between dynamic programming and other optimization methods, as well as the topic of "heuristic" approaches on which I commented briefly in *Chapter 8* and discuss in *Appendix E* are pertinent to this issue.

For a sample of works in this area the reader is referred to the following publications: Mayne [1966], Larson [1967,1968], Wong and Luenberger [1968], Jacobson and Mayne [1970], Larson and Korak [1970], Wong [1970a,1970b], Heidari et al. [1971], Howson and Sancho [1975], Morin and Marsten [1976], Morin [2977], Schrage and Baker [1978], Denardo and Fox [1979a,1979b], Murray and Yakowitz [1979,1981], Lawler [1979], Yakowitz and Rutherford [1982], Yao [1982], Bellman and Roosta [1982], Denardo [1982, 2003], Ozden [1983], Axsater [1983], Yakowitz [1983,1985], Carraway and Morin [1988, 1990], Gal [1989], Carraway and Schmidt [1991], Bertsekas and Tsitsiklis [1996], Liu and Stoller, [2003], Popova-Zeugmann [2006], Hartman and Perry [2006], Sniedovich and Voß [2006], Wilbaut [2006], Garcia de la Banda and Stuckey [2007], Powell [2007], Puchinger and Stuckey [2008]. Other references relevant to this subject are listed in the bibliography.

I should also note that in *Chapter 15* I take up an algorithm that has become one of the most widely-used algorithms in *Operations Research* and *Computer Science.* This is *Dijkstra's Algorithm.* Surprisingly, this algorithm is not known outside the area of dynamic programming for what it actually is: *a dynamic programming algorithm.*

So, in *Chapter 15* I give a dynamic programming perspective on this powerful algorithm showing that it is in effect a dynamic programming successive approximation method.

9.7 Applications

The term 'application' poses a certain difficulty. The difficulty arises from the different meanings that, one assumes, would be attributed it by the academic as opposed to the practitioner. For, judging by the literature produced by academics, an application from this point of view seems to designate a proposition to use dynamic programming to solve what is deemed by the academic to be a ‘real-world problem’, which is supported by an example illustrating the solution strategy. This, however, may not strike the practitioner as an application. The practitioner, presumably, would think of an application more along the lines of an implementation. A dynamic programming strategy at work in a certain real-world setting about which one would report *a posteriori.* As far as can be established, no surveys covering dynamic programming applications in this sense of the word are available. The reader is referred, however, to White's [1985] survey: *Real Applications of Markov Decision Processes.*

I might add that

· Two of the finalist papers selected for the 1985 Edelman Award for Management Science Achievement (an award which recognizes accomplishments in the *application* of operations research methods) present dynamic programming applications (see Lembersky and Chi [1986], and Terry et al. [1986]).

· The INFORMS 2009 *Daniel H. Wagner Prize for Excellence in Operations Research Practice* was awarded to Abraham George, Warren Powell, Hugo Simao, Jeff Day, Ted Gifford, and John Nienow for their project *Approximate Dynamic Programming Captures Fleet Operations for Schneider National.*

And as a final note I should point out that the widely used *Critical Path Method* (CPM) is a dynamic programming application par excellence. It is unfortunate therefore that in much of the literature this fact is not made explicit. The same is true, by the way, for *Decision Trees*.

More on dynamic programming applications can be found in the bibliography.

Turning now to the term “application” as it is understood by the academic. If the term “application” is taken to have the meaning ascribed it by the academic, then the list of dynamic programming applications is too long to cite; and this by the way is said without a trace of cynicism. Indeed, the rich literature on dynamic programming (academic) applications is further testimony of dynamic programming's ability to handle a wide range of problems, provided of course that “size” is no impediment.

The first descriptions of dynamic programming applications would of course be found in Bellman's early work (see bibliography). For later ones the reader is referred to Kaufman and Cruon's [1967] book, and to Yakowitz's [1982] survey on dynamic programming applications in water resources planning and management.

An examination of the optimization and operations research literatures reveals that dynamic programming had been incorporated into almost every new trend in these two disciplines. For instance, in the nineteen-seventies, when interest started to develop in multiobjective optimization, dynamic programming models were used to formulate and solve preference order problems; and in the early nineteen-eighties, the dynamic programming approach was applied to fuzzy optimization problems and to problems in the area of artificial intelligence, e.g. Bolt [1984, p. 36, p. 42], Warwick and Phelps [1986], and Russell et al. [1986]. In more recent years, dynamic programming had become an important method in the area of Reinforcement Learning (see for example the program of the *2009 IEEE International Symposium on Adaptive Dynamic Programming and Reinforcement Learning,* March 31- April 2, 2009, Nashville, TN, USA.

Other recent applications of dynamic programming had, unsurprisingly, been in the area of *bioinformatics,* e.g. Sagot and Walter [2007]. An interesting overview of the evolution of dynamic programming in this area can be found in Sankoff [2000].

And, finally I should of course mention that dynamic programming figures in the work of Finn Kydland and Edward Prescott who were awarded the 2005 Nobel Prize in *Economics* for their contribution to dynamic macroeconomics. See, for example, Kydland and Prescott [1977, 1982, 1988].

9.8 Dynamic Programming Software

It appears that, apart from a few codes designed for some very specific dynamic programming problems — (e.g. Hastings [1975], Broin and Morin [1983], Jensen [1986], Parlar [1986], Ho [1987a, 1987b], Lin and Bricker [1990], Labadie [1990], Lubow, [1995], Raffensperger and Richard [2005], Sklenar and Popela [2005, 2006], Mauch [2006], Lew and Mauch [2007], Jensen and Bard [2008], Kátai and Csiki [2009]) — no "truly" general-purpose dynamic programming computer codes seem to be available commercially.

This, however, is not surprising, considering that, as we have seen, the functional equation stemming from a dynamic programming formulation requires handling on a case by case basis. Hence, efficient, truly general-purpose, dynamic programming codes can hardly be envisioned.

I take this opportunity to call attention to the powerful computer language *APL* which I found to be extremely effective for developing dynamic programming algorithms, for conducting numerical experimentation and for teaching dynamic programming. The great merit of *APL* is that it consists of a highly eloquent yet succinct mathematical notation that is directly machine executable. It thus enables conversing with the computer in the traditional style of mathematics. More on the use of APL in dynamic programming can be found in Sniedovich [1981], Crowder [1982], Mcleod [1983], Bricker [1985], and Lin and Bricker [1990].

I also wish to point out that some of the features of the package *Mathematica* (see Wolfram [1988]) especially those pertaining to recursive computation are extremely attractive from a dynamic programming standpoint. Still, the resulting recursive codes would be unable to cope with large problems. Dynamic programming also features as one of the major topics in Bird and de Moor's [1997] book *Algebra of Programming.*

And to conclude, a brief note on the computer language *JavaScript* (not Java!). In the past fifteen years or so I had used *JavaScript* extensively to develop online, web-based, interactive modules for a variety of dynamic programming models[1]. I found this language a surprisingly effective tool for online web applications.

9.9 Summary

The point I reinforce in this chapter ties in with an often encountered characterization of dynamic programming which portrays the method as art more than science, an approach more than a technique. Based on what we have seen so far, the idea here is that dynamic programming in effect amounts to a certain way of looking at problems, and a certain way of describing them. Or, as I have been stressing all along, it amounts to formulating an optimization problem in terms of a multistage decision model and in bringing it to the form of a functional equation.

The implication is then that explaining what dynamic programming is, *does not* amount to elucidating this or that efficient computational scheme for this or that functional equation, or outlining this or that application, but to making clear *how* dynamic programming formulates a problem in terms of a multistage decision model, and *how* it brings it to the form of a functional equation.

To do this effectively one needs a medium of discourse that enables identifying all the ingredients of the method without digressing into peripheral matters. Such a medium is provided by the simple multistage decision model outlined in the first part. I also pointed out that as a prototype formulation, this model furnishes the base for its extension in various directions.

In the second part of the book I single out for closer scrutiny a number of topics (for instance the state and how to formulate it) that are key to gaining better insight into the dynamic programming approach to problem solving.

To sum up then, my approach to dynamic programming, notably to its exposition, accords with Bellman's thesis that what really counts in dynamic programming is a good grasp of the essentials of the method because ... *"The rest is mathematics and experience".*

[1]See the tutORial website at http://www.ifors.ms.unimelb.edu.au/tutorial/

Part II

Art

10

Refinements

10.1 Introduction

In this introductory chapter to the *Art of Dynamic Programming* I round out the discussion on two matters that have immediate consequences for modeling and formulation in dynamic programming style: the *Markovian Condition*, and the *decomposition scheme*.

First, I show that a weaker version of the Markovian Condition equally ensures the validity of the functional equation, with the caveat, that the equation may not recover *all* the optimal solutions. This provides the justification for formulating a problem in terms of a Weak-Markovian model should such a need arise.

Next, I consolidate the concept of the decomposition scheme by arranging it according to five major categories. This furnishes a complete account of the types of schemes that one can expect to use in dynamic programming.

Finally, in the second part of the chapter I define a prototype sequential decision model, which as we shall see, enables a graphic formulation of discrete optimization problems.

10.2 Weak-Markovian Condition

Recall that, as we saw in *Section 4.3*, the validity of the dynamic programming functional equation is assured by the Markovian Condition, that is,

$$X^*(n,s,x) = X^*(n+1, T(n,s,x)) \tag{10.1}$$

for all $1 \leq n < N, s \in S_n, x \in D(n,s)$, which obviously implies that

$$X^*(n,s,x) \subseteq X^*(n+1, T(n,s,x)) \tag{10.2}$$

for all $1 \leq n < N, s \in S_n, x \in D(n,s)$, where $X^*(n,s)$ denotes the set of optimal solutions to *Problem* $P(n,s)$ and $X^*(n,s,x)$ denotes the set of optimal solutions to *Problem* $P(n,s,x)$.

In this section I show that it is sufficient that a weaker version be satisfied for the dynamic programming functional equation to be valid. To see that this is so consider the following:

Definition 10.2.1 *A decomposition scheme* (ρ, G) *is said to be* WEAK-MARKOVIAN *if*

$$X^*(n,s,x) \cap X^*(n+1, T(n,s,x)) \neq \varnothing \tag{10.3}$$

for all $1 \leq n < N, s \in S_n, x \in D(n,s)$*, where* $\varnothing$ *denotes the empty set.*

That is, a decomposition scheme is Weak-Markovian if for each triplet (n, s, x) such that $1 \leq n < N$, $s \in S_n$ and $x \in D(n, s)$, *Problem* $P(n, s, x)$ has at least one optimal solution that is also optimal for the modified problem induced by (n, s, x), namely *Problem* $P(n+1, T(n, s, x))$. Needless to say, any Markovian decomposition scheme by necessity has the Weak-Markovian property.

To show that such a decomposition scheme will yield a valid dynamic programming functional equation, let (n, s, x) be any triplet such that $1 \leq n < N$, $s \in S_n$, and $x \in D(n, s)$, and set $s' = T(n, s, x)$.

Then, by definition, there exists a $z \in X(n+1, s')$ such that $z \in X^*(n, s, x)$ and $z \in X^*(n+1, s')$. This implies in turn that there exists a $z \in X(n+1, s')$ such that

$$g_n(s, x, z) = f_n(s, x) := \operatorname*{opt}_{y \in X(n+1, s')} g_n(s, x, y) \tag{10.4}$$

and

$$g_{n+1}(s', z) = f_{n+1}(s') := \operatorname*{opt}_{y \in X(n+1, s')} g_{n+1}(s', y) \tag{10.5}$$

Now, because (ρ, G) is a decomposition scheme, we have,

$$g_n(s, x, z) = \rho(n, s, x, g_{n+1}(s', z)) \tag{10.6}$$

so that (10.4)-(10.6) entail that

$$f_n(s, x) = \rho(n, s, x, f_{n+1}(T(n, s, x))) \tag{10.7}$$

Thus, *Corollary 4.4.1*, namely,

$$f_n(s) = \operatorname*{opt}_{x \in D(n, s)} f_n(s, x) \ , \ \forall 1 \leq n \leq N, s \in S_n \tag{10.8}$$

in conjunction with (10.7), yield the following.

Lemma 10.2.1 — *Functional equation:*
If the decomposition scheme (ρ, G) *is Weak-Markovian, then*

$$f_n(s) = \operatorname*{opt}_{x \in D(n, s)} \rho(n, s, x, f_{n+1}(T(n, s, x))) \ , \ \forall 1 \leq n < N, s \in S_n \tag{10.9}$$

Having established this, I now proceed to illustrate the purport of the Markovian and Weak-Markovian conditions. I shall do this in the context of the following optimization problem:

$$\textit{Problem Q:} \quad p := \operatorname*{opt}_{\substack{b \in B(a) \\ a \in A}} q(a, b) \tag{10.10}$$

where q is a real-valued function on some set $A \times B'$ and $B(a) \subseteq B'$, $\forall a \in A$.

Assume that there exist a real-valued function r on B' and a real-valued function μ on $A \times \mathbb{R}$ such that

$$q(a,b) = \mu(a, r(b)) \ , \ \forall a \in A, b \in B(a) \tag{10.11}$$

This implies that *Problem Q* can be restated as follows:

$$\textit{Problem Q:} \quad p := \underset{\substack{b \in B(a) \\ a \in A}}{\text{opt}} \ \mu(a, r(b)) \tag{10.12}$$

An appeal to the *Principle of Conditional Optimization* yields

$$p = \underset{a \in A}{\text{opt}} \left\{ \underset{b \in B(a)}{\text{opt}} \ \mu(a, r(b)) \right\} \tag{10.13}$$

The conditional problem and modified problem deriving from $a \in A$ are:

$$\textit{Problem } Q, a \in A: \quad v(a) := \underset{b \in B(a)}{\text{opt}} \ \mu(a, r(b)) \tag{10.14}$$

$$\textit{Problem } R(a), a \in A: \quad w(a) := \underset{b \in B(a)}{\text{opt}} \ r(b) \tag{10.15}$$

Let $Q^*(a)$ and $R^*(a)$ denote the sets of optimal solutions to *Problem* $Q(a)$ and *Problem* $R(a)$ respectively. We say then that the decomposition scheme (μ, r) is Markovian if $Q^*(a) = R^*(a)$, $\forall a \in A$, and that it is Weak-Markovian if $Q^*(a) \cap R^*(a) \neq \varnothing$, $\forall a \in A$. To illustrate, suppose that A and B' are subsets of $\mathbb{R}$,

$$q(a,b) = q_1(a) \times q_2(b) \ , \ (a,b) \in A \times B' \tag{10.16}$$

and q_1 and q_2 are real-valued functions on A and B' respectively. Hence,

$$r(b) = q_2(b) \ , \ b \in B' \tag{10.17}$$

$$\mu(a,z) = q_1(a) \times z \ , \ a \in A, z \in \mathbb{R} \tag{10.18}$$

so that $q(a,b) = \mu(a, r(b))$, $\forall a \in A$, $b \in B(a)$. Three cases need to be considered.

Case 1: $q_1(a) > 0$, $\forall a \in A$.

Assume that opt = max and let a be any element of A and b^* any element of $Q^*(a)$. Then, by definition,

$$\mu(a, r(b^*)) \geq \mu(a, r(b)) \ , \ \forall b \in B(a) \tag{10.19}$$

which implies that

$$q_1(a) \times r(b^*) \geq q_1(a) \times r(b) \ , \ \forall b \in B(a) \tag{10.20}$$

Since $q_1(a) > 0$, dividing both sides of this inequality by $q_1(a)$ gives

$$r(b^*) \geq r(b) \ , \ \forall b \in B(a) \tag{10.21}$$

This implies that $b^* \in R^*(a)$. It follows therefore that $Q^*(a) \subseteq R^*(a)$. Conversely, let a be any element of A and let b^* be any element of $R^*(a)$ so that by definition,

$$r(b^*) \geq r(b) \ , \ \forall b \in B(a) \tag{10.22}$$

As $q_1(a) > 0$, multiplying both sides of this inequality by $q_1(a)$ yields

$$q_1(a) \times r(b^*) \geq q_1(a) \times r(b) \ , \ \forall b \in B(a) \tag{10.23}$$

Hence,

$$\mu(a, r(b^*)) \geq \mu(a, r(b)) \ , \ \forall b \in B(a) \tag{10.24}$$

the implication being that $b^* \in Q^*(a)$, and $R^*(a) \subseteq Q^*(a)$. It follows therefore that $Q^*(a) = R^*(a)$, $\forall a \in A$. The conclusion to be drawn is that here (μ, r) is a Markovian scheme which means that

$$v(a) = q_1(a) \times w(a) \ , \ \forall a \in A \tag{10.25}$$

$$= \mu(a, w(a)) \tag{10.26}$$

Case 2: $q_1(a) \geq 0$, $\forall a \in A$.

Reasoning along the lines of Case 1, we say that $Q^*(a) = R^*(a)$ for any $a \in A$ such that $q_1(a) > 0$. Consider then any $a^\circ \in A$ such that $q_1(a^\circ) = 0$. Since μ is multiplicative, clearly all the feasible solutions to *Problem* $Q(a^\circ)$ must be optimal in this case. Namely,

$$v(a^\circ) = q_1(a^\circ) \times q_2(b) \ , \ \forall b \in B(a^\circ) \tag{10.27}$$

$$= 0 \times q_2(b) \tag{10.28}$$

$$= 0 \tag{10.29}$$

Because by definition $R^*(a)$ is always a subset of $B(a)$, and as indicated above $Q^*(a^\circ) = B(a^\circ)$, it follows that $Q^*(a^\circ) \cap R^*(a^\circ) \neq \varnothing$. Therefore, the scheme (μ, r) is Weak-Markovian and

$$v(a) = \mu(a, w(a)), \forall a \in A \tag{10.30}$$

Observe that unless $R^*(a) = B(a)$ for all $a \in A$ such that $q_1(a) = 0$, this scheme will not be Markovian.

Case 3: q_1 takes both positive and nonnegative values on A.

Let a be any element of A such that $q_1(a) < 0$ and let b^* be any element of $Q^*(a)$ so that by definition

$$q_1(a) \times q_2(b^*) \geq q_1(a) \times q_2(b) \ , \ \forall b \in B(a) \tag{10.31}$$

Since $q_1(a) < 0$, it follows that

$$q_2(b^*) \leq q_2(b) \ , \ \forall b \in B(a) \tag{10.32}$$

Therefore, unless $q_2(b) = q_2(b')$ for all $b, b' \in B(a)$, b^* is clearly not an element of $R^*(a)$. Consequently, generally (μ, r) is not even Weak-Markovian in this case and furthermore

$$v(a) \neq \mu(a, w(a)) \tag{10.33}$$

for $a \in A$ such that $q_1(a) < 0$ $\square$

In view of what we have seen so far consider the following:

Definition 10.2.2 *A decomposition scheme (ρ, G) is said to be* MONOTONE *if ρ is monotone increasing with respect to its last argument. Specifically, (ρ, G) is a* MONOTONE DECOMPOSITION SCHEME *if it meets the following requirement. Let (n, s, x) be any triplet such that $1 \leq n < N$, $s \in S_n$, and $x \in D(n, s)$, and set $s' = T(n, s, x)$. Also let (z, z') be any pair such that $z, z' \in X(n+1, s')$ and set $a = g_{n+1}(s', z)$ and $a' = g_{n+1}(s', z')$. Then,*

$$a \geq a' \Rightarrow \rho(n, s, x, a) \geq \rho(n, s, x, a') \tag{10.34}$$

Similarly, (ρ, G) is STRICTLY MONOTONE *if*

$$a > a' \Rightarrow \rho(n, s, x, a) > \rho(n, s, x, a') \tag{10.35}$$

Clearly, there is a close kinship between the Markovian and monotonicity properties. This kinship can be summed up as follows.

Theorem 10.2.1 *Any monotone decomposition scheme is Weak-Markovian and any strictly monotone decomposition scheme is Markovian.*

PROOF. Let (ρ, G) be any decomposition scheme, let (n, s, x) be any triplet such that $1 \leq n < N$, $s \in S_n$ and $x \in D(n, s)$, and set $s' = T(n, s, x)$. Also let z be any element of $X^*(n, s, x)$ and let z' be any element of $X^*(n+1, s')$. As a consequence of (ρ, G) it follows that,

$$g_n(s, x, y) = \rho(n, s, x, g_{n+1}(s', y)) \ , \ \forall y \in X(n+1, s') \tag{10.36}$$

so that

$$g_n(s, x, z) = \rho(n, s, x, g_{n+1}(s', z)) \tag{10.37}$$

Next, assuming that opt = max, then $z \in X^*(n, s, x)$ entails that

$$g_n(s, x, z) \geq g_n(s, x, y) \ , \ \forall y \in X(n+1, s') \tag{10.38}$$

and as $z' \in X^*(n+1, s')$ implies that $z' \in X(n+1, s')$, it follows that

$$g_n(s, x, z) \geq \rho(n, s, x, g_{n+1}(s', z')) \tag{10.39}$$

Thus, (10.39), in conjunction with (10.37), yield

$$\rho(n, s, x, g_{n+1}(s', z)) \geq \rho(n, s, x, g_{n+1}(s', z')) \tag{10.40}$$

On the other hand, $z' \in X^*(n+1, s')$ entails that

$$g_{n+1}(s', z') \geq g_{n+1}(s', y) \ , \ \forall y \in X(n+1, s') \tag{10.41}$$

and because $z \in X^*(n, s, x)$ implies that $z \in X(n+1, s')$, it follows that

$$g_{n+1}(s', z') \geq g_{n+1}(s', z) \tag{10.42}$$

Part 1: (ρ, G) is monotone. In this case (10.42) indicates that

$$\rho(n, s, x, g_{n+1}(s', z')) \geq \rho(n, s, x, g_{n+1}(s', z)) \tag{10.43}$$

so that from (10.40), in conjunction with (10.43), we have

$$\rho(n, s, x, g_{n+1}(s', z)) = \rho(n, s, x, g_{n+1}(s', z')) \tag{10.44}$$

which in turn implies that

$$g_n(s, x, z) = g_n(s, x, z') \tag{10.45}$$

Considering that $z \in X^*(n, s, x)$, it follows that $z' \in X^*(n, s, x)$. Thus, z' being an element of $X^*(n+1, s')$, it is clear that $X^*(n, s, x) \cap X^*(n+1, s') \neq \varnothing$. The conclusion is therefore that the scheme is Weak-Markovian.

Part 2: (ρ, G) is strictly monotone. Having seen in the demonstration of *Corollary 4.7.1* that

$$X^*(n, s, x) \subseteq X^*(n+1, T(n, s, x)) \Rightarrow X^*(n, s, x) = X^*(n+1, T(n, s, x)) \tag{10.46}$$

it is sufficient to show that $z \in X^*(n, s, x)$ implies that $z \in X^*(n+1, s')$. Assume then the opposite, namely, assume that $z \notin X^*(n+1, s')$. In this case it would follow that $g_{n+1}(s', z') > g_{n+1}(s', z)$. Since the decomposition scheme is strictly monotone the inference is that

$$\rho(n, s, x, g_{n+1}(s', z')) > \rho(n, s, x, g_{n+1}(s', z)) \tag{10.47}$$

Therefore (10.37) implies that

$$g_n(s, x, z') > g_n(s, x, z) \tag{10.48}$$

This, however, is in contradiction to $z \in X^*(n, s, x)$. Note: If opt $=$ min, reverse the inequalities. $\square$

One may describe the monotonicity conditions formulated above as the clearest, indeed most direct, indication of a decomposition scheme being Markovian or Weak-Markovian.

As for the place and role of the Weak-Markovian condition in dynamic programming, this question is examined in *Chapters 11* and *13.* At this point it suffices to say that the ability to count on the Weak-Markovian condition to secure the validity of the functional equation translates into greater leeway in modeling and formulation. For what this assurance effectively means is that, should a problem under consideration prove readily amenable to a Weak-Markovian decomposition scheme rather than to a Markovian scheme, one would not have to worry whether to give the problem in question the Weak-Markovian formulation. The Weak-Markovian formulation would be totally justified given the assurance for obtaining a valid equation for this formulation (see *Section 10.5*). It is important to be aware, though, of the consequences of using a Weak-Markovian as opposed to a Markovian scheme. The point here is that, contrary to a Markovian decomposition scheme which guarantees that the resulting equation will recover *all* the optimal solutions, all that a Weak-Markovian scheme guarantees is the recovery of *an* optimal solution.

I examine the assurances provided by the Markovian decomposition scheme in the next section.

10.3 Markovian Formulations

By way of introduction, let us recall that as we saw in *Section 7.3,* a valid truncated dynamic programming functional equation is certain to recover *all* the pertinent optimal Markovian policies. Having just seen that a valid equation will equally be obtained from a Weak-Markovian formulation, the implication clearly is that a truncated dynamic programming functional equation leaning on the Weak-Markovian condition will also recover all the pertinent Markovian optimal policies. The question is then whether postulating the Markovian or Weak Markovian condition will have different outcomes with regard to recovering the sets of *optimal solutions* yielded by the respective equations. Consider then the following.

Theorem 10.3.1 *Assume that the decomposition scheme is Markovian, and consider any triplet* (n, s, z) *such that* $1 \leq n < N$, $s \in S_n$ *and* $z = (x_n, \dots, x_N)$ *is an element of* $X^*(n, s)$. *Then,* $z \in X^*(n+1, T(n, s, x_n))$.

Proof Let (n, s, z) be any triplet satisfying the conditions stipulated by the theorem and let $s' = T(n, s, x_n)$. Now, contrary to the theorem, suppose that $z \notin X^*(n+1, s')$. Since the decomposition scheme is assumed to be Markovian, it follows that $z \notin X^*(n, s, x_n)$.

Case 1: opt = max. In this case, $z \notin X^*(n, s, x_n)$ implies the existence of some sequence $z' = (x'_{n+1}, \dots, x'_N) \in X(n+1, s')$ such that

$$g_n(s, x_n, z') > g_n(s, x_n, z) \tag{10.49}$$

or equivalently

$$g_n(s, x_n, x'_{n+1}, x'_{n+2}, \dots, x'_N) > g_n(s, x_n, x_{n+1}, x_{n+2}, \dots, x_N) \tag{10.50}$$

But this is in contradiction to $(x_n, \ldots, x_N) \in X^*(n,s)$, as $a = (x_n, \ldots, x_N) \in X^*(n,s)$ and $b = (x'_{n+1} \ldots, x'_N) \in X(n+1, s')$ entail that $(x_n, b) \in X(n,s)$.

Case 2: opt = min. Reverse the above inequalities. □

Successive appeals to this theorem yield the following result.

Corollary 10.3.1 *Assume that the decomposition scheme is Markovian, and consider any triplet $(n, s, (x_n, \ldots, x_N))$ such that $1 \leq n < N$, $s \in S_n$, and $(x_n, \ldots, x_N)$ is an element of $X^*(n,s)$. Then*

$$(x_m \ldots, x_N) \in X^*(m, s_m) \ , \ \forall n \leq m \leq N \tag{10.51}$$

where $s_m = s$, and $s_{m+1} = T(m, s_m, x_m)$, $n \leq m \leq N$

Putting this in words: an optimal solution to a modified problem induced by a Markovian decomposition scheme is optimal for all the modified problems that it generates.

I shall now show that the set of dynamic programming policies associated with a truncated dynamic programming functional equation deriving from a Markovian formulation, spans the *entire* set of optimal solutions to the family of modified problems.

Theorem 10.3.2 *Assuming that N is finite and the decomposition scheme is Markovian, let (n,s) be any pair such that $n \in \mathbb{N}$ and $s \in S_n$ and let z be any element of $X^*(n,s)$. Then, there exists a dynamic programming policy $\delta \in \Delta^\circ$ such that $x(n,s,\delta) = z$, recalling that $x(n,s,\delta)$ denotes the sequence of decisions generated by applying δ to (n,s) and Δ° denotes the set of dynamic programming policies.*

Proof. Let $(n, s, (x_n, x_{n+1}, \ldots, x_N))$ be any triplet such that $n \in \mathbb{N}$, $s \in S_n$ and $(x_n, \ldots, x_N) \in X^*(n,s)$. We need to show then that if N is finite and the decomposition scheme is Markovian, there exists a dynamic programming policy $\delta \in \Delta^\circ$ such that $x(n,s,\delta) = (x_n, \ldots, x_N)$.

I shall first prove, by induction, that

$$x_m \in D^\circ(m, s_m) \ , \ \forall n \leq m \leq N \tag{10.52}$$

where $s_n = s$ and $s_{m+1} = T(m, s_m, x_m)$, $n \leq m \leq N$, recalling that by definition

$$D^\circ(N,s) := \{x \in D(N,s) : g_N(s,x) = f_N(s)\} \ , \ s \in S_N \tag{10.53}$$

and

$$D^\circ(n,s) := \{x \in D(n,s) : \rho(n,s,x,f_{n+1}(T(n,s,x))) = f_n(s)\} \tag{10.54}$$

for $1 \leq n < N, s \in S_n$.

Since *Corollary 10.3.1* implies that $x_N \in X^*(N, s_N)$, it follows that $g_N(s_N, x_N) = f_N(s)$. Hence, the inductive hypothesis holds for $m = N$. Assume now that the inductive hypothesis holds for $N \geq m \geq k+1$ and that $m = k$. Since *Corollary 10.3.1* insures that $(x_k, \dots, x_N) \in X^*(k, s_k)$, it follows that

$$\begin{aligned} f_k(s_k) &= g_k(s_k, x_k, x_{k+1}, \dots, x_N) \\ &= \rho(k, s_k, x_k, g_{k+1}(s_{k+1}, x_{k+1}, x_{k+2}, \dots, x_N)) \end{aligned} \tag{10.55}$$

Also, with $(x_{k+1}, \dots, x_N) \in X^*(k+1, s_{k+1})$ entailing that

$$g_{k+1}(s_{k+1}, x_{k+1}, x_{k+1}, \dots, x_N) = f_{k+1}(s_{k+1}) \tag{10.56}$$

it follows from (10.55) that

$$f_k(s_k) = \rho(k, s_k, x_k, f_{k+1}(T(k, s_k, x_k))) \tag{10.57}$$

Thus, considering that $x_k \in D(k, s_k)$, it further follows from (10.57) that $x_k \in D^\circ(k, s_k)$. Therefore, the inductive hypothesis holds for $m = k$. In short, (10.52) is true. Finally, observe that, by definition,

$$\Delta^\circ := \{\delta \in \Delta : \delta(n, s) \in D^\circ(n, s), \forall 1 \leq n \leq N, s \in S_n\} \tag{10.58}$$

Since we established that $x_m \in D^\circ(m, s_m)$ for all $n \leq m \leq N$, it follows that there exists at least one policy $\delta \in \Delta^\circ$ such that $\delta(m, s_m) = x_m$ for all $n \leq m \leq N$.

This completes the proof, for this policy clearly has the property that $x(n, s, \delta) = (x_n, \dots, x_N)$. $\square$

To sum up then, a functional equation deriving from a Markovian formulation is assured to recover the entire set of optimal solutions to the problem in question. However, as we shall see in *Section 10.5,* a functional equation stemming from a Weak-Markovian formulation, *is not* assured to recover *all* the optimal solutions.

10.4 Decomposition Schemes

In this section I classify dynamic programming decomposition schemes according to five major types and I describe their respective distinctive properties. My principal objective is to make the point that these are essentially the kinds of schemes that one would be using on a regular basis. I shall also mention briefly a number of exceptions.

To facilitate a formal discussion of this matter, I invoke again the binary function or operator, denoted by the symbol $\oplus$ that was first introduced in *Section 6.7.* I find it useful to borrow from the computer programming language *APL* the following notation to designate the conventional max and min functions. For any pair of real numbers (a, b), define:

$$a \lceil b := \max(a, b) \tag{10.59}$$

$$a \lfloor b := \min(a, b) \tag{10.60}$$

The merit of conceiving of the max and min functions as binary operators is that this enables treating them in the same manner that we treat the conventional binary functions $+$ and $\times$.

I begin by noting that the most prevalent composition function in dynamic programming is of the following form:

$$\rho(n, s, x, a) = w(n, s, x) \oplus a \tag{10.61}$$

for $1 \leq n < N, s \in S_n, x \in D(n, s), a \in \mathbb{R}$, where w is a real-valued function on $\mathbb{N} \times S \times \mathbb{D}$ and the binary operator $\oplus$ serves as a COMPOSITION OPERATOR.

The function w is considered to be a MYOPIC OBJECTIVE FUNCTION. For example, if

$$\rho(n, s, x, a) = \beta^n (x - s)^{1/2} + a \tag{10.62}$$

we would set

$$w(n, s, x) = \beta^n (x - s)^{1/2} \tag{10.63}$$

and $\oplus = +$.

Before proceeding to explain the characteristics of the operator $\oplus$ as it is used in dynamic programming, I note that the following convention will be observed: *the last argument of ρ will be regarded as the* **right** *argument of the composition operator $\oplus$.* That is, the a in $\rho(n, s, x, a)$ will be the right argument of $\oplus$.

Consider now the following: the operator $\oplus$ is said to be MARKOVIAN on $A \subset R$ at $b \in \mathbb{R}$ if

$$\{a^* : a^* = \text{opt}\{a : a \in A\}\} = \{a^\circ : a^\circ \in A, b \oplus a^\circ = \underset{a \in A}{\text{opt}}\{b \oplus a\}\} \tag{10.64}$$

It is said to be WEAK-MARKOVIAN on A at b if

$$\{a^* : a^* = \text{opt}\{a : a \in A\}\} \cap \{a^\circ : a^\circ \in A, b \oplus a^\circ = \underset{a \in A}{\text{opt}}\{b \oplus a\}\} \neq \varnothing \tag{10.65}$$

By the same token, the operator $\oplus$ is said to be MONOTONE on $A \subset \mathbb{R}$ at $b \in \mathbb{R}$ if

$$((a', a) \in A \times A, a' \geq a) \Rightarrow (b \oplus a') \geq (b \oplus a) \tag{10.66}$$

and it is said to be STRICTLY MONOTONE on A at b if

$$((a', a) \in A \times A, a' > a) \Rightarrow (b \oplus a') > (b \oplus a) \tag{10.67}$$

How then are these qualities manifested with respect to our optimization problem?

Assume that ρ admits of the representation defined in (10.61). Thus, with (ρ, G) being a decomposition scheme, we have

$$g_n(s,x,z) = \rho(n,s,x,g_{n+1}(T(n,s,x),z)) \tag{10.68}$$

$$= w(n,s,x) \oplus g_{n+1}(T(n,s,x),z) \tag{10.69}$$

for all $1 \le n < N$, $s \in S_n$, $x \in D(n,s)$ and $z \in X(n+1, T(n,s,x))$ *Problem* $P(n,s,x)$ can therefore be restated as follows:

$$f_n(s,x) = \underset{\substack{s'=T(n,s,x)\\ z\in X(n+1,s')}}{\text{opt}} \{w(n,s,x) \oplus g_{n+1}(s',z)\} \tag{10.70}$$

Similarly, the functional equation

$$f_n(s) = \underset{x\in D(n,s)}{\text{opt}} \rho(n,s,x,f_{n+1}(T(n,s,x))) \ , \ 1 \le n < N, s \in S_n \tag{10.71}$$

can be rewritten thus:

$$f_n(s) = \underset{x\in D(n,s)}{\text{opt}} \{w(n,s,x) \oplus f_{n+1}(T(n,s,x))\} \ , \ 1 \le n < N, s \in S_n \tag{10.72}$$

Four situations arise in this context. To examine them it is convenient to define the following:

$$V := \{(n,s,x) : 1 \le n < N, s \in S_n, x \in D(n,s)\} \tag{10.73}$$

and

$$A(n,s,x) := \{g_n(s',z) : s' = T(n,s,x), z \in X(n+1,s')\} \tag{10.74}$$

Case 1. $\oplus$ is Markovian.
That is for each $(n,s,x) \in V$ the composition operator $\oplus$ is Markovian on $A(n,s,x)$ at $w(n,s,x)$.

Case 2. $\oplus$ is Weak-Markovian.
As above except that $\oplus$ is Weak-Markovian rather than Markovian.

Case 3. $\oplus$ is strictly monotone.
For every $(n,s,x) \in V$, the composition operator $\oplus$ is monotone increasing on $A(n,s,x)$ at $w(n,s,x)$.

Case 4. $\oplus$ is monotone.
As above except that $\oplus$ is monotone increasing rather than strictly monotone increasing.

The relationships between these cases can be summarized as follows:

$$\begin{tikzcd}
\oplus \text{ is strictly monotone} \arrow[r, Rightarrow] \arrow[d, Rightarrow] & \oplus \text{ is Markovian} \arrow[d, Rightarrow] \\
\oplus \text{ is monotone} \arrow[r, Rightarrow] & \oplus \text{ is Weak-Markovian}
\end{tikzcd}$$

The composition operator $\oplus$ will thus be part of the following four types of decomposition schemes:

10.4.1 Additive

This is by far the most commonly encountered scheme in dynamic programming. It consists of an objective function of the following familiar form:

$$g(s_1,x_1,x_2,\dots,x_N)=\sum_{n=1}^{N} w(n,s_n,x_n) \tag{10.75}$$

Obviously, the objective functions pertaining to the modified problems are also additive, namely

$$g_n(s_n,x_n,x_{n+1},\dots,x_N)=\sum_{m=n}^{N} w(m,s_m,x_m) \tag{10.76}$$

and therefore

$$\rho(n,s,x,a)=w(n,s,x)+a\ ,\ a\in\mathbb{R} \tag{10.77}$$

in which case $\oplus = +$.

Needless to say, $\oplus$ is strictly monotone — and therefore Markovian — in this case.

10.4.2 Multiplicative

Here the objective function is

$$g(s_1,x_1,x_2,\dots,x_N)=\prod_{n=1}^{N} w(n,s_n,x_n) \tag{10.78}$$

so that

$$g_n(s_n,x_n,x_{n+1},\dots,x_N)=\prod_{m=n}^{N} w(m,s_m,x_m)\ ,\ 1\le n\le N \tag{10.79}$$

and

$$\rho(n,s,x,a)=w(n,s,x)\times a\ ,\ a\in\mathbb{R} \tag{10.80}$$

By construction then,

$$g(s_1,x_1,x_2,\dots,x_N)=g_1(s_1,x_1,x_2,\dots,x_N) \tag{10.81}$$

and

$$\begin{aligned} g_n(s_n,x_n,x_{n+1},\dots,x_N) &= w(n,s_n,x_n)\times g_{n+1}(s_{n+1},x_{n+1},\dots,x_N) && (10.82)\\ &= \rho(n,s_n,x_n,g_{n+1}(s_{n+1},x_{n+1},\dots,x_N)) && (10.83)\end{aligned}$$

for all $1\le n<N$. Hence, $(\rho, G=\{g_n\})$ is a proper decomposition scheme, and we can set $\oplus = \times$.

This type of decomposition scheme will have three sub-types:

Case 1. $w(n, s, x) > 0$, $\forall 1 \leq n < N$, $s \in S_n$, $x \in D(n, s)$
Here $\oplus$ is strictly monotone — therefore Markovian — as

$$a' > a \Rightarrow b \times a' > b \times a \tag{10.84}$$

Case 2. $w(n, s, x) \geq 0$, $\forall 1 \leq n < N$, $s \in S_n$, $x \in D(n, s)$
Since

$$a' \geq a \Rightarrow (b \times a' \geq b \times a \ , \ \forall b \geq 0) \tag{10.85}$$

plainly $\oplus$ is monotone — therefore Weak-Markovian.

Case 3. Neither Case 1 nor Case 2.
In this case $\oplus$ is not necessarily monotone, and by the same token not necessarily strictly monotone. Furthermore, $\oplus$ is not necessarily Weak-Markovian. All the same, it is not difficult to envisage an instance where $\oplus$ would be Markovian.

10.4.3 $\lfloor$-decomposition Scheme

Under this heading are classified schemes whose objective functions are of the form:

$$g(s_1, x_1, x_2, \ldots, x_N) = \min_{1 \leq n \leq N} \{w(n, s_n, x_n)\} \tag{10.86}$$

in which case

$$g_n(s_n, x_n, x_{n+1}, \ldots, x_N) = \min_{n \leq m \leq N} \{w(m, s_m, x_m)\} \ , \ 1 \leq n \leq N \tag{10.87}$$

and the composition function is of the following form:

$$\rho(n, s, x, a) = \min\{w(n, s, x), a\} \ , \ a \in \mathbb{R} \tag{10.88}$$

Clearly, $g = g_1$ and

$$\begin{aligned} g_n(s_n, x_n, \ldots, x_N) &= \min\{w(n, s_n, x_n), g_{n+1}(s_{n+1}, x_{n+1}, \ldots, x_N)\} &(10.89)\\ &= \rho(n, s_n, x_n, g_{n+1}(s_{n+1}, x_{n+1}, x_{n+2}, \ldots, x_N)) &(10.90)\end{aligned}$$

for all $1 \leq n < N$.

Hence, $(\rho, G = \{g_n\})$ is a proper decomposition scheme, and we can set $\oplus = \lfloor$. Since,

$$b' \geq b \Rightarrow \min(a, b') \geq \min(a, b) \ , \ \forall a \in \mathbb{R} \tag{10.91}$$

it is clear that $\oplus$ is monotone — and therefore Weak-Markovian — in this case.

10.4.4 $\lceil$-decomposition Scheme

Schemes falling in this category are analogous to those falling in the previous one except that here min is replaced by max, hence $\oplus = \lceil$. Note that $\oplus$ is monotone — therefore Weak-Markovian — in this case.

10.4.5 The Final State Scheme

You will recall that in *Section 4.6* I described the final state model as a multistage decision model in whose case the composition function ρ has this form:

$$\rho(n,s,x,a) = a \ , \ \forall 1 \leq n < N, s \in S_n, x \in D(n,s), a \in \mathbb{R} \tag{10.92}$$

Now, on the face of it, this phrasing seems to be incongruent with that of (10.61) as the myopic objective function w is absent from it. But this apparent discrepancy can be easily rectified. To do this, let $\oplus$ be any binary operator possessing a left identity element. That is, suppose that

$$L \oplus a = a \ , \ \forall a \in \mathbb{R} \tag{10.93}$$

for some $L \in \mathbb{R}$. We refer to to L as the *left identity element of* $\oplus$.

Then clearly, the function ρ specified by (10.92) can now be stated thus:

$$\rho(n,s,x,a) = L \oplus a \ , \ \forall 1 \leq n < N, s \in S_n, x \in D(n,s), a \in \mathbb{R} \tag{10.94}$$

Observe that in contrast to $+$, $\times$, $\lfloor$ and $\lceil$, this binary operator is *non-associative.*

Recall from *Section 6.8* that the right identity elements of $+$ and $\times$ are 0 and 1 respectively. Now as these functions are commutative, it follows that their left and right identity elements are identical, hence the left identity element of $+$ is equal to 0 and the left identity element of $\times$ is equal to 1.

As for the identity elements of $\lfloor$ and $\lceil$, because these operators are commutative and

$$-\infty \lceil a = a \ , \ \forall a \in \mathbb{R} \tag{10.95}$$

$$\infty \lfloor a = a \ , \ \forall a \in \mathbb{R} \tag{10.96}$$

the implication is that both the left and right identity elements of $\lceil$ and $\lfloor$ are equal to $-\infty$ and ∞ respectively.

In short, defining $L_\oplus$ as the left identity element of $\oplus$, we have:

$$L_\oplus = \begin{cases} 0 & , \ \oplus = + \\ 1 & , \ \oplus = \times \\ \infty & , \ \oplus = \lfloor \\ -\infty & , \ \oplus = \lceil \end{cases} \tag{10.97}$$

The upshot of all this is that we can square (10.61) with (10.92) by setting:

$$w_\oplus(n,s,x) = L_\oplus \ , \ \forall 1 \leq n < N, s \in S_n, x \in D(n,s), a \in \mathbb{R} \tag{10.98}$$

whereupon the composition function of the final state scheme would be stated thus:

$$\rho(n,s,x,a) = w_{\oplus}(n,s,x) \oplus a \tag{10.99}$$

for all $1 \leq n < N, s \in S_n, x \in D(n,s), a \in \mathbb{R}$

It should be noted that in this case $\oplus$, and therefore ρ, is strictly monotone with a. □

10.4.6 Salvage Value

There are many situation where it is convenient to associate a *salvage value* with the final state, s_{N+1}, of a truncated process. This means that the decomposition scheme practically has the property that

$$g_N(s,x) = \rho(s,x,W(T(N,s,x))) \tag{10.100}$$

where W is a real-valued function on S.

If a binary composition operator $\oplus$ is used to construct the objective function, then we would have

$$g(s_1,x_1,x_2,\dots,x_N) = w(s_1,x_1)\cdots \oplus w(s_N,x_N) \oplus W(s_{N+1}) \tag{10.101}$$

I shall refer to W as the *salvage-value function*. Observe that if such a function is not required, we can set $W(s) = R_{\oplus}$, namely $W(s)$ is equal to the right identity element of $\oplus$. This in effect neutralizes the impact of W on the objective function.

Definition 10.4.1 *A* BINARY DECOMPOSITION SCHEME *for the objective function of a truncated multistage decision model* $(\mathbb{N},S,D,T,S_1,g)$ *is a triplet* $(w,W,\oplus)$ *such that (10.101) holds.*

10.4.7 Remarks

The above catalogue practically covers the main types of decomposition schemes that one would be likely to employ in dynamic programming. Generally, there would be no need to resort to the use of other types of scheme. Indeed, such cases would be totally unrepresentative and rare. However, such rare cases exist.

For instance, consider the case where $N = 3$ and

$$g(s_1,x_1,x_2,\dots,x_N) = [w(1,s_1,x_1)]^{[w(1,s_2,x_2)]^{[w(1,s_3,x_3)]}} \tag{10.102}$$

so that

$$g_3(3,s_3,x_3) = w(3,s_3,x_3) \tag{10.103}$$

$$g_2(s_2,x_2,x_3) = [w(2,s_2,x_2)]^{g_3(3,s_3,x_3)} \tag{10.104}$$

$$g_1(s_1,x_1,x_2,x_3) = [w(1,s_1,x_1)]^{g_2(s_2,x_2,x_3)} \tag{10.105}$$

$$\rho(n,s,x,a) = [w(n,s,x)]^{a} \tag{10.106}$$

for $1 \leq n < 3, s \in S_n, x \in D(n,s), a \in \mathbb{R}$.

Here $\oplus$ is the *power operator,* that is, $\oplus = *$ where $b * a := b^a$. Clearly, if $w(n,s,x) > 1$, for all (n,s,x), then $\oplus = *$ is strictly monotone. Similarly, if $w(n,s,x) \geq 1$, then $\oplus = *$ is monotone. Note, however, that for optimization purposes, this scheme can be transformed into an equivalent *multiplicative scheme* by the usual log *transformation.*

Also, there may be cases calling for the use of more than one composition operator in the same formulation. For example, consider the case where $N = 100$ and the objective function is of the form:

$$g(s_1, x_1, x_2, \dots, x_{100}) = \sum_{n=1}^{50} w(n, s_n, x_n) + \min_{51 \leq n \leq 100} \{w(n, s_n, x_n)\} \tag{10.107}$$

Then

$$g_n(s_n, x_n, x_{n+1}, \dots, x_{100}) = \min_{n \leq m \leq 100} \{w(m, s_m, x_m)\} \tag{10.108}$$

for $51 \leq n \leq 100$,

$$g_n(s_n, x_n, x_{n+1}, \dots, x_{100}) = \sum_{k=n}^{50} w(m, s_k, x_k) + \min_{51 \leq m \leq 100} \{w(m, s_m, x_m)\} \tag{10.109}$$

for < 51, and

$$\rho(n,s,x,a) = \begin{cases} \min(w(n,s,x), a) & , 51 \leq n \leq 99 \\ w(n,s,x) + a & , 1 \leq n \leq 50 \end{cases} \tag{10.110}$$

The formulation would then involve two composition operators, namely

$$\oplus_n = \begin{cases} + & , 51 \leq n \leq 99 \\ \lfloor & , 1 \leq n \leq 50 \end{cases} \tag{10.111}$$

And, in principle, there could be cases where $\oplus$ depends on the state variable, and perhaps even on the decision variable.

It should be pointed out, though, that the quadruplet $(+, \times, \lfloor, \lceil)$ cannot handle all the cases admitting of a dynamic programming formulation. For example, consider the case where $N = 6$

$$g(s_1, x_1, x_2, \dots, x_7) = x_1 + \sqrt{x_2 + \sqrt{x_3 + \sqrt{x_4 + \sqrt{x_5 + \sqrt{x_6}}}}} \tag{10.112}$$

so that we can set

$$\rho(n,s,x,a) = x + \sqrt{a} \tag{10.113}$$

observing that ρ is strictly increasing with its last argument hence this scheme is Markovian.

To express this scheme with the aid of a binary operation we would have to define the handy $\oplus$ as follows:

$$b \oplus a = b + \sqrt{a} \tag{10.114}$$

observing that in this case $w(n,s,x) = x$ and therefore two elementary operations are at work here, namely *addition* and *square rooting.*

And how about the following continued-fraction type expression:

$$\rho(n,s,x,a) = x + \cfrac{1}{1 + \cfrac{1}{a}} \quad , \ a > 0 \tag{10.115}$$

observing that ρ is strictly monotone with its last argument, hence the associated decomposition scheme is Markovian.

We need do no more than simply let

$$b \oplus a = b + \cfrac{1}{1 + \cfrac{1}{a}} \quad , \ a > 0 \tag{10.116}$$

in which case $w(n,s,x) = x$ and a combination of *addition* and *division* operations will do the job.

The fact that the scheme $(w,\oplus)$ is binary means that it is not as general as the $(\rho,\{g_n\})$ scheme. For example, consider the more exotic continued fraction type expression:

$$\rho(n,s,x,a) = s + \cfrac{1}{x + \cfrac{1}{a}} \quad , \ x,a > 0 \tag{10.117}$$

How would we express this scheme using the binary $(w,\oplus)$ format?

On the face of it, this seems impossible because this operation requires three arguments, namely (s,x,a). So, to be able to perform it, we need to be a bit more cunning and do as follows:

$$(s,x) \oplus a := s + \cfrac{1}{x + \cfrac{1}{a}} \quad , \ x,a > 0 \tag{10.118}$$

So, let the left argument of $\oplus$ be a pair of numbers, representing the state and decision variables. Formally, in this case $w(n,s,x) = (s,x)$, hence w is not a real-valued function.

To summarize, then: The most commonly used decomposition schemes in dynamic programming can be classified according to five major types. Indeed, this classification can be further consolidated to obtain *three* types: *additive*, $\lfloor$ and *final state.* This is due to the $\lfloor$ and $\lceil$ schemes being equivalent (subject to multiplication by -1) and the fact that a strictly monotone multiplicative scheme can be converted into an equivalent additive scheme by a log transformation.

The main consequence of this classification is in the implications it has for the degree of flexibility that one would be able to exercise when modeling a problem in dynamic programming style. The point to note here is that, by and large one would give the problem an *additive* (*multiplicative*) formulation or a $\lfloor$ ($\lceil$) formulation, and depending on the conditions that would hold, the formulations would be either Markovian (strictly monotone) or Weak-Markovian (monotone). Should the problem concerned prove defiant of these schemes one would always be able to resort to a final state option (see *Chapter 12* for examples). There are, of course, problems with genuine final state objective functions and there are situations that require the application of other decomposition schemes.

Of particular interest are decomposition schemes that are not associative, for instance $a \oplus b = a + \sqrt{b}$ and $a \oplus b = a + b^2$. These are discussed in *Chapter 14* and *Chapter 15* in connection with the use of *forward* decomposition schemes. Although these somewhat exotic schemes are primarily of theoretical rather than practical interest, they are important because they illustrate the flexibility that dynamic programming enables in the modeling of a problem.

In a nutshell, when modeling a problem in dynamic programming style, one would not have to consider an array of composition functions that would have to be subjected to a close analysis in order to determine which will yield a valid functional equation for the problem concerned. Basically, there are three major types of decomposition schemes, whose properties are well known and fully understood in terms of the types of solutions that their respective functional equations yield. The inference is therefore that any investigation of the Markovian and monotonicity properties would primarily be of theoretical interest.

Before winding up this part of the discussion, I take up again the question of the relation between the truncation method and the successive approximation method, as a better familiarity with the identity element concept should now enable a better appreciation of this point.

10.4.8 Truncation Method Revisited

Recall that I argued in *Section 6.8* that the truncation and successive approximations methods basically amount to the same thing, and that the element through which the two methods converge is a specific initial approximation $u^{(0)}$.

I shall now show that this approximation is in actual fact equal to the value of the right identity element of the composition operator $\oplus$.

For simplicity, assume that the formulation is stationary and that the functional equation is of the form

$$f(s) = \max_{x \in D(s)} \rho(s, x, f(T(s,x))) \ , \ s \in S \tag{10.119}$$

Recall that the successive approximation procedure is concerned with a map A defined as follows:

$$(Au)(s) := \sup_{x \in D(s)} \rho(s, x, u(T(s,x))) \ , \ s \in S \tag{10.120}$$

On the other hand, the truncation method postulates a sequence of truncated objective functions $\{g^{(m)}\}$ such that

$$g(s_1, x_1, x_2, \dots) = \lim_{m \to \infty} g^{(m)}(s_1, x_1, x_2, \dots, x_m) \tag{10.121}$$

$$g^{(m+1)}(s_1, x_1, x_2, \dots, x_{m+1}) = \rho(s_1, x_1, g^{(m)}(s_2, z)) \tag{10.122}$$

$$z = (x_2, x_3, \dots, x_{m+1}) \tag{10.123}$$

$$f^{(m+1)}(s) = \operatorname*{opt}_{x \in D(s)} \rho(s, x, f^{(m)}(T(s,x))) \tag{10.124}$$

for all $s \in S$.

The functions $\{f^{(m)}\}$ themselves are defined thus:

$$f^{(m)}(s) := \operatorname*{opt}_{(x_1, x_2, \dots, x_m)} g^{(m)}(s, x_1, x_2, \dots, x_m) \tag{10.125}$$

$$x_k \in D(s_k) \ , \ 1 \leq k \leq m \tag{10.126}$$

$$s_1 = s \tag{10.127}$$

$$s_{k+1} = T(s_k, x_k) \ , \ 1 \leq k \leq m \tag{10.128}$$

I demonstrated in *Section 6.8* that the connecting link between the two methods is a specific initial approximation $u^{(0)}$ such that

$$g^{(1)}(s, x) = \rho(s, x, u^{(0)}(T(s,x))) \ , \ \forall s \in S, x \in D(s) \tag{10.129}$$

Now, assume that ρ admits the following representation:

$$\rho(s, x, a) = w(s, x) \oplus a \ , \ \forall s \in S, x \in D(s), a \in \mathbb{R} \tag{10.130}$$

In this case (11.87) entails that

$$g^{(1)}(s, x) = w(s, x) \oplus u^{(0)}(T(s,x)) \ , \ \forall s \in S, x \in D(s) \tag{10.131}$$

Next, assume that $\oplus$ has a right identity element, namely, assume that there exists some $R_{\oplus} \in \mathbb{R}$ such that

$$w(s, x) \oplus R_{\oplus} = w(s, x) \ , \ \forall s \in S, x \in D(s) \tag{10.132}$$

If we now set

$$u^{(0)}(s) = R_{\oplus} \ , \ \forall s \in S \tag{10.133}$$

then clearly it follows from (10.131) that

$$g^{(1)}(s,x) = w(s,x) \ , \ \forall s \in S, x \in D(s) \tag{10.134}$$

and consequently from (10.123) that

$$\begin{align} g^{(2)}(s_1,x_1,x_2) &= w(s_1,x_1) \oplus g^{(1)}(s_2,x_2) \tag{10.135}\\ &= w(s_1,x_1) \oplus w(s_2,x_2) \tag{10.136}\\ g^{(3)}(s_1,x_1,x_2,x_3) &= w(s_1,x_1) \oplus g^{(2)}(s_2,x_2,x_3) \tag{10.137}\\ &= w(s_1,x_1) \oplus [w(s_2,x_2) \oplus w(s_3,x_3)] \tag{10.138}\\ &\mathit{etc} \tag{10.139}\end{align}$$

Note that because $w(s,x) \oplus R_\oplus = w(s,x)$, the right identity element of $\oplus$ can be appended infinitely many times on the right-hand side of $g^{(m)}$ without its value being affected. Thus,

$$g^{(1)}(s_1,x_1) = w(s_1,x_1) \oplus [R_\oplus \oplus [R_\oplus \oplus [R_\oplus \oplus [R_\oplus \oplus \cdots\cdots\cdot]]]] \tag{10.140}$$

$$g^{(2)}(s_1,x_1,x_2) = w(s_1,x_1) \oplus [w(s_2,x_2) \oplus [R_\oplus \oplus [R_\oplus \oplus [R_\oplus \oplus \cdots\cdot]]]] \tag{10.141}$$

etc, observing that the square brackets are superfluous if $\oplus$ is associative.

In short, $g^{(m)}$ can be viewed as a nontruncated objective function originating in g which is obtained by replacing the *myopic* objective function w with $R_\oplus$ at any stage $n > m$. For example, consider the case where

$$g(s_1,x_1,x_2,...) = \sum_{n=1}^{\infty} \beta^{n-1} x^{1/2} \ , \ 0 \leq \beta < 1 \tag{10.142}$$

Set

$$g^{(m)}(s_1,x_1,x_2,\dots,x_m) = \sum_{n=1}^{m} \beta^{n-1} x^{1/2} \ , \ m = 1,2,3,\dots \tag{10.143}$$

and

$$\rho(s,x,a) = x^{1/2} + \beta a \ , \ a \in \mathbb{R} \tag{10.144}$$

Then

$$\begin{align} w(s,x) &= x^{1/2} \tag{10.145}\\ b \oplus a &= b + \beta a \tag{10.146}\\ R_\oplus &= 0 \tag{10.147}\end{align}$$

where $R_\oplus$ denotes the right identity element of $\oplus$. Clearly, $g^{(m)}$ can be regarded as a real-valued function on $S \times \mathbb{R}^\infty$ such that

$$g^{(m)}(s_1,x_1,x_2,...) = \sum_{n=1}^{\infty} w'(n,s_n,x_n) \tag{10.148}$$

where

$$w'(n,s,x) = \begin{cases} \sum\limits_{n=1}^{m} \beta^{n-1} x^{1/2} & , \quad n \leq m \\ R_{\oplus} = 0 & , \quad n > m \end{cases} \tag{10.149}$$

As we shall shortly see, the right identity element of $\oplus$ can be used with good effect in the formulation and analysis of truncated problems.

10.5 Sequential Decision Models

In this section I introduce a generic dynamic programming model for the description of optimization problems that is devoid of an explicit stage variable. This model also has the merit of depicting problems with feasible solutions of various lengths in a manner that immediately captures this property.

The specific structure of the model reflects a compromise between countervailing considerations so that while it is not trivially simple it is decidedly not too complicated. It thus proves suitable for the formulation of a number of important classes of problems notably *cyclic* problems.

I shall unfold the generic model in stages and when it is formulated in full I shall proceed to consider a number of special cases. The dynamic programming functional equations that I shall derive for this model will prove similar to those obtained in the preceding chapters for the multistage model.

I begin with the following extremely simple construct.

Definition 10.5.1 *A* STATE TRANSITION MODEL *is a triplet* (S, D, T) *where:*

1. *S is a nonempty set, called the* STATE SPACE.
2. *D is a function on S such that it assigns to each $s \in S$ a subset of some nonempty set $\mathbb{D}$. The set $D(s)$ is understood to be the collection of* FEASIBLE DECISIONS *available at state s. We refer to $\mathbb{D}$ as the* DECISION SPACE.
3. *T is a map from $S \times \mathbb{D}$ to S called the* TRANSITION FUNCTION. *Thus, if $s \in S$ and $x \in D(s)$, then $s' = T(s,x)$ is construed as the state resulting from the choice of decision x at state s.*

A distinction is made between two types of states, namely

$$S' := \{s \in S : D(s) \neq \varnothing\} \tag{10.150}$$

$$S'' := \{s \in S : D(s) = \varnothing\} \tag{10.151}$$

where $\varnothing$ denotes the empty set.

The elements of S'' are labeled TERMINAL STATES. *These are states where the decision making process terminates. In contrast, the elements of S' are states where the decision making process must continue. The elements of this set are termed* NONTERMINAL STATES.

We say that the model (S, D, T) is FINITE *if the state space S and the decision space $\mathbb{D}$ consist of finitely many elements, namely if $|S| < \infty$ and $|\mathbb{D}| < \infty$.*

And we say that the model (S, D, T) is CYCLIC *if there is a state $s \in S'$ and a sequence of decisions, $(x_1, \dots, x_k)$ such that*

$$s_1 = s_{k+1} = s \tag{10.152}$$

$$x_n \in D(s_n)\ ,\ n = 1, 2, \dots, k \tag{10.153}$$

$$s_{n+1} = T(s_n, x_n)\ ,\ n = 1, 2, \dots, k \tag{10.154}$$

If there are no such cycles then the model is said to be ACYCLIC.

Finite state transition models have an immediate appeal. This is due to the fact that they naturally lend themselves to simple graphic representations. That is, such models can be depicted as *directed graphs* where the vertices and arcs represent states and decisions, respectively. Conversely, a directed graph can be interpreted as a finite state transition model.

10.5.1 Example

Consider the directed graph in Figure 10.1. Formally the finite, acyclic state transition model embodied in this graph would be defined as follows:

$$\begin{aligned}
&S = \{1, 2, 3, 4, 5, 6\}\ ,\ \mathbb{D} = \{a, b, c, d, e, f, g, h, i, j\}\\
&D(1) = \{a, b, c\}; D(2) = \{d, e\}; D(3) = \{f, g\}; D(4) = \{h, i\}\\
&D(5) = D(6) = \varnothing \Rightarrow S'' = \{5, 6\}, S' = \{1, 2, 3, 4\}\\
&T(1, a) = 2; T(1, b) = 4; T(1, c) = 3; T(2, d) = 5; T(2, e) = 4; T(3, f) = 4\\
&T(3, g) = 6; T(4, h) = 5; T(4, i) = 6; T(4, j) = 6
\end{aligned}$$

Observe that if the process is at state (node) 4 then only one transition is

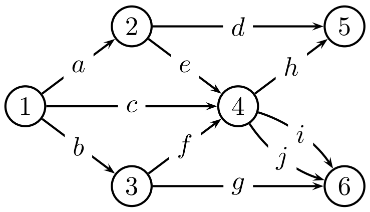


Figure 10.1: A simple finite, acyclic state transition model

possible before a terminal state is reached. This means that at state 4 only one additional decision can be made. In contrast, if the process is at state 2 the number of possible additional transitions is either one or two.

And if the process is at node 1 then the number of possible additional transitions varies from 2 to 3. □

The next example features a finite cyclic model.

10.5.2 Example

Consider the directed graph in Figure 10.2 where all the decisions (arcs) are labeled. Clearly, this is a cyclic model as the states (nodes) 2,4,5, are linked via a cycle.

One of the consequences of this is that although the model is finite, there are infinitely many ways of reaching say, node 6 from node 1. Here is a short lists of sequences of decisions that lead from node 1 to node 6:

$$
\begin{aligned}
A &= (a,b,e)\\
B &= (a,b,c,d,b,e)\\
C &= (a,b,c,d,b,c,d,b,e)\\
D &= (a,b,c,d,b,c,d,b,c,d,b,e)\\
E &= (a,b,c,d,b,c,d,b,c,d,b,c,d,b,e)\\
F &= (a,b,c,d,b,c,d,b,c,d,b,c,d,b,c,d,b,e)\\
G &= (a,b,c,d,b,c,d,b,c,d,b,c,d,b,c,d,b,c,d,b,e)
\end{aligned}
$$

And there are infinitely long sequences of feasible decisions that will cycle

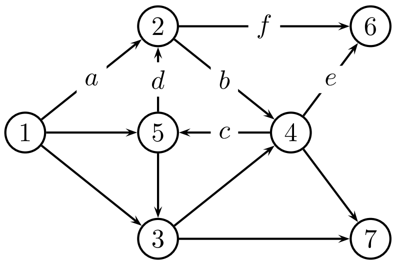


Figure 10.2: A simple finite, cyclic state transition model

indefinitely between the nodes 2,4,5. For instance, the sequence of decisions

$$\mathbf{z} = (a,b,c,d,b,c,d,b,c,d,b,c,d,\dots)$$

is a case in point. □

The state transition model has another merit. It can be used as a device to represent the *decision space* of optimization problems.

I shall illustrate this in the context of a number of optimization problems. As a prelude, it will be instructive first to classify the decisions sequences that the model can generate by means of the following organizing idea:

· When a terminal state is reached, the process terminates. No more decisions are made.
· When the decision-making process is at a non-terminal state, a sequence of decisions is made until a terminal state is reached.
· If no terminal state is reached, the decision making process continues indefinitely.

To distinguish between decision sequences of different lengths, let $X^{(k)}(s)$ denote the set of all feasible decision sequences that, through exactly k transitions, progress from state s to a terminal state. More formally,

Definition 10.5.2 *Let $X^{(k)}(s)$ denote the set of admissible sequences of decisions of length k emanating from state s in accordance with the structure of the state transition model (S, D, T). That is, let*

$$X^{(k)}(s) := \{(x_1, \dots, x_k) : x_n \in D(s_n), s_1 = s, s_{n+1} = T(s_n, x_n) \\ , n = 1, 2, \dots, k \ , \ s_{k+1} \in S''\} \tag{10.155}$$

for $s \in S, k = 1, 2, \dots,$ observing that by construction

$$X^{(k+1)}(s) = \{(x, \mathbf{z}) : x \in D(s), \mathbf{z} \in X^{(k)}(T(s, x))\} \tag{10.156}$$

Clearly, if there are no terminal states, then $X^{(k)}(s) = \varnothing$ for all $s \in S, k = 1, 2, \dots.$

We say that the state transition model is TRUNCATED *if there exists a $k < \infty$ such that $X^{(k)}(s) = \varnothing, \forall s \in S$.*

Extend the definition of $X^{(k)}(s)$ to $k = \infty$, namely let

$$X^{(\infty)}(s) := \{(x_1, x_2, \dots) : x_n \in D(s_n), s_1 = s, s_{n+1} = T(s_n, x_n), n = 1, 2, \dots\} \tag{10.157}$$

and define

$$X(s) := \left\{\bigcup_{k=1}^{\infty} X^{(k)}(s)\right\} \bigcup X^{(\infty)}(s) \tag{10.158}$$

Thus, $X(s)$ denotes the set of all admissible sequences of decisions emanating from state s. For terminal states it is convenient to set

$$X(s) := \{(\,)\} \ , \ s \in S'' \tag{10.159}$$

where $(\,)$ denotes the empty sequence, as this entails that

$$X(s) = \{(x, \mathbf{z}) : x \in D(s), \mathbf{z} \in X(T(s, x))\} \ , \ \forall s \in S' \tag{10.160}$$

Observe that in this expression if $T(s, x)$ is a terminal state then $X(T(s, x))$ is a singleton consisting of the empty sequence $(\ )$ and therefore $(x, \mathbf{z}) = x, \forall \mathbf{z} \in X(T(s, x))$.

It is important to note that $X(s)$ may contain sequences of various lengths. For instance, in the case of *Example 10.4,* if we set the initial state to be $\sigma = 1$ and let $S'' = \{6,7\}$, then under the first scenario $X(1)$ consists of infinitely many sequences ranging from short ones such as (a, f) and a, b, e, to extremely long ones, with each consisting of billions of decisions, for instance

$$(a,b,c,b,c,d,b,c,d,b, \overset{\text{965432187654321357 replicas of } (c,d,b)}{\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots}, e) \tag{10.161}$$

The following example features a very simple instance where for any state $s \in S$ the set $X(s)$ consists only of sequences of infinite lengths.

10.5.3 Example

In this example I construct a state transition model for the following familiar constraints:

$$\sum_{j=1}^{\infty} x_j \le V \ , \ V > 0 \tag{10.162}$$

$$x_j \ge 0 \ , \ j = 1,2,3,\ldots \tag{10.163}$$

where $\{x_j\}$ are the decision variables and V is a given positive scalar.

Let the state variable s denote the amount of resource left after the instantiation of some of the decision variables. Since all the decision variables are non-negative, then clearly by definition $0 \le s \le V$, hence we can set $S = [0, V]$.

And given that the amount of resource left is s, the next decision cannot exceed s, hence we can set $D(s) = [0, s]$.

From the above it follows that the state variable changes according to the rule $s' = s - x$, hence we can set $T(s, x) = s - x$.

The complete state transition model for the above constraints is therefore as follows:

$$S = [0, V] \tag{10.164}$$

$$D(s) = [0, s] \tag{10.165}$$

$$T(s, x) = s - x \tag{10.166}$$

Note that since $D(s) \neq \varnothing, \forall s \in S$, the implication is that there are no terminal states, hence $S'' = \varnothing$ and $S' = S$. That is, once the decision making process commences, it cannot be terminated: infinitely many decisions continue to be made. This accords with (10.162).

A sequence of decisions $(x_1, x_2, \ldots)$ is feasible with respect to the constraints under consideration if the sequence of states it generates when applied to state $s = V$ produces states in S, observing that this sequence of states is as follows:

$$s_1 = V \tag{10.167}$$
$$s_2 = T(s_1, x_1) = s_1 - x_1 = V - x_1 \tag{10.168}$$
$$s_3 = T(s_2, x_2) = s_2 - x_2 = V - x_1 - x_2 \tag{10.169}$$
$$s_4 = T(s_3, x_3) = s_3 - x_3 = V - x_1 - x_2 - x_3 \tag{10.170}$$
$$\cdots = \cdots$$
$$s_{n+1} = T(s_n, x_n) = s_n - x_n = V - \sum_{j=1}^{n} x_j \tag{10.171}$$

hence, the sequence of states will converge to

$$\tilde{s} = \lim_{n\to\infty} s_n = V - \sum_{j=1}^{\infty} x_j \tag{10.172}$$

For example, the sequence of decision given by $\mathbf{x} = (0, 0, V/2, 0, V/2, 0, 0, 0, 0, 0, \dots)$ is feasible and so is the sequence defined by

$$x_j = \frac{V}{2^j} \; , \; j = 1, 2, 3, \dots \tag{10.173}$$

There are of course other, indeed infinitely many, sequences that satisfy the constraints under consideration. Formally then, in this case

$$X^k(s) = \varnothing \; , \; k = 1, 2, 3, \dots \tag{10.174}$$
$$X(s) = X^{(\infty)}(s) = \left\{ (x_1, x_2, \dots) : \sum_{n=1}^{\infty} x_n \leq s, x_n \geq 0, n = 1, 2, \dots \right\} \tag{10.175}$$

It follows then that the set corresponding to the constraints under consideration is $X(V)$. □

To go back then to where we left off, recall that my objective is to illustrate how the state transition model and its related sets of admissible sequences of decisions, $X(s), s \in S$, can represent the feasible regions of the optimization problem being solved. Obviously, to this end provisions must be made in the model to ensure that these sets are consistent with the decision space of the optimization problem in question.

So, the point to keep in mind in this regard is that *optimization problems* often allow a measure of flexibility in the construction of the *decision space of the problem.* One would therefore have a free hand to "juggle" the decision space of the problem considered according to one needs with the proviso that this must be done in a manner that would not hamper the recovery of an optimal solution to the problem in question. For, after all, this is the point of the whole exercise: the recovery of an *optimal solution* to the problem under consideration.

My next task is then to construct a *sequential decision model* out of our state transition model. This requires the introduction of an *objective function* and the specification of an *initial state.* The end product is this:

Definition 10.5.3 *A* SEQUENTIAL DECISION MODEL *is a collection* (S, D, T, σ, g) *where:*

- (S, D, T) *is a state transition model.*
- σ *is an element of* S.
- g *is a real-valued function on* $\{\sigma\} \times X(\sigma)$.

The state σ *is understood to be the* INITIAL STATE *of the process,* g *is viewed as an* OBJECTIVE FUNCTION. *If the state transition model* (S, D, T) *is finite then the sequential decision model is said to be* FINITE. *If the state transition model is cyclic then the sequential decision model is said to be* CYCLIC *and if the state transition model is truncated then the sequential decision model is said to be* TRUNCATED.

Our generic sequential decision model represents the following sequential decision problem: the initial state of the process being, $s = \sigma$, select out of all the sequences of feasible decisions, a sequence that optimizes the objective function g. To state it formally,

Definition 10.5.4

$$\text{Problem } P(\sigma): \quad f(\sigma) := \underset{z\in X(\sigma)}{\text{opt}}\ g(\sigma, z) \tag{10.176}$$

We shall refer to this problem as the INITIAL PROBLEM AT σ. *Let* $X^*(\sigma)$ *denote the set of optimal solutions to Problem* $P(\sigma)$.

More detailed formulations of *Problem* $P(\sigma)$, where the set $X(\sigma)$ is replaced by constraints on the individual decisions $\{z_j\}$, would depend on the structure of $X(\sigma)$, specifically on the *lengths* of the sequences of decisions contained in this set.

For instance, if all the elements of $X(\sigma)$ are of the SAME LENGTH, say k, then *Problem* $P(\sigma)$ can be expressed as follows:

$$f(\sigma) := \underset{x_1,\dots,x_k}{\text{opt}}\ g(s_1, x_1, x_2, \dots, x_k) \tag{10.177}$$

$$s_1 = \sigma \tag{10.178}$$

$$x_j \in D(s_j)\ ,\ j = 1, 2, \dots, k \tag{10.179}$$

$$s_{j+1} = T(s_j, x_j)\ ,\ j = 1, 2, \dots, k \tag{10.180}$$

Formally, this model would be regarded a STATIONARY multistage decision model, that is: a multistage decision model where $D(j,s)$ and $T(j,s,x)$ are independent of the stage variable j.

This would also be the case if all the sequences of feasible decisions are non-truncated, where we would set $k=\infty$.

If the elements of $X(\sigma)$ are of varying lengths, the formulation would be as follows:

$$f(\sigma) := \underset{\substack{k \\ x_1,\dots,x_k}}{\text{opt}} \ g(s_1,x_1,x_2,\dots,x_k) \tag{10.181}$$

$$s_1 = \sigma \tag{10.182}$$

$$s_{k+1} \in S'' \tag{10.183}$$

$$x_j \in D(s_j) \ , \ j=1,2,\dots,k \tag{10.184}$$

$$s_{j+1} = T(s_j,x_j) \ , \ j=1,2,\dots,k \tag{10.185}$$

recalling that S'' denotes the set of TERMINAL STATES, namely states such that $D(s)$ is empty.

Observe that in this formulation S'' can be practically any subset of S, not necessarily the set of terminal states. The constraint $s_{k+1} \in S''$ simply requires the final state of the process to be an element of S''. This, by itself does not require $D(s_{k+1})$ to be empty.

Now, to formulate a dynamic programming model for *Problem* $P(\sigma)$ the objective function g would be decomposed in the usual dynamic programming fashion.

Definition 10.5.5 *Let (S,D,T,σ,g) be a sequential decision model. The objective function g is said to be* SEPARABLE *if there exists a function ρ on $S \times \mathbb{D} \times \mathbb{R}$ such that*

$$g(s,x) = \rho(s,x,g(T(s,x))) \ , \ \forall s \in S', x \in D(s) \tag{10.186}$$

$$g(s,x,\mathbf{z}) = \rho(s,x,g(T(s,x),\mathbf{z})) \tag{10.187}$$

for all $s \in S', x \in D(s), \mathbf{z} \in X(T(s,x))$.

We refer to ρ as a COMPOSITION FUNCTION *and to the pair (ρ,g) as a* DECOMPOSITION SCHEME.

Note that if $T(s,x) \in S''$, then $X(T(s,x)) = \{\varnothing\}$, and therefore $\mathbf{z} \in X(T(s,x))$ entails $\mathbf{z}=\varnothing$, in which case we have

$$g(s,x,\mathbf{z}) = \rho(s,x,g(T(s,x),\mathbf{z})) \tag{10.188}$$

$$= \rho(s,x,g(T(s,x),\varnothing)) \tag{10.189}$$

$$= \rho(s,x,g(T(s,x))) \tag{10.190}$$

The following example illustrates the construction of a decomposition scheme for a familiar sequential decision model.

10.5.4 Example

Consider the following familiar optimization problem:

$$\max_{x_1,x_2,x_3,\dots} \sum_{n=1}^{\infty} \beta^{n-1} w(x_n) \ , \ 0 < \beta < 1 \tag{10.191}$$

$$\sum_{n=1}^{\infty} x_n \leq V \ , \ V > 0 \tag{10.192}$$

$$x_n \geq 0 \ , \ n = 1, 2, 3, \dots \tag{10.193}$$

Now let

$$S = [0, V] \tag{10.194}$$

$$D(s) = [0, s] \ , \ s \in S; \implies S'' = \varnothing \tag{10.195}$$

$$T(s, x) = s - x \tag{10.196}$$

$$\sigma = V \tag{10.197}$$

$$g(s, x_1, x_2, \dots) = \sum_{n=1}^{\infty} \beta^{n-1} w(x_j) \ , \ 0 < \beta < 1 \tag{10.198}$$

Convince yourself that for this model $X(V)$ consists of all the feasible solutions to the optimization problem under consideration.

By construction,

$$g(s, x_1, x_2, x_3, \dots) = \sqrt{x_1} + \sum_{n=2}^{\infty} \beta^{n-1} \sqrt{x_n} \ , \ s \in S \tag{10.199}$$

$$= w(x_1) + \beta \sum_{n=1}^{\infty} \beta^{n-1} w(x_{n+1}) \tag{10.200}$$

$$= w(x_1) + \beta g(T(s, x_1), x_2, x_3, \dots) \tag{10.201}$$

So if we let

$$\rho(s, x, r) = \sqrt{x} + \beta r \ , \ s \in S, x \in D(s), r \in \mathbb{R} \tag{10.202}$$

we would have

$$g(s, x_1, x_2, x_3, \dots) = \rho(s, x_1, g(T(s, x_1), x_2, x_3, \dots))) \tag{10.203}$$

or more formally,

$$g(s, x, \mathbf{z}) = \rho(s, x, g(T(s, x), \mathbf{z}))) \tag{10.204}$$

for $s \in S', x \in D(s), \mathbf{z} \in X(T(s, x))$.

Thus, g is separable and ρ is a proper composition function. It should be noted that if the state $s = 0$ is reached then $D(0) = \{0\}$ and $T(0, 0) = 0$. This means that once this state is reached, the process will continue indefinitely even though there is no change in the state.

But there are admissible sequences of decisions that in effect will never reach this state: the sequences of states that they generate will converge to 0 but will never reach it. As indicated above, the sequence defined by $x_j = \frac{V}{2^j}$ is a case in point.

Now, suppose that the constraint (10.192) is replaced with an equality constraint:

$$\sum_{n=1}^{\infty} x_n = V \tag{10.205}$$

How would this affect the structure of the sequential decision model?

One of the models examined at a later stage is designed to handle situations of this kind. □

So, continuing in the dynamic programming fashion, assume that the objective function is separable. The modified and conditional problems associated with the sequential decision problem are defined as follows:

Definition 10.5.6 *In connection with Definition 10.5.5, define*

Problem $P(s), s \in S'$:

$$f(s) := \underset{\mathbf{z}\in X(s)}{\text{opt}} \; g(s,\mathbf{z}) \tag{10.206}$$

Problem $P(s,x), s \in S', x \in D(s)$:

$$f(s,x) := \underset{\substack{\mathbf{z}\in X(s') \\ s'=T(s,x)}}{\text{opt}} \; g(s,x,\mathbf{z}) \tag{10.207}$$

We shall refer to Problem $P(s)$ as the MODIFIED PROBLEM AT s *and to Problem $P(s,x)$ as the* CONDITIONAL PROBLEM AT (s,x). *Let $X^*(s)$ and $X^*(s,x)$ denote the set of optimal solutions to Problem $P(s)$ and Problem $P(s,x)$ respectively. We shall assume that all these problems are well-behaved in that each has at least one optimal solution.*

For the special case where s is a terminal state we define

$$f(s) := g(s) \;,\; s \in S'' \tag{10.208}$$

Clearly,

Lemma 10.5.1

$$f(s) = \underset{x\in D(s)}{\text{opt}} \; f(x,s) \;,\; \forall s \in S' \tag{10.209}$$

Next, to enable the derivation of the functional equation the familiar *Markovian* conditions are invoked.

Definition 10.5.7 *A decomposition scheme* (ρ, g) *is said to be* MARKOVIAN *if*

$$X^*(s,x) = X^*(T(s,x)) \ , \ \forall s \in S', x \in D(s) \tag{10.210}$$

Similarly, (ρ, g) *is said to be* WEAK-MARKOVIAN *if*

$$X^*(s,x) \cap X^*(T(s,x)) \neq \varnothing \ , \ \forall s \in S', x \in D(s) \tag{10.211}$$

Lemma 10.5.2 *If the sequential decision model is Weak-Markovian then*

$$f(s,x) = \rho(s,x,f(T(s,x))) \ , \ \forall s \in S', x \in D(s) \tag{10.212}$$

This, in conjunction with (10.5.1), yields:

Theorem 10.5.1 *Consider the case where the objective function of the sequential decision model is separable and the decomposition scheme* (g, ρ) *is Weak-Markovian. Then,*

$$f(s) = \operatorname*{opt}_{x \in D(s)} \rho(s,x,f(T(s,x))) \ , \ s \in S' \tag{10.213}$$

recalling that by definition $f(s) = g(s), \forall s \notin S'$.

This is the *dynamic programming functional equation* for our generic sequential decision problem.

Definition 10.5.8 *A* SEQUENTIAL DYNAMIC PROGRAMMING MODEL *is any collection* $(S, D, T, \sigma, g, \rho)$ *such that* (S, D, T, σ, g) *is a sequential decision model,* (ρ, g) *is a decomposition scheme, and the* FUNCTIONAL EQUATION *(10.213) holds.*

Let $D^\circ(s)$ *denote the subset of* $D(s)$ *whose elements are the solutions to (10.213), that is, define*

$$D^\circ(s) := \{x \in D(s) : \rho(s,x,f(T(s,x))) = f(s)\} \ , \ s \in S' \tag{10.214}$$

A MARKOVIAN DYNAMIC PROGRAMMING POLICY *is a map* δ *from* S' *to* $\mathbb{D}$ *such that*

$$\delta(s) \in D^\circ(s) \ , \ \forall s \in S' \tag{10.215}$$

Let Δ° *denote the set of dynamic programming policies associated with the functional equation (10.213). If the state transition model* (S, D, T) *is finite, then the sequential dynamic programming model is said to be* FINITE.

The principal difference between the dynamic programming model defined in this section and the multistage decision model formulated in *Chapter 3* is that the former makes no explicit reference to *stages*.

Still, stages can be easily incorporated in sequential decision models. That is, they can be incorporated as elements of the state variable. Specifically, a state can be expressed as a pair, say $s = (n, s')$, such that the first component would act as a stage. For this reason any optimization problem that can be expressed in terms of the multistage decision model is also amenable to the sequential decision model format. Obviously, the converse is also true.

What this means is that essentially the choice between these two formats is a matter of convenience.

Now, in the absence of an explicit stage variable there is no obvious way to distinguish between truncated and nontruncated sequential decision models. Let us then consider the following. For each $s \in S$ define

$$S^{(0)}(s) := \{s\} \tag{10.216}$$

and

$$S^{(n+1)}(s) := \{T(s', x) : s' \in S^{(n)}(s), x \in D(s')\} \tag{10.217}$$

Note that since $D(s) = \varnothing$, $\forall s \in S''$, it follows that

$$S^{(n)}(s) = \varnothing \ , \ \forall s \in S'', n \geq 1 \tag{10.218}$$

By construction then, $S^{(n)}(s)$ is the set of all the states that can be reached from state s in exactly n transitions. We may therefore maintain that a state transition model (S, D, T) is TRUNCATED if there exists a finite integer K such that

$$S^{(K+1)}(s) = \varnothing \ , \ \forall s \in S \tag{10.219}$$

Note that (10.219) implies that

$$s \notin S^{(n)}(s) \ , \ \forall s \in S, n \geq 1 \tag{10.220}$$

This means that truncated state transition models are *acyclic* in the sense that no sequence of decisions — other than the empty one — can regenerate a given state.

Note that if a state transition model is finite and acyclic, then if necessary, it is possible to label the states so that the label of state s will be a scalar $l(s)$ such that $l(T(s, x)) > l(s), \forall s \in S^{'}, x \in D(s)$. That is, in such cases we can label the states in an *increasing order*.

It is therefore convenient to assume that in cases where the state transition model is finite and acyclic the labeling of states is such that

$$l(T(s, x)) > l(s) \ , \ \forall s \in S', x \in D(s) \tag{10.221}$$
$$l(s'') > l(s') \ , \ \forall s'' \in S'', \ s' \in S' \tag{10.222}$$

In such cases the dynamic programming functional equation can be solved by the following simple iterative procedure.

10.5.5 Procedure

Step 1. Initialization
Set $F(s) = g(s), \ \forall s \in S''$.

Step 2. Iteration
For each $s \in S'$ — in decreasing order of $l(s)$ — Do:

$$F(s) = \operatorname*{opt}_{x \in D(s)} \rho(s, x, F(T(s,x))) \qquad \square \tag{10.223}$$

In other words, if the dynamic programming functional equation is valid and the model is finite and acyclic, then the function F recovered by this procedure is identical to the function f defined by (10.206), that is $F(s) = f(s), \forall s \in S$. Consequently, the sets $\{D^\circ(s)\}$ can be identified through the solution of (10.223), namely

$$D^\circ(s) = \{x \in D(s) : \rho(s, x, F(T(s,x))) = F(s)\} \ , \ s \in S' \tag{10.224}$$

The following example illustrates this procedure in action.

10.5.6 Example

Consider the directed graph in Figure 10.3. The figure on an arc stipulates the arc's length. The task is to identify the shortest path between node 1 and node 7, the length of a path being equal to the sum of the arcs' lengths on that path.

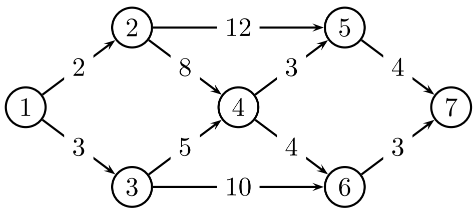


Figure 10.3: A simple acyclic sequential decision problem

Formally then, the sequential decision model representing this problem can be defined thus:

$$\begin{aligned}
S &= \{1,2,3,4,5,6,7\}; \mathbb{D} = \{2,3,4,5,6,7\}\\
D(1) &= \{2,3\}; D(2) = \{4,5\}; D(3) = \{4,6\}; D(4) = \{5,6\}\\
D(5) &= D(6) = \{7\}\\
D(7) &= \varnothing \Rightarrow S'' = \{7\}\\
T(s,x) &= x\\
\sigma &= 1
\end{aligned}$$

where $x \in D(s)$ is interpreted as the "next node" (state).

Next, the objective function g is of the regular additive form:

$$g(s_1, x_1, x_2, \dots, x_k) = \sum_{n=1}^{k} w(s_n, x_n) \ , \ s_{n+1} = T(s_n, x_n) \tag{10.225}$$

where $w(i, j)$ denotes the length of arc (i, j). Formally, we also let $g(7) = 0$.

Since this objective function is additive, we can set $\oplus = +$ so that the functional equation has the following form:

$$\begin{align} f(s) &= \min_{x \in D(s)} \{w(s, x) + f(T(s, x))\} \ , \ s \in S' = \{1, 2, 3, 4, 5, 6\} \tag{10.226} \\ &= \min_{x \in D(s)} \{w(s, x) + f(x)\} \tag{10.227} \end{align}$$

with $f(7) = g(7) = 0$.

As the graph representing the dynamic system under consideration is finite and acyclic, the functional equation can be solved iteratively as outlined above. Indeed, because the vertices are labeled in such a manner that $T(s, x) > s, \forall s \in S', x \in D(s)$, we can simply let $l(s) = s$ and enumerate the states in Step 2 of the procedure in *decreasing order.* That is, we would solve the functional equation for $s = 6, 5, 4, 3, 2, 1$ — in this order.

Here is a detailed description of the procedure:

$$\begin{align*} F(7) &= 0 \\ F(6) &= \min\{w(6, x) + F(x) : x \in D(6)\} \\ &= \min\{w(6, x) + F(x) : x \in \{7\}\} \\ &= \min\{w(6, 7) + 0\} = \min\{3 + 0\} = 3 \Rightarrow D^\circ(6) = \{7\} \\ F(5) &= \min\{w(5, x) + F(x) : x \in D(5)\} \\ &= w(5, 7) + 0 = 4 \Rightarrow D^\circ(5) = \{7\} \\ F(4) &= \min\{w(4, x) + F(x) : x \in D(4)\} \\ &= \min\{3 + 4, 4 + 3\} = \min\{7, 7\} = 7 \Rightarrow D^\circ(4) = \{5, 6\} \\ F(3) &= \min\{w(3, x) + F(x) : x \in D(3)\} \\ &= \min\{w(3, 4) + F(4), w(3, 6) + F(6)\} \\ &= \min\{5 + 7, 10 + 3\} = \min\{12, 13\} = 12 \Rightarrow D^\circ(3) = \{4\} \\ F(2) &= \min\{w(2, x) + F(x) : x \in D(2)\} \\ &= \min\{w(2, 4) + F(4), w(2, 5) + F(5)\} \\ &= \min\{8 + 7, 12 + 4\} = \min\{15, 16\} = 15 \Rightarrow D^\circ(2) = \{4\} \\ F(1) &= \min\{w(1, x) + F(x) : x \in D(1)\} \\ &= \min\{w(1, 2) + F(2), w(1, 3) + F(3)\} \\ &= \min\{2 + 15, 3 + 12\} = \min\{17, 15\} = 15 \Rightarrow D^\circ(1) = \{3\} \end{align*}$$

Therefore, the length of the shortest path from node 1 to node 7 is equal to $F(s) = 15$.

Note that the set of dynamic programming policies consists of two Markovian policies, namely $\Delta^\circ = \{\delta', \delta''\}$, where

$$
\begin{aligned}
&\delta'(1) = \delta''(1) = 3\\
&\delta'(2) = \delta''(2) = \delta'(3) = \delta''(3) = 4\\
&\delta'(4) = 5 \ ; \ \delta''(4) = 6\\
&\delta'(5) = \delta''(5) = \delta'(6) = \delta''(6) = 7
\end{aligned}
$$

Applying δ' to $s_1 = 1$ yields the optimal path $(1, 3, 4, 5, 7)$, whereas applying δ'' to s_1 yields the optimal path $(1, 3, 4, 6, 7)$. As we know by now, because the decomposition scheme is Markovian, the set of dynamic programming policies spans *all* the pertinent optimal paths.

On the other hand, suppose that the length of a path is equal to the length of the *shortest arc* on that path. In that case the objective function would be of this form

$$g(s_1, x_1, x_2, \dots, x_k) = \min_{1 \le n \le k} \{w(s_n, x_n)\} \tag{10.228}$$

Thus, using a $\lfloor$-decomposition scheme, the following functional equation would be obtained:

$$f(s) = \min_{x \in D(s)} \{w(s, x) \lfloor f(x)\} \ , \ s \in S' = \{1, 2, 3, 4, 5, 6\} \tag{10.229}$$

with $f(7) = g(7) = \infty$, recalling that $b \lfloor a := \min(b, a)$ and that the identity element of $\lfloor$ is equal to ∞. Solving this functional equation we recover the following results:

$$
\begin{aligned}
F(7) &= \infty\\
F(6) &= \min\{w(6, x) \lfloor F(x) : x \in D(6)\}\\
&= \min\{w(6, 7) \lfloor F(7)\}\\
&= 3 \lfloor \infty = 3 \Rightarrow D^\circ(6) = \{7\}\\
F(5) &= \min\{w(5, x) \lfloor F(x) : x \in D(5)\}\\
&= \min\{w(5, 7) \lfloor F(7)\}\\
&= 4 \lfloor \infty = 4 \Rightarrow D^\circ(5) = \{7\}\\
F(4) &= \min\{w(4, x) \lfloor F(x) : x \in D(4)\}\\
&= \min\{w(4, 5) \lfloor F(5), w(4, 6) \lfloor F(6)\}\\
&= \min\{3 \lfloor 4, 4 \lfloor 3\}\\
&= \min\{3, 3\} = 3 \Rightarrow D^\circ(4) = \{5, 6\}\\
F(3) &= \min\{w(3, x) \lfloor F(x) : x \in D(3)\}\\
&= \min\{w(3, 4) \lfloor F(4), w(3, 6) \lfloor F(6)\}\\
&= \min\{5 \lfloor 3, 10 \lfloor 3\}\\
&= \min\{3, 3\} = 3 \Rightarrow D^\circ(3) = \{4, 6\}
\end{aligned}
$$

$$\begin{aligned}
F(2) &= \min\{w(2,x)\lfloor F(x) : x \in D(2)\} \\
&= \min\{w(2,4)\lfloor F(4), w(3,5)\lfloor F(5)\} \\
&= \min\{8\lfloor 3, 12\lfloor 4\} \\
&= \min\{3,4\} = 3 \Rightarrow D^\circ(2) = \{4\}
\end{aligned}$$

$$\begin{aligned}
F(1) &= \min\{w(1,x)\lfloor F(x) : x \in D(1)\} \\
&= \min\{w(1,2)\lfloor F(2), w(1,3)\lfloor F(3)\} \\
&= \min\{2\lfloor 3, 3\lfloor 3\} \\
&= \min\{2,3\} = 2 \Rightarrow D^\circ(1) = \{2\}
\end{aligned}$$

As this equation is yielded by a Weak-Markovian scheme, all the dynamic programming policies are optimal in this case. Note, however, that the equation failed to find the optimal path $(1,2,5,7)$. This bears out the contention that there is no assurance that such an equation will recover all the optimal solutions to the problem in question. □

Now, if the dynamic system underlying the sequential decision model is nontruncated, the functional equation emanating from it would be solved by successive approximation as described in *Chapter 6.* But it will also prove possible — sometime even effective — to deploy a successive approximation method (and the truncation method) to solve a functional equation originating from a truncated model.

The following is a generic successive approximation procedure for the functional equation (10.213). $\mathcal{F}$ denotes a suitable initial approximation, as discussed in *Chapter 6.*

10.5.7 Procedure

Step 1. Initialization
Set $F(s) = \mathcal{F}(s), \ \forall s \in S$.

Step 2. Iteration
For each $s \in S'$ Do:

$$F(s) = \underset{x \in D(s)}{\text{opt}} \ \rho(s, x, F(T(s,x))) \tag{10.230}$$

and let $D^\circ(s)$ denote the set of optimal solutions obtained for this state.

Step 3. Stopping Rule
Stop if there has been no change in F. Otherwise, go to Step 2. □

The following example illustrates how this procedure would be applied to solve a truncated problem.

10.5.8 Example

We shall solve the functional equation associated with the problem featured in *Example 10.5.6,* namely

$$f(s) = \min_{x \in D(s)} \{w(s,x) + f(x)\} \ , \ s \in \{1,2,3,4,5,6\} \tag{10.231}$$

with $f(7) = 0$.

The sequential decision model is as follows:

$$\begin{aligned} S &= \{1,2,3,4,5,6,7\}; \mathbb{D} = \{2,3,4,5,6,7\} \\ D(1) &= \{2,3\}; D(2) = \{4,5\}; D(3) = \{4,6\}; D(4) = \{5,6\} \\ D(5) &= D(6) = \{7\} \\ D(7) &= \varnothing \Rightarrow S'' = \{7\} \\ T(s,x) &= x \\ \sigma &= 1 \end{aligned}$$

where $x \in D(s)$ is interpreted as the *next node* (state).

Since the objective function is additive, we set the initial approximation as follows:

$$\mathcal{F}(s) := \begin{cases} 0 & , \quad s \in S'' = \{7\} \\ \infty & , \quad s \in S' = \{1,2,3,4,5,6\} \end{cases} \tag{10.232}$$

So we set $F = \mathcal{F}$ and in the first iteration of the procedure we solve the following functional equation:

$$F(s) = \min_{x \in D(s)} \{w(s,x) + F(x)\} \tag{10.233}$$

for $s = 1,2,3,4,5,6$ — in this order. We obtain

$$\begin{aligned} &F(1) = F(2) = F(3) = F(4) = \infty \ ; \ F(5) = 4, F(6) = 3 \\ &D^\circ(1) = \{2,3\}, D^\circ(2) = \{4,5\}, D^\circ(3) = \{4,6\}, D^\circ(4) = \{5,6\} \\ &D^\circ(5) = D^\circ(6) = \{7\} \end{aligned}$$

Repeating this exercise in the second iteration, we obtain

$$\begin{aligned} &F(1) = \infty, F(2) = 16, F(3) = 13, F(4) = 7, F(5) = 4, F(6) = 3 \\ &D^\circ(1) = \{2,3\}, D^\circ(2) = \{5\}, D^\circ(3) = \{6\}, D^\circ(4) = \{5,6\} \\ &D^\circ(5) = D^\circ(6) = \{7\} \end{aligned}$$

Repeating this exercise in the third iteration, we obtain

$$\begin{aligned} &F(1) = 16, F(2) = 15, F(3) = 12, F(4) = 7, F(5) = 4, F(6) = 3 \\ &D^\circ(1) = \{3\}, D^\circ(2) = \{4\}, D^\circ(3) = \{4\}, D^\circ(4) = \{5,6\} \\ &D^\circ(5) = D^\circ(6) = \{7\} \end{aligned}$$

In the fourth iteration we obtain

$$\begin{aligned}&F(1) = 15, F(2) = 15, F(3) = 12, F(4) = 7, F(5) = 4, F(6) = 3\\&D^\circ(1) = \{3\}, D^\circ(2) = \{4\}, D^\circ(3) = \{4\}, D^\circ(4) = \{5, 6\}\\&D^\circ(5) = D^\circ(6) = \{7\}\end{aligned}$$

In the fifth iteration we obtain the same results, so we stop. The conclusion is that

$$\begin{aligned}&f(1) = F(1) = 15, f(2) = F(2) = 15, f(3) = F(3) = 12, f(4) = F(4) = 7,\\&f(5) = F(5) = 4, f(6) = F(6) = 3, f(7) = F(7) = 0\end{aligned}$$

We recover an optimal solution to the problem by tracing the optimal decisions obtained for the functional equation in the last iteration, starting at the initial state $s_1 = \sigma = 1$. In case of a tie, we select an arbitrary element of $D^\circ(s)$.

So, we select $x_1 = 3$ from $D^\circ(1)$. This takes us to state $s_2 = T(s_1, x_1) = x_1 = 3$. So we select $x_2 = 4$ from $D^\circ(3)$. This takes us to state $s_3 = T(s_2, x_2) = x_2 = 4$. So we select $x_3 = 5$ from $D^\circ(4)$. This takes us to state $s_4 = T(s_3, x_3) = x_3 = 5$. So we select $x_4 = 7$ from $D^\circ(5)$. This takes us to the terminal state $s = 7$, so we stop.

We have thus recovered the optimal solution $\mathbf{x} = (3, 4, 5, 7)$. Computing $g(\sigma, \mathbf{x}) = g(1, 3, 4, 5, 7)$ we obtain $f(1) = 15$ which is consistent with $F(1) = 15$.

Remark:

Note that had we decided in Step 2 of the procedure to enumerate the non-terminal states in *reverse order*, namely s=7,6,5,4,3,2,1, we would have completed the procedure in *one* iteration! □

At this point it is fitting to consider a situation where the optimal solution is *cyclic* in that it contains a loop. In such cases we can use *successive approximation methods* to solve the functional equation. However, as we shall see, there is no assurance that the optimal policy is Markovian.

10.5.9 Example

Consider the sequential decision problem represented by the graph depicted in Figure 10.4. The objective is to find the shortest path from node (state) 1 to node (state) 5 where the length of a path is equal to *the length of the shortest arc on the path.* For example, the length of the path $(1, 3, 4, 5)$ is equal to $\min\{3, 5, 1\} = 1$.

By inspection, the shortest arc is the one connecting node 4 to node 2, and its length is equal to 0. The shortest path from node 1 to node 5 should contain this arc. This means that node 4 leads to node 2. However, in this case the destination, namely node 5, is never reached.

The conclusion is therefore that the optimal policy is *not* Markovian: The

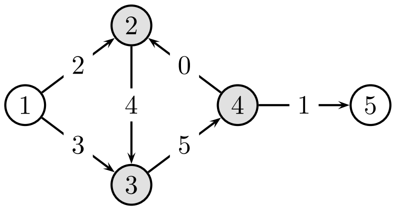


Figure 10.4: A cyclic problem

first transition to node 4 does not terminate the process by reaching node 5, rather it continues to node 2. The second transition to node 4 terminates the process. Note that there are infinitely many optimal paths: Node 4 can be reached infinitely many times so long as ultimately node 5 is reached.

Let us now examine how this problem is solved formally with the aid of a simple successive approximation procedure. For this purpose set

$$
\begin{aligned}
S = \mathbb{D} &= \{1,2,3,4,5\} \\
D(1) = \{2,3\}; D(2) &= \{3\}; D(3) = \{4\}; D(4) = \{2,5\}; D(5) = \varnothing \\
S' &= \{1,2,3,4\} \ ; \ S'' = \{5\} \\
T(s,x) &= x \ ; \ \sigma = 1 \\
g(s_1, x_1, x_2, \dots, x_k) &= \min_{1 \le n \le k} \{w(s_n, x_n)\} \qquad (10.234)
\end{aligned}
$$

where $w(i,j)$ denote the length of arc (i,j).

Using the binary operator $\lfloor$ we can rewrite the objective function as follows:

$$g(s_1, x_1, x_2, \dots, x_k) = w(s_1, x_1) \lfloor w(s_2, x_2) \lfloor \cdots \lfloor w(s_k, x_k) \qquad (10.235)$$

Also, let opt $=$ min and set $g(5) = \infty$ (the identity element of $\lfloor$). The functional equation is then

$$f(s) = \min_{x \in D(s)} \{w(s,x) \lfloor f(x)\} \ , \ s \in S' = \{1,2,3,4\} \qquad (10.236)$$

with $f(5) = g(5) = \infty$.

To enable $\oplus = \lfloor$, and us, identify an infeasible solution, it is modified slightly through an appeal to the following "$\star$ convention":

$$a \lfloor b := \begin{cases} \min\{a,b\} & , \ b \neq \star \\ \star & , \ b = \star \end{cases} \qquad (10.237)$$

That is, the symbol $\star$ denotes INFEASIBILITY[1]. The opt $=$ min operation prefers any real number to $\star$, hence $\min\{a, \star\} = a, \forall a \in \mathbb{R}$.

Note that should the result be $f(1) = \star$, namely the length of the shortest

[1]Note that the conventional *Big M* convention does not work here!

path from node 1 to node 5 is equal to $\star$, then the conclusion would be that node 5 cannot be reached from node 1.

This clear, we are ready to initiate the successive approximation procedure. Let us begin then with the following initial approximation for f:

$$\mathcal{F}(s) = \star \ , \ \forall s \in S' = \{1,2,3,4\} \tag{10.238}$$

with $\mathcal{F}(5) = g(5) = \infty$.

This initial approximation indicates that only state 5 leads to state 5 in 0 state transitions. To obtain the next approximation we solve

$$\begin{aligned} F(s) &= \min_{x \in D(s)} \rho(s,x,F(T(s,x))), \ \ s \in S' && (10.239)\\ &= \min_{x \in D(s)} w(s,x) \lfloor F(T(s,x)), \ \ s \in \{1,2,3,4\} && (10.240)\end{aligned}$$

So, doing our $\star$ calculus carefully we obtain

$$\begin{aligned} F(4) &= \min_{x \in D(4)} w(4,x) \lfloor F(T(4,x))\\ &= \min_{x \in \{2,5\}} w(4,x) \lfloor F(T(4,x))\\ &= \min \{w(4,2) \lfloor F(T(4,2)), w(4,5) \lfloor F(T(4,5))\}\\ &= \min \{0 \lfloor F(2), 1 \lfloor F(5)\} = \min \{0 \lfloor \star, 1 \lfloor \infty\} = \min \{\star, 1\}\\ &= 1 \ (\textit{So set } D^{(1)}(4) = \{5\})\\ F(3) &= \min_{x \in D(3)} w(3,x) \lfloor F(T(3,x))\\ &= \min_{x \in \{4\}} w(3,x) \lfloor F(T(3,x)) = w(3,4) \lfloor F(T(3,4)) = 5 \lfloor 1\\ &= 1 \ (\textit{So set } D^{(1)}(3) = \{4\})\\ F(2) &= \min_{x \in D(2)} w(2,x) \lfloor F(T(2,x))\\ &= \min_{x \in \{3\}} w(2,x) \lfloor F(T(2,x)) = w(2,3) \lfloor F(T(2,3)) = 4 \lfloor 1\\ &= 1 \ (\textit{So set } D^{(1)}(2) = \{3\})\\ F(1) &= \min_{x \in D(1)} w(1,x) \lfloor F(T(1,x))\\ &= \min_{x \in \{2,3\}} w(1,x) \lfloor F(T(1,x))\\ &= \min \{w(1,2) \lfloor F(T(1,2), w(1,3) \lfloor F(T(1,3))\}\\ &= \min \{2 \lfloor F(2), 3 \lfloor F(3)\} = \min \{2 \lfloor 1, 3 \lfloor 1\} = \min\{1,1\}\\ &= 1 \ (\textit{So set } D^{(1)}(1) = \{2,3\}) \end{aligned}$$

This completes the first iteration of the successive approximation procedure. Here $D^{(1)}$(s) denotes the set of optimal decisions for state s found in the first iteration of the successive approximation procedure.

Since this approximation is clearly different from the initial approximation, we continue.

$$\begin{aligned}
F(4) &= \min_{x\in D(4)} w(4,x)\lfloor F(T(4,x))\\
&= \min_{x\in\{2,5\}} w(4,x)\lfloor F(T(4,x))\\
&= \min\{w(4,2)\lfloor F(T(4,2)), w(4,5)\lfloor F(T(4,5))\}\\
&= \min\{0\lfloor F(4), 1\lfloor F(5)\} = \min\{0\lfloor 1, 1\lfloor\infty\} = \min\{0,1\}\\
&= 0\ (\textit{So set } D^{(2)}(4)=\{2\})\\
F(3) &= \min_{x\in D(3)} w(3,x)\lfloor F(T(3,x))\\
&= \min_{x\in\{4\}} w(3,x)\lfloor F(T(3,x)) = w(3,4)\lfloor F(T(3,4)) = 0\lfloor 1\\
&= 0\ (\textit{So set } D^{(2)}(3)=\{4\})\\
F(2) &= \min_{x\in D(2)} w(2,x)\lfloor F(T(2,x))\\
&= \min_{x\in\{3\}} w(2,x)\lfloor F(T(2,x)) = w(2,3)\lfloor F(T(2,3)) = 4\lfloor F(3) = 4\lfloor 0\\
&= 0\ (\textit{So set } D^{(2)}(2)=\{3\})\\
F(1) &= \min_{x\in D(1)} w(1,x)\lfloor F(T(1,x))\\
&= \min_{x\in\{2,3\}} w(1,x)\lfloor F(T(1,x))\\
&= \min\{w(1,2)\lfloor F(T(1,2), w(1,3)\lfloor F(T(1,3))\}\\
&= \min\{2\lfloor F(2), 3\lfloor F(3)\}\\
&= \min\{2\lfloor 0, 3\lfloor 0\}\\
&= \min\{0,0\}\\
&= 0\ (\textit{So set } D^{(2)}(1)=\{2,3\})
\end{aligned}$$

Repeating this exercise once more shows that F cannot be improved, so we stop. The conclusion is that

$$f = F \Longrightarrow f(1)=f(2)=f(3)=f(4)=0, f(5)=\infty \tag{10.241}$$

Our next task is to find an optimal solution to the problem and an optimal policy, keeping in mind that as the process is cyclic, the optimal policy would not necessarily be Markovian.

So, we check the optimal decision for the initial state $s_1=\sigma=1$ at the last iteration of the procedure.

Since $D^{(2)}(1)=\{2,3\}$ the choice is between 2 and 3. So let $x_1=2$, and therefore $s_2=T(s_1,x_1)=x_1=2$. Since $D^{(2)}(2)=\{3\}$ we set $x_2=3$ and $s_3=T(s_2,x_2)=x_2=3$. Since $D^{(2)}(3)=\{4\}$ we set $x_3=4$ and $s_4=T(s_3,x_3)=x_3=4$. Since $D^{(2)}(4)=\{2\}$ we set $x_4=2$ and $s_5=T(s_4,x_4)=x_4=2$.

We are thus in a cycle, and must therefore select optimal decisions from the sets $D^{(1)}(s)$ generated in the first iteration of the procedure, starting with $s = s_5 = 2$.

Since $D^{(1)}(2) = \{3\}$ we set $x_5 = 3$ and $s_6 = T(s_5, x_5) = x_5 = 3$. Since $D^{(1)}(3) = \{4\}$ we set $x_6 = 4$ and $s_7 = T(s_6, x_6) = x_6 = 4$. Since $D^{(1)}(4) = \{5\}$ we set $x_7 = 5$ and $s_8 = T(s_7, x_7) = x_7 = 5$.

Having reached a terminal state we stop. The optimal path we recovered is then $(1, 2, 3, 4, 2, 3, 4, 5)$. There are of course infinitely many other optimal solutions.

Note that all the optimal solutions are cyclic and non-Markovian. However, the model can be modified by incorporating a counter in the state variable — operating like a stage variable — that would stipulate the number of state-transitions that have been carried out. In such a model the optimal policies will be Markovian with respect to the expanded state variable. Still, these solutions will not be stationary. Indeed, the multistage model will require a *stopping rule* to terminate the process. The forward *Push* method discussed in *Chapter 15* is particularly suitable for problems of this kind.

Remark:
Note that anticipating a cyclic optimal path, the solution procedure was slightly modified, using $D^{(k)}(s)$ rather than $D^{\circ}(s)$ to record the optimal solutions to the dynamic programming functional equation. □

The next example features an infeasible problem, namely a problem with no feasible solutions.

10.6 Example

Consider the shortest path problem depicted in Figure 10.5. The goal is to find a path from state (node) 1 to state (node) 5 where the length of a path is the length of the longest arc on the path.

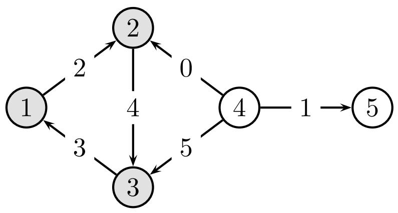


Figure 10.5: An infeasible acyclic problem

It is clear, by inspection, that the problem has no feasible solution because node 5 cannot be reached from node 1. The objective of the exercise is to illustrate that dynamic programming algorithms can handle such cases.

So, set

$$\begin{aligned} S = \mathbb{D} &= \{1,2,3,4,5\};\ S^{'} = \{1,2,3,4\}\ ;\ S'' = \{5\} \\ D(1) &= \{2\}; D(2) = \{3\}; D(3) = \{1\}; D(4) = \{2,3\}; D(5) = \varnothing \\ T(s,x) &= x \\ \sigma &= 1 \end{aligned}$$

and

$$g(s_1, x_1, x_2, \dots, x_k) = \max_{1\leq n\leq k}\{w(s_n, x_n)\} \tag{10.242}$$

$$= w(s_1,x_1)\lceil w(s_2,x_2)\lceil \cdots \lceil w(s_k,x_k) \tag{10.243}$$

where $w(i,j)$ denote the length of arc (i,j) and $\lceil$ denotes the binary max operation. Also, let opt = max and set $g(5) = -\infty$ (the identity element of $\lceil$).

Now, because $\oplus = \lceil$ violates the *Big M* convention, the definition of $\lceil$ has to be modified slightly so that the $\star$ convention is used as follows:

$$a\lfloor b := \begin{cases} \max\{a,b\} & ,\ b \neq \star \\ \quad\star & ,\ b = \star \end{cases} \tag{10.244}$$

That is, $\star$ denotes infeasibility.

Thus, should the result be $f(1) = \star$, the conclusion would be that node 5 cannot be reached from node 1.

This does not affect the operation of opt = max, namely $\max\{\star, a\} = a$ for any $a \in \mathbb{R}$. The functional equation is as follows:

$$f(s) = \max_{x\in D(s)} w(s,x)\lceil f(x)\ ,\ s \in \{1,2,3,4\} \tag{10.245}$$

with $f(5) = -\infty$.

We are ready now to initiate the successive approximation procedure, so we begin with the following initial approximation for f:

$$\mathcal{F}(s) = \star\ ,\ \forall s \in S' = \{1,2,3,4\} \tag{10.246}$$

with $\mathcal{F}(5) = g(5) = -\infty$.

This initial approximation indicates that state 5 leads to state 5 in 0 state transitions. To obtain the next approximation we now solve

$$F(s) = \max_{x\in D(s)} w(s,x)\lceil F(T(s,x)),\ s \in \{1,2,3,4\} \tag{10.247}$$

So, doing our $\star$ calculus carefully we obtain

$$\begin{aligned}
F(4) &= \max_{x\in D(4)} w(4,x)\lceil F(T(4,x)) \\
&= \max_{x\in\{2,3,5\}} w(4,x)\lceil F(T(4,x)) \\
&= \max\{w(4,2)\lceil F(T(4,2)), w(4,3)\lceil F(T(4,3)), w(4,5)\lceil F(T(4,5))\} \\
&= \min\{0\lceil F(2), 5\lceil F(3), 1\lceil F(5)\} \\
&= \max\{0\lceil F(2), 5\lceil F(3), 1\lceil F(5)\} \\
&= \max\{0\lceil\star, 5\lceil\star, 1\lceil -\infty\} \\
&= \max\{\star,\star,1\ \} = 1\ (\textit{So set } D^{\circ}(4) = \{5\})
\end{aligned}$$

$$\begin{aligned}
F(3) &= \max_{x\in D(3)} w(3,x)\lceil F(T(3,x)) \\
&= \max_{x\in\{1\}} w(3,x)\lceil F(T(3,x)) \\
&= w(3,1)\lceil F(T(3,1)) = 3\lceil F(1) \\
&= \star\ (\textit{So set } D^{\circ}(3) = \varnothing) \\
F(2) &= \max_{x\in D(2)} w(2,x)\lceil F(T(2,x)) \\
&= \max_{x\in\{3\}} w(2,x)\lceil F(T(2,x)) = w(2,3)\lceil F(T(2,3)) = 4\lceil F(3) = 4\lfloor\star \\
&= \star\ (\textit{So set } D^{\circ}(2) = \varnothing) \\
F(1) &= \max_{x\in D(1)} w(1,x)\lceil F(T(1,x)) \\
&= \max_{x\in\{2\}} w(1,x)\lceil F(T(1,x)) = w(1,2)\lceil F(T(1,2)) = 2\lceil\star \\
&= \star\ (\textit{So set } D^{\circ}(1) = \varnothing)
\end{aligned}$$

This completes the first approximation. Since this approximation is clearly different from the initial approximation, we proceed to the second iteration.

$$\begin{aligned}
F(4) &= \max_{x\in D(4)} w(4,x)\lceil F(T(4,x)) \\
&= \max_{x\in\{2,3,5\}} w(4,x)\lceil F(T(4,x)) \\
&= \max\{w(4,2)\lceil F(T(4,2)), w(4,3)\lceil F(T(4,3)), w(4,5)\lceil F(T(4,5))\} \\
&= \max\{0\lceil F(2), 5\lceil F(3), 1\lceil F(5)\} \\
&= \max\{0\lceil F(2), 5\lceil F(3), 1\lceil F(5)\} \\
&= \max\{0\lceil\star, 5\lceil\star, 1\lceil -\infty\} \\
&= \max\{\star,\star,1\ \} = 1\ (\textit{So set } D^{\circ}(4) = \{5\}) \\
F(3) &= \max_{x\in D(3)} w(3,x)\lceil F(T(3,x)) \\
&= \max_{x\in\{1\}} w(3,x)\lceil F(T(3,x)) = 3\lceil F(1) = 3\lceil\star \\
&= \star\ (\textit{So set } D^{(2)}(3) = \varnothing)
\end{aligned}$$

$$\begin{aligned}
F(2) &= \min_{x\in D(2)} w(2,x)\lceil F(T(2,x)) \\
&= \min_{x\in\{3\}} w(2,x)\lceil F(T(2,x)) \\
&= w(2,3)\lceil F(T(2,3)) \\
&= 4\lceil F(3) \\
&= 4\lceil \star \\
&= \star \ (\textit{So set } D^\circ(2) = \varnothing) \\
F(1) &= \max_{x\in D(1)} w(1,x)\lceil F(T(1,x)) \\
&= \max_{x\in\{2\}} w(1,x)\lceil F(T(1,x)) \\
&= w(1,2)\lceil F(T(1,2)) \\
&= 2\lceil \star \\
&= \star \ (\textit{So set } D^\circ(2) = \varnothing)
\end{aligned}$$

As the approximation remains unchanged, we stop. And since $F(1) = \star$ we conclude that the problem has no feasible solution. □

Next on the agenda is a slightly more complex model where the process can be terminated in spite of its states being non-terminal.

10.6.1 Stopping Rules

To terminate the process at a $s \in S$ such that $D(s)$ is not empty, we need a means to halt it at this state. The following model is designed with this requirement in mind.

Definition 10.6.1 *A* SEQUENTIAL DECISION MODEL WITH STOPPING STATES *is a collection $(S, D, T, \sigma, g, S''')$ where:*

- *(S, D, T) is a state transition model.*
- *σ is an element of S.*
- *g is a real-valued function on S and on $\{\sigma\} \times X(\sigma)$.*
- *S''' is a subset of S'.*

The state σ is understood to be the INITIAL STATE *of the process, and the states in S''' are called* STOPPING STATES. *At these states terminating the process is permitted. Recall that S' is the subset of S consisting of states at which $D(s)$ is not empty.*

So when a state $s \in S'''$ is reached there are two alternatives: we can either terminate the process at this state, or we can select a decision $d \in D(s)$ and let the process continue.

The definition of $X(s)$ must be adjusted accordingly. For $s \in S'''$ we have

$$X(s) = \{\varnothing\} \bigcup \{(x, \mathbf{z}) : x \in D(s), \mathbf{z} \in X(T(s,x))\} \tag{10.248}$$

Insofar as the modified problems are concerned, the situation is as follows. If $s \notin S'''$, then the situation is simple, namely if $s \in S'$ then the process must continue, whereas if $s \in S''$ the process must terminate. If $s \in S'''$ then the situation is more complicated:

· At state $s \in S' \backslash S'''$.
At this state the decision making process must continue. Hence, the modified problem is as follows:

Problem $P(s)$: $s \in S' \backslash S'''$:

$$f(s) = \operatorname*{opt}_{\mathbf{x} \in X(s)} g(s, \mathbf{x}) \tag{10.249}$$

If the objective function is separable and the composition function is Weak-Markovian, it would follow that

$$f(s) = \operatorname*{opt}_{s \in D(s)} \rho(s, x, f(T(s, x))) \tag{10.250}$$

This is the familiar dynamic programming equation obtained for situations devoid of stopping states.

· At state $s \in S' \cap S'''$.
Formally, the decision-making situation at such a state can be described as follows:

Problem $P(s)$: $s \in S''' \backslash S''$:

$$f(s) := \operatorname{opt} \left\{ \overset{\text{stop}}{g(s)} \ , \ \overset{\text{continue}}{\operatorname*{opt}_{\mathbf{x} \in X(s)} g(s, \mathbf{x})} \right\} \tag{10.251}$$

That is, if a non-terminal stopping state is reached, either the process stops, in which case the cost/benefit incurred/gained is $g(s)$. Or, the process continues, in which case the idea is to obtain the best out of the remaining part of the process, by optimizing $g(s, \mathbf{x})$ with respect to $\mathbf{x}$ over $X(s)$.

The following result is then a consequence of incorporating stopping states in the sequential decision model.

Theorem 10.6.1 *Consider a sequential decision model with stopping states. If the objective function is separable and Weak-Markovian then*

$$f(s) = \begin{cases} g(s) & , \quad s \in S'' \\ \operatorname*{opt}_{x \in D(s)} \rho(s, x, f(T(s, x))) & , \quad s \in S' \backslash S''' \\ \operatorname{opt} \left\{ g(s) \ , \ \operatorname*{opt}_{x \in D(s)} \rho(s, x, f(T(s, x))) \right\} & , \quad s \in S' \cap S''' \end{cases} \tag{10.252}$$

Proof. In the first case s is a terminal state. The process therefore terminates at this state, hence by definition $f(s) = g(s)$.

The second case follows, as usual, from the Weak-Markovian condition: s is non-terminal and a non-stopping state. So here we must select a decision from $D(s)$. Hence $f(s) = \rho(s, x, f(T(s, x)))$ for some $x \in D(s)$. The best decision is that which optimizes the conditional problem at (s, x).

Lastly, if s is a non-terminal stopping state, we either stop and take the reward $g(s)$, or we continue and select the next decision in the usual fashion. The idea is to select the best out of these two options. □

The following example illustrates a simple application of a dynamic programming model with stopping states.

10.6.2 Example

Consider the problem depicted in Figure 10.6 and the following story behind it. Traversing the directed arcs of the graph, and starting at node 1, when a new node is reached, the reward gained is the amount (Australian \$) listed above the node. If an arc is traversed, the penalty incurred is the amount (Australian \$) shown on the arc. The process can terminate at any node other than node 1. What tour maximizes the total net reward?

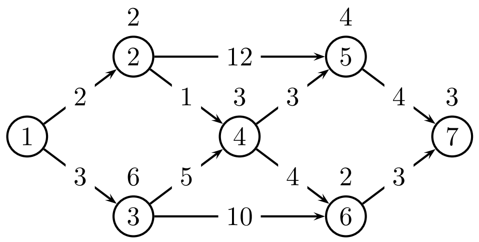


Figure 10.6: A simple acyclic sequential decision problem with stopping states

Consider the following model:

$$
\begin{aligned}
S = \mathbb{D} &= \{1,2,3,4,5,6,7\}\\
D(1) = \{2,3\},\ D(2) &= \{4,5\},\ D(3) = \{4,6\},\ D(4) = \{5,6\}\\
D(5) = D(6) &= \{7\},\ D(7) = \varnothing \Longrightarrow S'' = \{7\}\\
T(s,x) &= x\\
\sigma = 1\ &;S' = \{1,2,3,4,5,6\}\ ;\ S''' = \{2,3,4,5,6,7\}
\end{aligned}
$$

$$
g(s, x_1, x_2, \dots, x_k) = \sum_{n=1}^{k} [b(x_n) - c(s_n, x_n)] \tag{10.253}
$$

where $s_1 = s, s_{n+1} = T(s_n, x_n), n = 1, 2, \dots, k$ and $g(s) = 0, s \in S''$, where $b(j)$ denotes the reward at node j and $c(i, j)$ denotes the cost of traversing arc (i, j).

Since all states except $s = 1$ are stopping states, the dynamic programming functional equation is as follows:

$$f(s) = \begin{cases} 0 & , \quad s = 7 \\ \max\limits_{x \in D(s)} \{b(x) - c(s,x) + f(x)\} & , \quad s = 1 \\ \max\left\{0, \max\limits_{x \in D(s)} [b(x) - c(s,x) + f(x)]\right\} & , \quad s = \{1,2,3,4,5,6\} \end{cases} \tag{10.254}$$

As the model is acyclic, this functional equation can be solved for $s = 1,2,3,4,5,6,7$ — in reverse order — using the direct method. So, let us do it in an orderly fashion.

Remark:
The symbol ! will denote the decision to stop (at a non-terminal stopping state). Should this be an optimal decision, it will be included in the set of optimal decisions $D^\circ(s)$ pertaining to that state.

· $s = 7$: This is a terminal, non-stopping state. So, we stop. Hence, $f(7) = g(7) = 0$.

· $s = 6$: This is a non-terminal stopping state. Hence,

$$\begin{aligned} f(6) &= \max\left\{0, \max_{x \in D(6)} \{b(x) - c(6,x) + f(x)\}\right\} \\ &= \max\left\{0, \max_{x \in \{7\}} \{b(x) - c(6,x) + f(x)\}\right\} \\ &= \max\{0, b(7) - c(6,7) + f(7)\} = \max\{0, 3 - 3 + 0\} = \max\{0,0\} \\ &= 0 \text{ (we can either stop here, or go to 7, } D^\circ(6) = \{!,7\}) \end{aligned}$$

· $s = 5$: This is a non-terminal stopping state. Hence,

$$\begin{aligned} f(5) &= \max\left\{0, \max_{x \in D(5)} \{b(x) - c(5,x) + f(x)\}\right\} \\ &= \max\left\{0, \max_{x \in \{7\}} \{b(x) - c(5,x) + f(x)\}\right\} \\ &= \max\{0, b(7) - c(5,7) + f(7)\} = \max\{0, 3 - 4 + 0\} \quad = \max\{0,-1\} \\ &= 0 \text{ (we stop here, set } D^\circ(6) = \{!\}) \end{aligned}$$

· $s = 4$: This is a non-terminal stopping state. Hence,

$$\begin{aligned} f(4) &= \max\left\{0, \max_{x \in D(4)} [b(x) - c(4,x) + f(x)]\right\} \\ &= \max\left\{0, \max_{x \in \{5,6\}} [b(x) - c(4,x) + f(x)]\right\} \end{aligned}$$

$$\begin{aligned}
&= \max\{0, \max\{b(5) - c(4,5) + f(5), b(6) - c(4,6) + f(6)\}\}\\
&= \max\{0, 4 - 3 + 0, 2 - 4 + 0\} = \max\{0, 1, -2\}\\
&= 1 \text{ (we go to 5, set } D^{\circ}(4) = \{5\})
\end{aligned}$$

· $s = 3$: This is a non-terminal stopping state. Hence,

$$\begin{aligned}
f(3) &= \max\left\{0, \max_{x\in D(3)} [b(x) - c(3,x) + f(x)]\right\}\\
&= \max\left\{0, \max_{x\in \{4,6\}} [b(x) - c(3,x) + f(x)]\right\}\\
&= \max\{0, \max\{b(4) - c(3,4) + f(4), b(6) - c(3,6) + f(6)\}\}\\
&= \max\{0, 3 - 5 + 1, 2 - 10 + 0\} = \max\{0, -1, -8\}\\
&= 0 \text{ (we stop here, set } D^{\circ}(4) = \{!\})
\end{aligned}$$

· $s = 2$: This is a non-terminal stopping state. Hence,

$$\begin{aligned}
f(2) &= \max\left\{0, \max_{x\in D(2)} [b(x) - c(2,x) + f(x)]\right\}\\
&= \max\left\{0, \max_{x\in \{4,5\}} [b(x) - c(2,x) + f(x)]\right\}\\
&= \max\{0, \max\{b(4) - c(2,4) + f(4), b(5) - c(2,5) + f(5)\}\}\\
&= \max\{0, 3 - 1 + 1, 4 - 12 + 0\} = \max\{0, 3, -8\}\\
&= 3 \text{ (we go to 4, } D^{\circ}(2) = \{4\})
\end{aligned}$$

· $s = 1$: This is a non-terminal non-stopping state. Hence,

$$\begin{aligned}
f(1) &= \max_{x\in D(1)} [b(x) - c(2,x) + f(x)]\\
&= \max_{x\in \{1,2\}} [b(x) - c(2,x) + f(x)]\\
&= \max\{b(2) - c(1,2) + f(2), b(3) - c(2,3) + f(3)\}\\
&= \max\{2 - 2 + 3, 6 - 3 + 0\} = \max\{3, 3\}\\
&= 3 \text{ (we go to 3, } D^{\circ}(2) = \{2,3\})
\end{aligned}$$

The recovery procedure generates two optimal paths, namely $(1,3)$ and $(1,2,4,5)$ both yielding AUD\$3.

The first runs from node 1 to node 3 then stops, the other leads to node 2, then node 4, then node 5 and then stops. □

Note that in certain cases the complication in (10.252) due to the existence of stopping states can be ignored. For instance, if opt = max, $g(s) = 0, \forall s \notin S'$ and $\rho(s,x,a) \geq 0$ for all relevant values of s, x and a, then (10.252) yields

$$f(s) = \begin{cases} g(s) & , \quad s \in S'' \\ \max\limits_{x\in D(s)} \rho(s,x,f(T(s,x))) & , \quad s \notin S' \end{cases} \tag{10.255}$$

Next on the agenda is the most famous instance of the sequential decision model discussed above.

10.7 Shortest Path Model

My account of the sequential decision model has made it clear that dynamic programming's formulation of certain optimization problems intuitively extends into a graphic representation. One can hardly imagine therefore a problem better suited for dynamic programming than the classic discrete optimization problem: the *shortest path problem.*

Clearly, this problem most naturally lends itself to dynamic programming as the task set by it is to identify the shortest path between two given vertices of a directed graph.

To be able to state the shortest path problem in terms of the sequential decision model the latter must have the following ingredients:

· A finite *state space,* S.

· The set of *decisions* associated with state s, denoted $D(s)$, must be a subset of S (allowed to be empty).

· A *transition function* T of this form: $T(s,x)=x$.

· An *initial state* $\sigma \in S$.

· A *destination* state $\hat{s} \in S$.

As we have already seen, in this case $D(s)$ denotes the set of *immediate successors* of node s.

Insofar as the objective function is concerned, often, but not always, g will admit of the following representation:

$$g(s_1,x_1,x_2,\dots,x_k) = w(s_1,x_1) \oplus w(x_1,x_2) \oplus \cdots \oplus w(x_{k-1},x_k) \tag{10.256}$$

where w is a real-valued function on $S \times S$, and $\oplus$ is a binary composition operator.

In this case $w(i,j)$ will be viewed as the length of $arc(i,j)$ and the value yielded by $g(x_1,x_2,\dots,x_k)$ as the 'length' of the path $(x_1,x_2,\dots,x_k)$. The objective will be to find the shortest or longest path from the initial state σ to the destination node $\hat{s}$.

To ensure that the optimal path $(x_1,x_2,\dots,x_k)$ indeed reaches the destination state $\hat{s}$, the requirement is that $x_k=\hat{s}$. Hence, the generic shortest path problem would be formulated as follows:

$$\min_{\substack{k \\ x_1,\dots,x_k}} \{w(\sigma,x_1) \oplus w(x_1,x_2) \oplus w(x_2,x_3) \oplus \cdots \oplus w(x_{k-1},x_k)\} \tag{10.257}$$

$$x_k = \hat{s} \tag{10.258}$$

$$x_j \in D(x_{j-1}) \ , \ j=2,3,\dots,k \tag{10.259}$$

Observe that placing k below min is a gentle reminder that the value of k is not fixed in advance so that strictly speaking it should be regarded a *decision variable* in this context.

It should also be noted that a slightly different formulation, where the state variable is stated explicitly is also an option namely:

$$\min_{\substack{k \\ x_1,\dots,x_k}} \{w(s_1,x_1) \oplus w(s_2,x_2) \oplus \cdots \oplus w(s_k,x_k)\} \tag{10.260}$$

$$x_j \in D(s_j)\ ,\ j = 1,2,3,\dots,k \tag{10.261}$$

$$s_{j+1} = x_j\ ,\ j = 1,2,3,\dots,k \tag{10.262}$$

$$s_{k+1} = \hat{s} \tag{10.263}$$

$$s_1 = \sigma \tag{10.264}$$

This formulation makes explicit that this is a sequential decision model whose state transition function is of the form $T(s,x) = x$. In this formulation s_j represents the "current node" and x_j represents the "current decision" which is . . . the next node to be visited.

As for representing the shortest path problem in terms of a graph, the ingredients of a typical shortest path model are:

· A directed graph.
· The lengths of the arcs of the graph.
· A path-length composition operator.
· A node of origin.
· A destination node.
· The relevant instance of opt (max or min).

Hence, stated formally:

Definition 10.7.1 *A* SHORTEST PATH MODEL *is a sequential decision model with stopping states,* $(S, D, T, \sigma, g, S''')$*, possessing the following properties:*

· *The state space,* S*, and the decision space* $\mathbb{D}$ *are finite.*
· *The transition function has the form* $T(s,x) = x, s \in S, x \in D(s)$*.*
· *The set of stopping states,* S'''*, is a singleton,* $S''' = \{\hat{s}\}$*. We refer to* $\hat{s}$ *as the destination state.*

The problem posed by this model is then this: find the shortest path from the initial state σ to the destination state $\hat{s}$.

If the objective function is of the form given by (10.256), and the decomposition operator is Weak-Markovian, then the dynamic programming functional equation associated with this model is as follows:

$$f(s) = \operatorname*{opt}_{x \in D(s)} \{w(s,x) \oplus f(x)\}\ ,\ s \in S \backslash \{\hat{s}\} \tag{10.265}$$

with

$$f(\hat{s}) = R_{\oplus} \tag{10.266}$$

where $R_{\oplus}$ denotes the right identity element of $\oplus$ and $f(s)$ denotes the length of the shortest path from the origin node σ to node s.

This model is a simplified version of the generic sequential decision model. In both cases the basic characteristic is: a transition function of the form $T(s,x) = x$.

Remark:
The structure of the transition function T excludes situations where the graph depicting the shortest path model has multiple arcs between a pair of nodes. But this is no more than a simple modeling technicality that can be easily dealt with. There are two ways to handle such cases.

· Ignore (delete) *inferior* arcs.

· Use *dummy* nodes and arcs.

Consider for instance the situation shown in Figure 10.7 observing that there are two arcs from node 6 to node 7. One is of length 2 and one is of length 3.

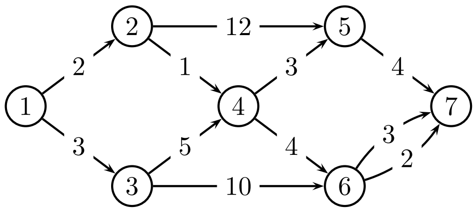


Figure 10.7: Illegal multiple arcs between a pair of nodes

If opt = min and the objective function is additive, then clearly the shorter arc is superior. Therefore, the situation would be dealt with by ignoring the arc from node 6 to node 7 whose length is equal to 3.

Alternatively, a dummy node and a dummy arc can be incorporated as shown in Figure 10.8. The length of the dummy arc must not affect the length of a path containing it. If the objective function is of the type given by (10.256), then we can set the length of the dummy arc to the identity element of $\oplus$. For instance, if $\oplus = +$ then the length of this arc can be set to 0.

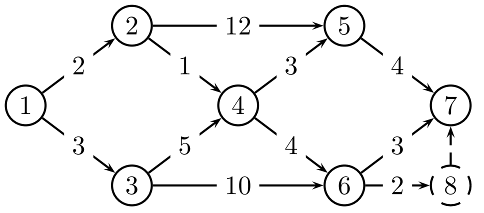


Figure 10.8: Using a dummy node and a dummy arc

Note that since the shortest path model allows cycles, there is no conflict

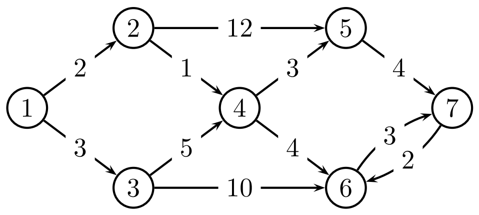


Figure 10.9: Legal multiple arcs between a pair of nodes

between a directed (i,j) arc and a direct (j,i) arc. For instance, the example shown in Figure 10.9 is not problematic. □

The purpose of the next example is to show how successive approximation methods identify an *unbounded* shortest path problem.

10.7.1 Example

Consider the shortest path problem depicted in Figure 10.10 observing that it is an acyclic problem and that arc(4,2) has a negative length. Now, suppose that the goal is to find the shortest path from node 1 to node 5 and the objective function is additive, namely the length of a path is equal to the sum of the arcs' lengths on the path.

By inspection it is clear that there is no optimal solution in this case: for any path of a finite length there is a path (via a cycle through nodes $(4,2,3)$) that is shorter.

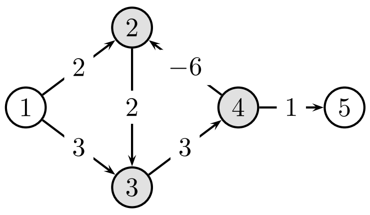


Figure 10.10: An unbounded cyclic shortest path problem

Let us ascertain that the conventional successive approximation method is indeed capable of identifying this fact.

So, in the accepted fashion we start with the initial approximation

$$\mathcal{F}(s) = \infty \ , \ s = 1,2,3,4 \tag{10.267}$$

and $\mathcal{F}(5) = 0$.

We then set $F = \mathcal{F}$ and solve the following functional equation until a fixed point is attained:

$$F(s) = \min_{x \in D(s)} \{w(s,x) \lfloor F(x)\} \ , \ s = \{1,2,3,4\} \tag{10.268}$$

where $F(s)$ is an approximation of $f(s)$ = length of shortest path from node s to node 5. Clearly $f(5) = 0$ so we also set $F(5) = 0$.

We then solve this equation in reverse order, starting with $s = 4$. The first iteration yields

$$F(4) = 1 \ , \ F(3) = 4 \ , \ F(2) = 6 \ , \ F(1) = 7$$

After the second iteration we have

$$F(4) = 0 \ , \ F(3) = 3 \ , \ F(2) = 5 \ , \ F(4) = 0$$

Next, we obtain

$$F(4) = -1 \ , \ F(3) = 2 \ , \ F(2) = 4 \ , \ F(1) = 5$$

and then

$$F(4) = -2 \ , \ F(3) = 1 \ , \ F(2) = 3 \ , \ F(1) = 4$$

and then

$$F(4) = -3 \ , \ F(3) = 0 \ , \ F(2) = 2 \ , \ F(1) = 3$$

and then

$$F(4) = -4 \ , \ F(3) = -1 \ , \ F(2) = 1 \ , \ F(1) = 2$$

As the iterations go through only 4 states, and no fixed point is attained after 6 iterations, we conclude that the process is cyclic and that the length of the cycle is -1. This means that were the procedure to continue, the value of $F(1)$ would decrease by 1 unit in each iteration. The implication is then that the objective function is unbounded (below) so that there is no optimal solution. □

You will recall that comparing in *Section 3.2 Problem P* to *Problem P*(s), I noted that the merit of the latter is in the extremely accommodating framework that it provides for problem formulation. Similar remarks apply to the shortest path model. Although it is starkly simple, the shortest path model furnishes an extremely flexible framework for problem formulation. Its greatest asset is in the state space S — and therefore the decision space $\mathbb{D}$ — being abstract objects namely, devoid of any structure, content, and meaning.

This means that we can read any content or meaning into the states and decisions of the shortest path model, to thereby assign them any role or function we want them to have. The limit is ... in our imagination.

Thus, in this discussion, the terms "shortest path model" and "shortest path problem" are *metaphors*. They can — and usually do — represent situations that have nothing to do with "real" shortest path models and problems.

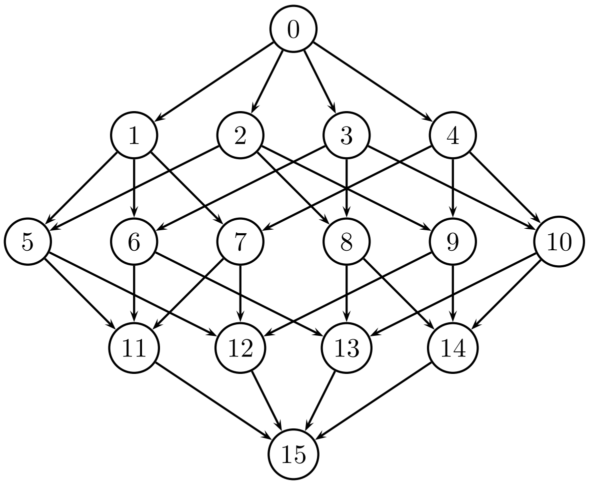


Figure 10.11: Mystery shortest path problem

10.7.2 Example

Consider the shortest path model shown in Figure 10.11. To avoid cluttering the figure, the lengths of the arcs are not displayed. Regard them as an arbitrary collection of non-negative numbers, and assume that the objective function is as shown in (10.256), with $\oplus = +$, $\sigma = 0$ and $\hat{s} = 15$.

The question is then this: what is the mystery problem behind this model?

It turns out that this model represents an extremely large class of problems, in fact, all the combinatorial problems whose decision space consists of all the (24) *permutations* of four distinct objects and whose objective functions have certain separability properties. The 16 nodes comprise the *state space of such problems.*

To highlight this fact, the same state transition model, is depicted in Figure 10.12. In this case, however, the nodes have more informative labels.

So, for example, the path $(0, 3, 34, 134, 1234)$ represents the permutation $(3, 4, 1, 2)$ of $(1, 2, 3, 4)$ whereas the path $(0, 2, 23, 123, 1234)$ represents the permutation $(2, 3, 1, 4)$ of $(1, 2, 3, 4)$. There are $4! = 24$ such permutations (paths) and they are represented by $2^4 = 16$ states.

Perhaps the most famous of these problems is the *Assignment problem.* Its generic form is as follows:

$$\min_{x_1,\dots,x_k} \sum_{j=1}^{k} t(x_j, j) \tag{10.269}$$

$$\{x_1, \dots, x_k\} = \mathcal{K} := \{1, 2, 3, \dots, k\} \tag{10.270}$$

The story behind it is the following: Given k jobs and k machines, the idea is to assign machines to jobs — one machine per job. The time it takes machine m to process job j is given by $t(m, j)$. The goal is to find a machine

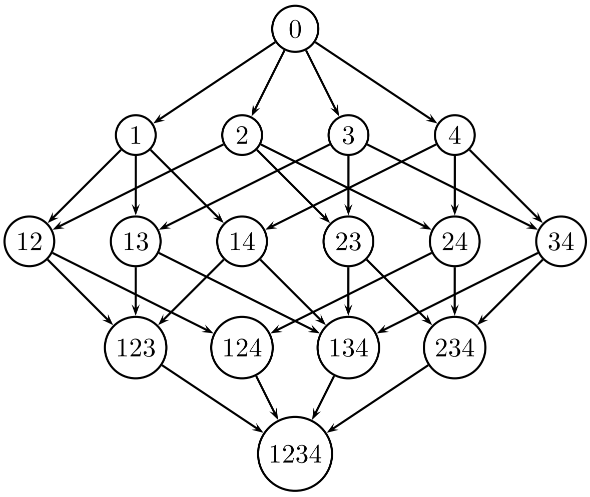


Figure 10.12: Mystery shortest path problem

assignment that minimizes the total jobs processing time. Note that in the above formulation the decision variable is defined as follows:

$$x_j = \text{ machine assigned to process job } j \in \{1, 2, \ldots, k\}$$

A feasible solution $(x_1, \ldots, x_k)$ is then a permutation of the k machines. For example, for $k = 4$, the solution $\mathbf{x} = (2, 3, 4, 1)$ means: assign machine 2 to job 2, machine 3 to job 2, machine 4 to job 3 and machine 1 to job 4.

To determine the set of possible values of x_j, the values of $x_1, x_2, \ldots, x_{j-1})$, or more precisely the set consisting of these values, must be known. This will prevent assigning the same machine to more than one job.

So let the state variable s be a subset of $\mathcal{K}$, representing the machines yet to be assigned. For instance, $s = \{2, 4, 7, 9\}$ represents a situation where 4 machines, namely 1,4,7, and 9, are yet to be assigned. This means that the state space S is the power set of $\mathcal{K}$, namely it is the set of all the subsets of $\mathcal{K}$, recalling that there are 2^k such subsets.

The set of feasible decisions pertaining to state s can therefore be set as follows:

$$D(s) = s \tag{10.271}$$

This means that $D(\varnothing) = \varnothing$ and therefore $\hat{s} = \varnothing$ is a terminal state, where $\varnothing$ denotes the empty set. Clearly, $\sigma = \mathcal{K} = \{1, 2, \ldots, k\}$ is the initial state. Note that the transition function and the reward functions are as follows:

$$T(s, x) = s \setminus \{x\} \ , \ x \in s \tag{10.272}$$

$$w(s, x) = t(x, k - |s| + 1) \ , \ 0 < |s| \leq k \tag{10.273}$$

In short, the *Assignment Problem* can be viewed as a sequential decision problem that seeks the shortest path from the initial state $\sigma = \mathcal{K}$ to the terminal state $\hat{s} = \varnothing$.

The dynamic programming functional equation associated with this model is therefore as follows:

$$f(s) = \min_{x \in s} \{t(x, k - |s| + 1) + f(s \setminus \{x\})\} \ , \ s \subseteq \mathcal{K}, s \neq \varnothing \qquad (10.274)$$

with $f(\varnothing) = 0$.

The difference between the outright shortest path formulation and the more detailed sequential decision formulation of this problem comes down to a mere relabeling of the state and decision variables. This relabeling is however edifying in that it infuses these objects with greater meaning. However, from a purely mathematical point of view these two formulations are equivalent

Also note that many other generic problems can be represented by the same shortest path model. They all share the same state transition model that essentially enumerates, in a dynamic programming fashion, all permutations of k distinct items. For instance, the above problem is equivalent to the problem discussed in *Section 8.3.2*. □

The conclusion to be drawn from all this is that, one of the implications of the discussion in *Section 3.2* is that *any combinatorial optimization problem can be formulated as a shortest path problem.* Of course, this does not mean that this model would always be the right modeling framework for specific combinatorial optimization problems. Much less does it imply that dynamic programming provides an easy remedy for the solution of all combinatorial optimization problems.

For, as the discussion in *Chapter 8* has shown, the exceedingly large state space required by dynamic programming models of certain combinatorial problems triggers the *Curse of Dimensionality*. Add to this that in the case of the *Assignment Problem* we have $|S| = 2^k$ and no more need be said!

Still, this general conclusion about the shortest path model and the sequential decision model is important, highlighting as it does, that despite their lean structures, the two models provide potent, universal, modeling paradigms.

10.7.3 Multistage vs Sequential Models

It is often the case that the same problem admits of various dynamic programming formulations. The implication is that the choice between the multistage decision model discussed in *Chapters 3* and *4* and the sequential decision model examined in this section is essentially a matter of convenience. There are no strict rules and regulations in this matter. The bottom line is that one chooses the model that best suits one's needs.

In particular, one model would be best for *teaching purposes*, another model for *computational purposes*.

The following example illustrates how the same problem can be given two distinct dynamic programming formulations: a multistage formulation and a sequential decision formulation.

10.7.4 Example

Consider again the classic (unbounded) knapsack problem that we examined in *Chapter 5,* namely

$$\max_{x_1,\dots,x_k} \sum_{n=1}^{k} c_n x_n \tag{10.275}$$

$$\sum_{n=1}^{k} a_n x_n \leq b \tag{10.276}$$

$$x_n \in \{0,1,2,3,\dots\}\ ,\ n=1,2,3,\dots,k \tag{10.277}$$

where $\{a_j\}$ and b are positive integers and $\{c_j\}$ are positive real numbers.

If the multistage model is used, the formulation of the modified problems would be as follows:

$$f_n(s) := \max_{x_m,\dots,x_k} \sum_{m=n}^{k} c_m x_m\ , \qquad n=1,2,\dots,k;\ \ s=0,1,2,\dots,b \tag{10.278}$$

$$\sum_{m=n}^{k} a_m x_m \leq s \tag{10.279}$$

$$x_m \in \{0,1,2,3,\dots\}\ ,\ m=n,n+1,\dots,k \tag{10.280}$$

So, the resulting dynamic programming equation would be

$$f_n(s) := \max_{\substack{x\ Integer\\ 0\leq xa_n\leq s}} \{xc_n + f_{n+1}(s-xa_n)\}\ ,\ n=1,2,\dots,k \tag{10.281}$$

for $s=0,1,2,\dots,b$ with $f_{n+1}(s)=0, \forall s$.

That is, in this case

$$S = \{0,1,2,3,\dots,b\} \tag{10.282}$$

$$D(n,s) = \left\{0,1,2,\dots,\left\lfloor \frac{s}{a_n} \right\rfloor\right\} \tag{10.283}$$

$$T(n,s,x) = s - xa_n \tag{10.284}$$

and the decision variables are interpreted as follows:

$x_n :=$ *number of items of type n selected*, $n=1,2,\dots,k$

The task set by this problem can be described by the following narrative. Given a number of piles (k), such that each contains (infinitely) many identical items of a certain type, proceed from the first pile to the last, selecting items from the piles so as to maximize the total value of the items selected, subject to an upper bound (b) on the total weight of the items selected.

If a sequential decision model is used, the problem would be formulated so that the modified problems would be defined as follows:

$$f(s) := \max_{\substack{M \\ x_1,\dots,x_M}} \sum_{n=1}^{M} c_{x_n} \ , \qquad s = 0, 1, 2, \dots, b \tag{10.285}$$

$$\sum_{n=1}^{M} a_{x_n} \leq s \tag{10.286}$$

$$x_n \in \{1, 2, \dots, k\} \tag{10.287}$$

observing that the decision variables are interpreted as follows:

M : = *Total number of items selected*

x_n : = *type of the n-th item selected,* $n = 1, 2, \dots, M$.

For instance, the sequence of decisions $\mathbf{x} = (4, 4, 1, 3, 4, 2, 1, 3, 2, 1)$ indicates that the first item selected is of type 4, the second is of type 4, the third is of type 1, the fourth is of type 3, and so on.

We would then set

$$S = \{0, 1, 2, 3, \dots, b\} \tag{10.288}$$

$$D(s) = \{j : a_j \leq s, j = 1, 2, \dots, k\} \tag{10.289}$$

$$T(s, x) = s - a_x \tag{10.290}$$

$$\sigma = b \tag{10.291}$$

Observe that the set of terminal states is

$$S'' := \{0, 1, 2, \dots, a^* - 1\} \tag{10.292}$$

where $a^* := \min\{a_1, \dots, a_k\}$.

The dynamic programming functional equation for this model would then be as follows:

$$f(s) = \max_{\substack{x \in \{1,\dots,k\} \\ a_x \leq s}} \{c_x + f(s - a_x)\} \ , \ s = a^*, \dots, b \tag{10.293}$$

with $f(s) = 0, s < a^*$.

So which model is "better"?

The answer of course is: . . . it all depends! □

The evidence seems to be that a preference is shown, especially by dynamic programming novices, for the sequential decision model. This, no doubt, is due to the fact that the sequential model directly lends itself to a graphic representation of the problem under consideration. It is important to note therefore that multistage models can equally be depicted graphically. All one needs to do is to to this end is to display the multistage decision model on a "grid" using one axes to represent the stages and the other to represent the states. This would be particularly appropriate for cases where the states are scalars and the state space is finite.

10.8 The Art of Dynamic Programming Modeling

As demonstrated by our discussion thus far, dynamic programming enables considerable flexibility in the formulation of models. I examine this aspect of dynamic programming from various angles in the remaining part of the book. Still, in anticipation of what is in store, I am going to conclude this chapter with a terse illustration of dynamic programming's powers as a modeling tool by showing the alternatives it offers for the formulation of the following two variants of the conventional knapsack problem:

$=$ Constrained Knapsack Problem	$\geq$ Constrained Knapsack Problem
$\max\limits_{(x_1,\dots,x_k)} \sum\limits_{j=1}^{k} c_j x_j$	$\min\limits_{(x_1,\dots,x_k)} \sum\limits_{j=1}^{k} c_j x_j$
$\sum\limits_{j=1}^{k} a_j x_j = b$	$\sum\limits_{j=1}^{k} a_j x_j \geq b$
$\mathbf{x} \geq 0$	$\mathbf{x} \geq 0$

where as accepted, $\{a_j\}$ and b are positive integers and $\{c_j\}$ are positive real numbers.

Just in case, note that:

· In the $=$ constrained version, the resource constraint is an **equality** constraint.

· In the $\geq$ constrained version, the resource constraint is a **greater than or equal to** constraint, and **opt = min**.

Over the years I have seen numerous undergraduate and graduate students struggle to work out dynamic programming formulations for these problems even though they managed quite easily to formulate correct dynamic programming models for the conventional version of the problem.

Why modeling these two versions proves more challenging than modeling the conventional version of the problem is an interesting question!

In the next chapter I address questions such as this in my discussion on modeling in dynamic programming style.

All I want to illustrate at this stage is that even slightly different versions of the same problem can give rise to significantly different mathematical formulations. And that dynamic programming has the capabilities to meet the specific demands of different versions of the same problem by supplying the appropriate means of formulation.

10.8.1 Example

Consider the following two slightly modified versions of the standard (unbounded) knapsack problem examined above.

- *Version 2*
  Items can be selected from any pile, provided that if an item is selected from pile m, at least one item must be selected from pile $1, 2, \ldots, m-1$.
- *Version 3*
  Items can be selected from at most m piles, $1 \leq m \leq k$, recalling that k denotes the number of available piles.

Dynamic programming offers a number of formulations for these two problems. Out of these I am suggesting the following, where *Version 1* gives the formulation of the standard version of the problem.

Version 1	*Version 2*	*Version 3*
$\max\limits_{(x_1,\ldots,x_k)} \sum_{j=1}^{k} c_j x_j$	$\max\limits_{\substack{1\leq m\leq k \\ (x_1,\ldots,x_m)}} \sum_{j=1}^{m} c_j x_j$	$\max\limits_{\substack{J\subseteq\mathcal{K} \\ \lvert J\rvert\leq m \\ x_j, j\in J}} \sum_{j\in J} c_j x_j$
$\sum_{j=1}^{k} a_j x_j \leq b$	$\sum_{j=1}^{m} a_j x_j \leq b$	$\sum_{j\in J} a_j x_j \leq b$
$x_j \in I, j \in \mathcal{K}$	$x_j \in I^+, j = 1, 2, \ldots, m$	$x_j \in I^+, j \in J$

where $I = \{0, 1, 2, \ldots\}, I^+ = \{1, 2, 3, \ldots\}, \mathcal{K} = \{1, 2, \ldots, k\}$.

Note that in Version 2, m is a decision variable whereas in Version 3 it is a given integer. The set J in Version 3 is a *decision variable* specifying the subset of piles from which items are selected.

I should add that other branches of optimization of course offer other valid formulations for these versions of the knapsack problem. And what is more, I do not contend that the above formulation are superior. Indeed, I suspect that my *Integer Programming* colleagues would argue that the following formulation of *Version 3* is far more "natural" and attractive:

$$\max_{(x_1,\ldots,x_k)} \sum_{j=1}^{k} c_j x_j \tag{10.294}$$

$$\sum_{j=1}^{k} a_j x_j \leq b \tag{10.295}$$

$$\sum_{j=1}^{k} y_j \leq m \tag{10.296}$$

$$x_j \leq \mathcal{M} y_j \ , \ j = \mathcal{K} \tag{10.297}$$

$$y_j \in \{0, 1\} \ , \ j \in \mathcal{K} \tag{10.298}$$

$$x_j \in I, j \in \mathcal{K} \tag{10.299}$$

where $\mathcal{M}$ denotes a very large number.

I would counter though that those who are not *Integer Programming* experts, may not find this formulation so "natural". In fact, they would have to be instructed that:

· The binary decision variable y_j indicates whether items are selected from pile j, that is, ($y_j = 1$ means “yes”; $y_j = 0$ means “no”.

· Constraint (10.298) guarantees that items are selected from no more than m piles.

· The constraint $x_j \leq \mathcal{M} y_j$ ensures that $x_j = 0$ if $y_j = 0$.

· Because $\mathcal{M}$ is very large, this constraint is superfluous if $y_j = 1$.

Of course, my *Integer Programming* colleagues may also argue that this formulation has the advantage of being a “standard” *linear integer programming* formulation and thus easily solved with *linear integer programming* software (provided that k and b are not exceedingly large)...

Be that as it may, my intention was not to compare dynamic programming's capabilities to those of other optimization methods. My intention was to preview my forthcoming illustrations of dynamic programming's versatility by hinting at its handling of different versions of the knapsack problem.

10.9 Summary

With the introduction of the Weak-Markovian Condition, the systematizing of the concept of decomposition scheme, and the formulation of the sequential decision model, the stage is set for a discussion of the larger questions of dynamic programming, foremost of which is problem *modeling* and *formulation.*

My main thrust in exploring this question will be to show that modeling a problem in dynamic programming style boils down to:

· An imaginative phrasing of the problem's components; and

· A clever manipulation of the problem's structural features.

I shall illustrate this point in the next chapter where my focus will be on the modeling of the state variable. Modeling and formulation will also be illustrated in *Chapter 14* where I discuss “forward” dynamic programming models and in *Chapter 15* in my examination of *Dijkstra's Algorithm* for the shortest path problem.

10.10 Bibliographic Notes

My approach to the sufficient conditions that underwrite the validity of dynamic programming's functional equation differs from the prevailing approach.

The conventional view is that the equation's validity is assured by the monotonicity conditions described in *Section 10.1.3,* e.g. Mitten [1964], Nemhauser [1966], Denardo and Mitten [1967], Karp and Held [1967], Yakowitz [1969], Ibaraki [1972], Iwamoto [1992, 1993a, 1993b, 1994a], Maruyama [1997, 1999a, 2003a, 2008], Morin [1982].

In contrast, on my approach the *Markovian property* is the foundational feature that assures the validity of the functional equation of dynamic programming. In this framework the traditional monotonicity condition is a sufficient condition for the Markovian property to hold. I look at this issue more closely in *Chapter 13.*

For a more detailed account on the solution of dynamic programming functional equations stemming from shortest path type problems, I refer the reader to Dreyfus [1969], Lawler [1976], White [1978], and Denardo [1982, 2003].

11

The State

11.1 Introduction

Formulating an optimization problem in dynamic programming style is considered by many to be an *art.* Roughly, what seems to be suggested by this characterization is that when using dynamic programming, one cannot rest assured that a meticulous execution of a given set of moves is certain to produce a sound dynamic programming model and consequently a valid dynamic programming functional equation.

More often than not, particularly in the case of intricate problems, the most crucial steps in the formulation process are those where one needs to decide what role to assign the problem's components in the model. To be precise, where one has to establish what are the state space, decision space, transition function, objective function etc., for the problem concerned; and more importantly, where one has to determine how to express these objects mathematically. In many cases, making these judgments is not a matter of correctly deducing them from antecedent moves. Rather, it is a matter of using to good effect one's imagination, experience, intuition, even one's common sense.

In short, setting out a sound dynamic programming model leading to a valid functional equation depends, in large part, on one's ability to be creative and imaginative as a mathematical *modeler.*

Having said all that, the question obviously arises whether any constructive instruction can be given on the subject of modeling and formulation at all. If personal input indeed has such weight, what concrete advice can be offered in the first place?

However valid such an objection may appear at first, the fact is that a fair amount of guidance can be given on this matter. For one thing, it is possible to bring to the fore those elements of a dynamic programming model that are central in the formulation process. Also, it is possible to *illustrate* how one might approach the modeling aspects of dynamic programming. To this end I concentrate in this chapter on the **state variable.**

My primary aim is to underscore the essential role that the state plays in the formulation of a dynamic programming model. That is, I want to show what course one can pursue in order to determine what the state is in the context of a problem, and to demonstrate how one would proceed to frame it mathematically.

11.2 Preliminary Analysis

Let us begin then by quickly reminding ourselves where we stand. You will recall that in *Chapter 4* I outlined a prototype dynamic programming model for optimization problems that are subsumed by the following general type:

$$\textit{Problem P:}\ p := \operatorname*{opt}_{x\in X}\ q(x)\ ,\ X \subseteq \operatorname*{\times}_{n=1}^{k} X_n \tag{11.1}$$

I showed that, subject to a technical regularity condition, any optimization problem of this type can be stated in terms of a dynamic programming model and I described the essential ingredients comprising such a model.

Methodologically then, I made clear what *kind* of model one would need to construct for a given instance of *Problem P.* However, I did not go into the specifics of what steps would have to be taken in order to bring a given instance of *Problem P* to such a formulation. To avoid discussing things in the abstract, let us examine how this question applies to a given case.

Consider then the following instance of *Problem P*:

$$p := \max_{(x_1,\dots,x_k)} \sum_{n=1}^{k} b_n(x_n) + \left[\sum_{n=1}^{k} a_n(x_n)\right]^2 \tag{11.2}$$

$$\{x_1, x_2, \dots, x_k\} = \{1, 2, \dots, k\} \tag{11.3}$$

$$\sum_{n=1}^{k} c_n(x_n) \leq r\ ,\ r \geq 0 \tag{11.4}$$

where k is a given positive integer, and $\{a_n\}$, $\{b_n\}$ and $\{c_n\}$ are given real-valued functions on $\{1, 2, \dots, k\}$. Formally we can set

$$X_n := \{1, 2, \dots, k\}\ ,\ 1 \leq n \leq k \tag{11.5}$$

and

$$q(x_1, \dots, x_k) := \sum_{n=1}^{k} b_n(x_n) + \left[\sum_{n=1}^{k} a_n(x_n)\right]^2 \tag{11.6}$$

and let X be the subset of $\{1, 2, \dots, k\}^k$ defined by (11.3)-(11.4).

Being clear on the *general profile* of a dynamic programming model, the question is then: how would one go about giving *this specific problem* a dynamic programming formulation?

My thesis is that the pivotal move in this effort is to establish what the *state variable* is and to give it a definite mathematical phrasing. Once the state has been decided upon and its mathematical form worked out, all the other elements of the model seem to fall into place almost automatically. My main thrust in this chapter will be then to outline a way for identifying the state variable and formulating it.

It is important to note, though, that by this I do not suggest that this "search tactic" *is,* or *should* be made, an integral part of every dynamic programming investigation. Obviously, my aim here is to describe an aide, an auxiliary device, on which one would be able to fall back, should such a need arise.

I shall first outline in broad terms the strategy that I propose to follow in this endeavor and shall then proceed to give it a more careful mathematical treatment.

11.2.1 General Outline of Strategy

Recall that the state was characterized in *Chapters 3* and *4* as the "information store" of the multistage decision model. It was described as that particular component in the model that, at each stage of the process, contains information that is required for determining the optimal values of the remaining decisions.

This being so, it only makes sense that if our objective is to track down the state and determine its structure, that we work our way back from a set of conditional problems of the form:

$$p(x_1^*, x_2^*, \dots, x_{n-1}^*) := \max_{(x_n,\dots,x_k)} q(x_1^*, x_2^*, \dots, x_{n-1}^*, x_n, \dots, x_k) \tag{11.7}$$

$$(x_1^*, x_2^*, \dots, x_{n-1}^*, x_n, \dots, x_k) \in X \tag{11.8}$$

where $x_1^*, x_2^*, \dots, x_{n-1}^*$ are *given* values of the first (n-1)st decision variables.

Thus, treating the conditioning sequence $(x_1^*, \dots, x_{n-1}^*)$ as a starting point, we would ask:

> *What information about the sequence $(x_1^*, \dots, x_{n-1}^*)$ is relevant for determining the optimal solutions to the conditional problem specified by (11.7)-(11.8) ?*

Clearly implicit in this question is the suggestion that not all the information contained in the sequence $(x_1^*, \dots, x_{n-1}^*)$ has a bearing on the optimization of the problem induced by it.

Indeed, as we shall progressively see, only certain elements of it are important for this purpose. The basic thesis of the proposed strategy is then that extracting from this sequence that vital kernel of information should give us the keys to the state's makeup.

To be able to treat this question formally, let $\tau(x_1^*, \dots, x_{n-1}^*)$ denote the information about the sequence $(x_1^*, \dots, x_{n-1}^*)$ that is needed in order to solve the problem specified by (11.7)-(11.8). The objective would then be to attempt to decipher the structure of the function τ, by seeking to identify that vital core in $(x_1^*, \dots, x_{n-1}^*)$ that it supplies.

Now, it seems rather obvious that the way to go about it is to seek to associate this information with the source from which it is likely to emanate.

And here, there are only two objects which can constitute this source. These are:

- The solution set X.
- The objective function q.

The aim would be then to attempt to find out in what way, if any, do the solution set and the objective function affect the function τ.

11.2.2 The Impact of the Solution Set on the State

Again, the obvious thing to do to work out an answer to this question is to examine what role might each of the *constraints* imposed on the solution set X have in this matter. The point here is, of course, that the constraints are critical in determining the nature of an optimal solution.

Now, when we talk of constraints in this context we need to distinguish between two types, LOCAL and GLOBAL.

To explain what is meant by this consider the case where the solution set is defined as follows:

$$X = \{(x_1, \dots, x_k) : \sum_{n=1}^{k} x_n \leq r \ , \ 0 \leq x_n \ , \ 1 \leq n \leq k\} \ , \ r \geq 0 \tag{11.9}$$

Here X is subject to two types of constraints, namely

$$0 \leq x_n \ , \ 1 \leq n \leq k \tag{11.10}$$

and

$$\sum_{n=1}^{k} x_n \leq r \tag{11.11}$$

Constraints of the type specified by (11.10) are called *local* whereas constraints of the type specified by (11.11) are termed *global.* A constraint is understood to be local if it affects only *one component* of the entire sequence of decisions $(x_1, \dots, x_k)$.

For example, the requirement "$0 \leq x_2$" is local because it applies only to the second component of x. The same is true of the constraint

$$x_n \in \{0, 1\} \ , \ 1 \leq n \leq k \tag{11.12}$$

which in effect consists of k *independent* local constraints. The following is a list of typical local constraints:

$$a_n \leq x_n \leq b_n \ , \ 1 \leq n \leq k \tag{11.13}$$

$$x_n \in \{n, n+1, \dots, k\} \ , \ 1 \leq n \leq k \tag{11.14}$$

$$x_n \in \{0, 1\} \ , \ 1 \leq n \leq k \tag{11.15}$$

$$x_n \in \{0, 1, 2, 3, \dots\} \ , \ 1 \leq n \leq k \tag{11.16}$$

In contrast, a global constraint involves *two or more* components of the sequence $y \in X$. Thus, (11.11) is a typical global constraint because it imposes a relationship between the components of y.

The following are typical global constraints:

$$\{x_1, \dots, x_k\} = \{1, 2, 3, \dots, k\} \tag{11.17}$$

$$x_1 \leq x_2 \leq x_3 \leq \cdots \leq x_k \tag{11.18}$$

$$\sum_{n=1}^{k} b_n x_n \leq v \tag{11.19}$$

$$x_2 \geq x_7 \tag{11.20}$$

Now, considering these fundamental differences between local and global constraints, one's immediate hunch would be that

> GLOBAL *constraints affect the* VERY STRUCTURE *of the state variable whereas* LOCAL *constraints affect only its* FEASIBLE VALUES.

Hence, by investigating the effect that the global constraint might have on the state variable we should be able to unravel the basic structure of this variable, and by examining that of the local constraint we should be able to pin down the feasible values that the state can take. By "structure" I mean whether the state is a numeric scalar, a numeric vector, a function, a set, and so on.

Let us consider a concrete example. Suppose that the solution set X is subject to the global constraint (11.11). In line with our statement of the conditional problem in (11.7), we would stipulate a feasible sequence of decisions $(x_1, \dots, x_k)$.

Next, we would decompose the global constraint in question into two parts, one corresponding to the sequence $(x_1, \dots, x_{n-1})$, and the other to the sequence $(x_n, \dots, x_k)$, for some arbitrary $1 < n < k$. Our object would be to formulate the constraint imposed on $(x_n, \dots, x_k)$ in terms of the conditioning sequence $(x_1, \dots, x_{n-1})$. And so, we decompose (11.11) as follows:

$$\sum_{j=1}^{n-1} x_j + \sum_{j=n}^{k} x_j \leq r \tag{11.21}$$

or equivalently as follows:

$$\sum_{j=n}^{k} x_j \leq r - \sum_{j=1}^{n-1} x_j \tag{11.22}$$

In either case the constraint imposed on $(x_n, \dots, x_k)$ can be narrowed down to two scalars, namely r and $\sum_{j=1}^{n-1} x_j$. Thus, assuming that the value of r is known, it follows that all that need be known about the sequence $(x_1, \dots, x_{n-1})$ in order to determine whether or not $(x_n, \dots, x_k)$ satisfies (11.21)-(11.22) is the value of the scalar $\sum_{j=1}^{n-1} x_j$. Insofar as the global constraint (11.11) is concerned, $(x_1, \dots, x_{n-1})$ is equivalent to $(x'_1, \dots, x'_{n-1})$ if

$$\sum_{j=1}^{n-1} x'_j = \sum_{j=1}^{n-1} x_j \tag{11.23}$$

regardless of the equivalence (or lack thereof) of the individual elements, x_i and x'_i, $i = 1, 2, \dots, n-1$. The upshot of all this is that the state variable would, in this case, be either:

$$s_n := \sum_{j=1}^{n-1} x_j \ , \ 1 < n \leq k+1 \tag{11.24}$$

where, $s_1 := 0$; or,

$$s_n := r - \sum_{j=1}^{n-1} x_j \ , \ 1 < n \leq k+1 \tag{11.25}$$

where $s_1 = r$.

Note that the right-hand side of (11.22) is uniquely determined by s_n and r, that is, (11.22)-(11.24) yield

$$\sum_{j=n}^{k} x_j \leq r - s_n \tag{11.26}$$

whereas (11.25), together with (11.22), yield

$$\sum_{j=n}^{k} x_j \leq s_n \tag{11.27}$$

In summary then, the state variable induced by the global constraint (11.11) is a scalar whose value is equal to the *sum* of the elements of the conditioning sequence $(x_1, \ldots, x_{n-1})$, or to r minus this sum. The choice between (11.24) and (11.25) is essentially a matter of style and convenience.

Having established that, let us now examine how a typical local constraint would influence the state variable deriving from (11.11). Consider then the local constraints $x_n \geq 0$, $1 \leq n \leq k$ defined in (11.10). The point to note here is that, as things stand, the state variable induced by the global constraint (11.11) is unbounded, that is, $-\infty \leq s_n \leq \infty$, irrespective of whether s_n is determined by (11.24) or by (11.25). The state space would therefore be $S = \mathbb{R}$, where $\mathbb{R}$ denotes the real line.

However, by imposing the local constraints $x_n \geq 0$, $1 \leq n \leq k$, the state variable becomes bounded by 0 and r, namely $0 \leq s_n \leq r$, in which case $S = [0, r]$. In sum, the local constraints delimit the range of feasible values that the state can take, but they have no affect on the basic definition of the state given by (11.24)-(11.25), which was induced entirely by the global constraint (11.11).

As can be gathered from the preceding discussion, each global constraint of *Problem P* induces a component in the state variable of the dynamic programming model representing this problem. Thus, in general, if there are M global constraints, the state variable would have at least M components.

The following example features a typical problem with two global constraints. The state variable of its dynamic programming model thus consists of two components.

11.2.3 Example

Consider the following problem. This is the classic knapsack problem except that in this case an additional global constraint is imposed.

$$w^* := \max_{x_1,\dots,x_k} \sum_{n=1}^{k} c_n x_n \tag{11.28}$$

$$\sum_{n=1}^{k} a_n x_n \leq \alpha \tag{11.29}$$

$$\sum_{n=1}^{k} b_n x_n \leq \beta \tag{11.30}$$

$$x_n \in \{0,1,2,\dots\} \ , \ \forall n \in \mathcal{K} := \{1,2,\dots,k\} \tag{11.31}$$

where all the coefficient are positive integers. Let X denote the set of all the feasible solutions to this problem.

The conventional dynamic programming model for this problem would be based on state variables of the form

$$s_n := \left(\alpha - \sum_{j=1}^{n-1} a_j x_j \ , \ \beta - \sum_{j=1}^{n-1} b_j x_j\right) \ , \ n = 2,\dots,k+1 \tag{11.32}$$

with $s_1 := (\alpha, \beta)$. The transition function would therefore be defined thus

$$T(n,u,v) := (u - a_n x_n \ , \ v - b_n x_n) \ , \ n \in \mathcal{K} := \{1,\dots,k\} \tag{11.33}$$

and the sets of feasible decisions would be defined as follows:

$$D(n,u,v) := \{0,1,\dots,R(n,u,v)\} \ , \ n \in \mathcal{K} := \{1,\dots,k\} \tag{11.34}$$

where

$$R(n,u,v) := \min\left\{\left\lfloor \frac{u}{a_n} \right\rfloor, \left\lfloor \frac{v}{b_n} \right\rfloor\right\} \tag{11.35}$$

Given this, the functional equation for the problem would be of this form:

$$f_n(u,v) = \max_{x_n \in D(n,u,v)} \{c_n x_n + f_{n+1}(u - a_n x_n, v - b_n x_n)\} \tag{11.36}$$

for $n \in \mathcal{K}$, with $f_{k+1}(u,v) := 0, \forall u,v$. The object of interest here is the value of $f_1(\alpha,\beta)$.

This functional equation would have to be solved for all $n \in \mathcal{K}$ and all the feasible values of the pair (u,v). So let $UV(j)$ denotes the set of all the feasible values of the (u,v) pairs at stage j, namely set

$$UV(n) := \{(u,v) : u \in U(n), v \in V(n,u)\} \ , \ n \in \mathcal{K} \tag{11.37}$$

where

$$U(n) := \left\{ \alpha - \sum_{j=1}^{n-1} a_j x_j : x \in X \right\} \ , \ n \in \mathcal{K} \tag{11.38}$$

$$V(n,u) := \left\{ \beta - \sum_{j=1}^{n-1} b_j x_j : \alpha - \sum_{j=1}^{n-1} a_j x_j = u, \ x \in X \right\} \tag{11.39}$$

for $n \in \mathcal{K}, u \in U(n)$. □

11.2.4 The Impact of the Objective Function

Consider the case where the objective function of *Problem P* has the following form:

$$q(x_1,\dots,x_k) := \sum_{n=1}^{k} a_n x_n + \prod_{n=1}^{k} b_n x_n \tag{11.40}$$

where $\{a_n\}$ and $\{b_n\}$ are given numeric constants.

Given a partial feasible solution, say $(x_1^*,\dots,x_{n-1}^*)$, the conditional problem can be rewritten as follows:

$$p(x_1^*,\dots,x_{n-1}^*) := \underset{(x_n,\dots,x_k)}{\text{opt}} \left\{ \sum_{n=1}^{k} a_n x_n + \prod_{n=1}^{k} b_n x_n \right\} \tag{11.41}$$

$$x_j = x_j^* \ , \ 1 \le j \le n-1 \tag{11.42}$$

$$(x_1^*,\dots,x_{n-1}^*, x_n,\dots,x_k) \in X \tag{11.43}$$

Because the sequence $(x_1^*,\dots,x_{n-1}^*)$ is treated as a known object, we incorporate (11.42) in the objective function and rewrite (11.41) as follows:

$$p(x_1^*,\dots,x_{n-1}^*) = \underset{(x_n,\dots,x_k)}{\text{opt}} \left\{ \left(\sum_{j=1}^{n-1} a_j x_j^* + \sum_{j=n}^{k} a_j x_j \right) + \left(\prod_{j=1}^{n-1} b_j x_j^* \right) \times \left(\prod_{j=n}^{k} b_j x_j \right) \right\} \tag{11.44}$$

subject to (11.43).

As before, the question is: what information about the sequence $(x_1^*,\dots,$ $x_{n-1}^*)$ is required for solving the conditional problem defined by (11.43)-(11.44) ?

And the answer is: since the decisions $x_1^*,\dots,x_{n-1}^*$ are viewed as known objects, it only makes sense to define

$$c_n := \sum_{j=1}^{n-1} a_j x_j^* \ , \ n = 2,3,\dots,k+1 \tag{11.45}$$

and

$$d_n := \prod_{j=1}^{n-1} b_j x_j^* \ , \ n = 2, 3, \dots, k+1 \tag{11.46}$$

with $c_1 = 0$ and $d_1 = 1$, thus regarding c_n and d_n as given parameters. In this case (11.44) will be rewritten as follows:

$$p(x_1^*, \dots, x_{n-1}^*) := \underset{(x_n,\dots,x_k)}{\text{opt}} \left\{ c_n + \left(\sum_{j=n}^{k} a_j x_j \right) + d_n \left(\prod_{j=n}^{k} b_j x_j \right) \right\} \tag{11.47}$$

where c_n and d_n are the known constants defined above.

Clearly then, whatever effect the conditioning sequence $(x_1^*, \dots, x_{n-1}^*)$ may have on the set of optimal solutions to (11.47), to figure out this set only the scalars c_n and d_n defined by (11.45)-(11.46) are required.

Thus, following our previous line of reasoning, our first hunch would be to conclude that the pair (c_n, d_n) would be stored in the state variable. On closer scrutiny, however, it turns out that the constant c_n is not a factor in determining the set of optimal solutions to (11.47). This is so because (11.47) is equivalent to

$$\underset{(x_n,\dots,x_k)}{\text{opt}} \left\{ \left(\sum_{j=n}^{k} a_j x_j \right) + \left(d_n \prod_{j=n}^{k} b_j x_j \right) \right\} \tag{11.48}$$

in that the respective problems have the same set of optimal solutions.

This means that the only relevant information about $(x_1^*, \dots, x_{n-1}^*)$ that is required for obtaining the optimal solutions of (11.47) is the value of the scalar d_n defined by (11.46). And therefore the conclusion is that d_n needs to be incorporated in the state variable s_n.

It is important to note, though, that not all objective functions leave an imprint on the state variable. For instance, *separable additive* objective functions of the form

$$q(x_1, \dots, x_k) := \sum_{n=1}^{k} q_n(x_n) \tag{11.49}$$

have no effect on it whatsoever. Needless to say, one would not seek to investigate an objective function of this type with a view to track down the state.

Having described the main outlines of the approach that I propose for tracing the state and its structure, I shall now give it a more careful mathematical treatment.

11.3 Mathematically Speaking

Imagine that you have to make a sequence of k decisions, call it $\mathbf{x} = (x_1, \dots, x_k)$ and that you have already determined the first $n-1$ decisions. This means that you still have to make $k-n+1$ decisions. The notation required for this purpose is the following:

$X \subseteq \underset{j=1}{\overset{k}{\times}} X_n$ denotes the set of feasible solutions.

$\mathbf{x}$ denotes a complete sequence of decisions.

y denotes the sequence consisting of the *first* $n-1$ elements of $\mathbf{x}$.

z denotes the sequence consisting of the *last* $k-n+1$ elements of $\mathbf{x}$.

$Y(n)$ denotes the set of all feasible values of $(x_1, \dots, x_n)$.

$Z(y)$ denotes the set of all feasible continuations of y.

$\mathcal{K} = \{1, 2, \dots, k\}$

The picture then is this:

$$\mathbf{x} = \left(\overbrace{x_1, x_2, \dots, x_{n-1}}^{y} \; , \; \overbrace{x_n, x_{n+1} \dots, x_{k-1}, x_k}^{z} \right)$$

In this framework n is a parameter, taking values in $\mathcal{K}$. We shall use other indices such as j for this purpose.

Formally, Y_j is the projection of X on $X_1 \times \dots \times X_j$, that is

$$Y_0 := \{\varnothing\} \tag{11.50}$$

and

$$Y_j := \{(x_1, \dots, x_j) : (x_1, \dots, x_k) \in X\} \; , \; 1 \le j \le k \tag{11.51}$$

where $\varnothing$ denotes the empty sequence. Similarly, let

$$Z(y) := \{z : (y, z) \in X\} \; , \; , y \in Y_j, 0 \le j < k \tag{11.52}$$

observing that by construction,

$$Z(\varnothing) = Y_k = X \tag{11.53}$$

Our objective is to determine a framework for the identification of a proper state variable, s_n, induced by the sequence $y = (x_1, \dots, x_{n-1})$.

With these ingredients in hand, we can now frame the following family of conditional problems:

Definition 11.3.1 *Problem $C(y), y \in Y_n, 0 \leq n \leq k$:*

$$p(y) := \underset{z \in Z(y)}{\text{opt}} \; q(y,z) \tag{11.54}$$

We shall refer to Problem $C(y)$ as the CONDITIONAL PROBLEM AT *y. Let $Z^*(y)$ denote the set of optimal solutions to Problem $C(y)$. Note that, by construction, Problem P is identical to Problem $C(\varnothing)$.*

We want to identify that core of information about the sequence $y = (x_1, \ldots, x_{n-1})$ that is required for determining the optimal solutions to *Problem $C(y)$*.

To give this idea a formal statement, let us express the relation between y and the state deriving from it as follows:

$$s = \tau(y) \tag{11.55}$$

where τ is assumed to be a function on

$$Y := \bigcup_{n=0}^{k} Y_n \tag{11.56}$$

with values in some set whose elements are called *states*. Set

$$S := \{\tau(y) : y \in Y\} \tag{11.57}$$

and

$$S_n := \{\tau(y) : y \in Y_{n-1}\} \ , \ 1 \leq n \leq k+1 \tag{11.58}$$

One way of looking at τ is to see it as a function that gleans from y all the information required for determining the optimal solutions to *Problem $C(y)$*; another is to see it as a function that rids y of all the information that is irrelevant for this purpose. Whatever the interpretation, $s = \tau(y)$ is understood to be the *state* induced by a partial solution $y \in Y$ to *Problem P*.

The aim is then to decipher the makeup of τ. This will be done from the standpoint that the following relation should hold:

$$\left[y, y' \in Y_n, \tau(y) = \tau(y')\right] \Longrightarrow \left[Z(y) = Z(y') \text{ and } Z^*(y) = Z^*(y')\right] \tag{11.59}$$

In words, if two partial solutions $y, y' \in Y_n$ generate the same state, namely, $\tau(y) = \tau(y')$, then *Problem $C(y)$* and *Problem $C(y')$* must be *equivalent* in that both have the same set of *feasible* solutions, namely $Z(y) = Z(y')$, and the same set of *optimal* solutions, namely $Z^*(y) = Z^*(y')$.

The dual role of the function τ, implied by (11.59), suggests that it be viewed as having the following basic structure:

$$\tau(y) = (\tau_f(y), \tau_o(y)) \tag{11.60}$$

where $\tau_f(y)$ reflects the impact of the decision set X and $\tau_o(y)$ that of the objective function.

Thus, τ_f comprises that element of of the state which ensures that $Z(y) = Z(y')$; and τ_o complements τ_f by consisting of that element of the state which ensures that $Z^*(y) = Z^*(y')$. Obviously, the subscripts f and o associated with τ indicate **f**easibility and **o**ptimality, respectively.

The idea would then be to first seek to establish the makeup of τ_f and then that of τ_o. To this end we would seek to uncover the information about y that is needed to figure out the set of feasible solutions to *Problem C(y),* which will be stored in $\tau_f(y)$, and then the information needed in determining the set of optimal solutions to this problem, which will be stored in $\tau_o(y)$.

11.3.1 The Structure of τ_f

Let us begin then by focusing on the first part of (11.59), namely on

$$[y, y' \in Y_n, \tau_f(y) = \tau_f(y')] \Longrightarrow Z(y) = Z(y') \tag{11.61}$$

For our purposes it will be more instructive to restate this condition as follows:

$$[y, y' \in Y_n, \tau_f(y) = \tau_f(y')] \Longrightarrow [(y, z) \in X, z \in Z(y) \Longleftrightarrow (y', z) \in X, z \in Z(y')] \tag{11.62}$$

This rephrasing very naturally leads us onto the track of seeking to discover the structure of τ_f by decomposing the global constraints imposed on the elements of X.

As before, each global constraint is decomposed into two components which correspond to the decomposition scheme $x = (y, z)$, $y \in Y_n$, $z \in Z(y)$. Again, the guiding idea is to express the feasible values of $z = (x_n, \dots, x_k)$ in terms of the conditioning sequence $y = (x_1, \dots, x_{n-1})$.

To illustrate, consider the case where the solution set X is of the following form:

$$X = \{(x_1, \dots, x_k) : \{x_1, \dots, x_k\} = \{1, 2, 3, \dots, k\}\} \tag{11.63}$$

Note that by construction X is the set of all the *permutations* of the list $(1, 2, \dots, k)$. Since each feasible sequence $(x_1, \dots, x_k) \in X$ consists of k distinct elements, it follows that

$$\{x_1, \dots, x_{n-1}\} \bigcup \{x_n, \dots, x_k\} = \{1, 2, \dots, k\} \tag{11.64}$$

for all $(x_1, \dots, x_k) \in X$ and $1 < n \leq k$. Therefore,

$$\{x_n, \dots, x_k\} = \{1, 2, \dots, k\} \backslash \{x_1, \dots, x_{n-1}\} \tag{11.65}$$

where $A \backslash B := \{a : a \in A, a \notin B\}$. Having established this we can now define

$$\tau_f(x_1, \dots, x_{n-1}) := \{x_1, \dots, x_{n-1}\} \tag{11.66}$$

Clearly then, for $y = (x_1, \ldots, x_{n-1})$ we have

$$Z(y) = \{(x_n, \ldots, x_k) : \{x_n, \ldots, x_k\} = \{1, 2, 3, \ldots, k\} \setminus \tau_f(y)\}\} \tag{11.67}$$

A quick inspection reveals that τ_f indeed satisfies the condition stipulated by (11.61). The implication is therefore that the state variable would be formulated thus:

$$s_n := \tau_f(x_1, \ldots, x_{n-1}) = \{x_1, \ldots, x_{n-1}\} \tag{11.68}$$

with s_1 being the empty set. Or, alternatively, as follows:

$$s_n = \tau_f(x_1, \ldots, x_{n-1}) := \{1, 2, \ldots, k\} \setminus \{x_1, \ldots, x_{n-1}\} \tag{11.69}$$

in which case we would set $s_1 = \{1, 2, \ldots, k\}$.

Of course, the ultimate test to verify that τ_f is properly defined would involve constructing the sets $\{D(n, s)\}$ of the dynamic programming model with the view to ascertain that the set of feasible solutions generated by these sets is equal to X.

That is, one would seek to ensure that for each pair (n, s) such that $1 \leq n \leq k$ and $s = \tau_f(y)$, $y \in Y_{n-1}$, the set $D(n, s)$ indeed consists of all the feasible values of x_n given $y = (x_1, \ldots, x_{n-1})$. Let us examine then how this is done for the solution set X specified in (11.63), and the state defined by (11.68).

The first step is to express a typical element of the set $Z(y)$ in terms of $\tau_f(y)$. Thus, we interpret (11.67) as follows: $z = (x_n, \ldots, x_k) \in Z(y)$ can be any sequence containing $k - n + 1$ distinct elements of $\{1, 2, \ldots, k\}$, provided that none of these elements is an element of the set $\tau_f(y)$. As it is clear that x_n can be such an element of $\{1, 2, \ldots, k\}$, we define

$$D(s) := \{1, 2, \ldots, k\} \setminus s \ , \ s \in S := \{\tau_f(y) : y \in Y\} \tag{11.70}$$

Observe that in this example the state space S is the power set of $\{1, 2, \ldots, k\}$, namely the set comprising all the subsets of $\{1, 2, \ldots, k\}$.

This clear, let us see how the transition function T pertaining to the state variable deriving from τ_f can now be easily constructed. To do this note that (11.66) entails that

$$\begin{aligned} \tau_f(x_1, \ldots, x_{n-1}, x_n) &= \{x_1, \ldots, x_{n-1}, x_n) && (11.71) \\ &= \{x_1, \ldots, x_{n-1}\} \cup \{x_n\} \\ &= \tau_f(x_1, \ldots, x_{n-1}) \cup \{x_n\} && (11.72) \end{aligned}$$

Therefore,

$$s_{n+1} = s_n \cup \{x_n\} \tag{11.73}$$

so that

$$T(s, x) := s \cup \{x\} \ , \ s \in S, x \in D(s) \tag{11.74}$$

Note that both D and T are stationary, that is, both are independent of the stage variable n.

Let us now check whether the set of sequences $\{(x_1,\dots,x_k)\}$ induced by τ_f, D and T is indeed equal to the solution set X defined by (11.63). To this end let $(x_1,\dots,x_k)$ be any sequence such that $x_n \in D(s_n)$, where s_1 is the empty set and $s_{n+1} = T(s_n, x_n)$. We need to show that $(x_1,\dots,x_k) \in X$. This can be done quite easily by induction, that is , by showing that for each m, $1 \leq n \leq k$, the sequence $(x_1,\dots,x_n)$ consists of m distinct elements of $\{1,2,\dots,k\}$. Conversely, let $(x_1,\dots,x_k)$ be any element of the set X. Here we have to show that $x_n \in D(s_n)$ for all n, $1 \leq n \leq k$. Observe then that this follows immediately from the definitions of X, $D(s)$ and $T(s,x)$.

This exercise confirms the thesis that once the makeup of the state variable has been worked out, the structures of the model's remaining components can be easily determined.

After establishing the structure of the state variable through the analysis of the global constraints, one would seek to work out the range of feasible values that the state can take by examining the local constraints.

11.3.2 Example

Consider the case where the set of feasible solutions to *Problem P* is specified by the following constraints:

$$\sum_{n=1}^{k} a_n x_n \leq r \tag{11.75}$$

$$x_n \geq 0 \ , \ 1 \leq n \leq k \tag{11.76}$$

where r and $\{a_n\}$ are positive integers.

An analysis of the global constraint would yield a state of the form

$$s_n = \tau_f(x_1,\dots,x_{n-1}) := r - \sum_{j=1}^{n-1} a_j x_j \tag{11.77}$$

whose transition functions is defined thus:

$$T(n,s,x) := s - a_n x \tag{11.78}$$

Now, since the local constraints require the decision variables to be non-negative, we would set

$$D(n,s) := \left[0 \ , \ \frac{s}{a_n}\right] \tag{11.79}$$

This in turn yields the following state space

$$S = [0,r] \tag{11.80}$$

In contrast, suppose that the problem would additionally be subject to the following local constraints

$$x_n \in \{0, 1, 2, \dots, r\} \tag{11.81}$$

This will not bring about any change in the definitions of τ_f, T, and therefore s_n, but it will yield

$$D(n, s) := \left\{0, 1, 2, \dots, \left\lfloor \frac{s}{a_n} \right\rfloor \right\} \tag{11.82}$$

and

$$S = \{0, 1, 2, \dots, r\} \tag{11.83}$$

The point to note here is that in (11.79)-(11.80) we deal with *continuous* state and decision spaces, whereas in (11.82)-(11.83) we deal with *discrete* ones.

I call attention to this fact because, this is precisely the factor that determines the solution of the resulting dynamic programming functional equation. In other words, the questions whether the equation will be tractable at all, whether its solution will prove challenging or not, and what approach will be needed in order to solve it will hang on the crucial matter of whether the decision sets $\{D(n, s)\}$ are discrete or continuous. □

As a final note, to bear out the claim that, insofar as the optimization of *Problem* $C(y)$ is concerned, the state can be viewed as a distilled version of the sequence y, let us examine the state space induced by the solution set X defined by (11.63), namely the set

$$X = \{(x_1, \dots, x_k) : \{x_1, \dots, x_k\} = \{1, 2, \dots, k\}\} \tag{11.84}$$

Recall that our analysis revealed that in this case we can set

$$s_n = \tau_f(x_1, \dots, x_{n-1}) = \{x_1, \dots, x_{n-1}\} \tag{11.85}$$

This means that s_n is a set containing $(n - 1)$ distinct elements of $\{1, 2, \dots, k\}$. Hence, the set

$$S_n = \{\tau(x_1, \dots, x_{n-1}) : (x_1, \dots, x_{n-1}, x_n, \dots, x_k) \in X\} \tag{11.86}$$

has

$$|S_n| = \binom{k}{n-1} := \frac{k!}{(n-1)!(k-(n-1))!} \tag{11.87}$$

elements.

In comparison, the set Y_{n-1} which contains all the conditioning sequences that generate the elements of S_n consists of sequences each of which contains $n-1$ distinct elements of $\{1, 2, \dots, k\}$. Hence, Y_{n-1} has

$$\left|Y_{n-1}\right| = \binom{k}{n-1} \times (n-1)! \tag{11.88}$$

elements.

That is, a single value of s_n represents $(n-1)!$ distinct elements of Y_{n-1}. For example, consider the case where $k = 5$, $n = 4$ and $s_n = \{1, 2, 3\}$. This state is produced by six conditioning sequences, namely

$$y^{(1)} = (1,2,3);\; y^{(2)} = (1,3,2);\; y^{(3)} = (2,1,3)$$

$$y^{(4)} = (2,3,1);\; y^{(5)} = (3,1,2);\; y^{(6)} = (3,2,1)$$

This is so because the *order* in which the elements of y are arranged is immaterial for determining the feasible values of the remaining decisions. Clearly, the number of distinct states in this case is significantly smaller than the number of feasible partial solutions. The conclusion therefore is that τ_f would be fundamentally more effective in this case. Nevertheless, for large k the state space would still be extremely large. For example, for $k = 30$ and $n = 15$ we have

$$\left|S_{15}\right| = \binom{30}{15} = 155,117,520 \tag{11.89}$$

You will recall that this fact was described in *Chapter 8* as the trigger of the *Curse of Dimensionality.*

11.3.3 The Structure of τ_o

To go back then to where we left off, recall that, $s = \tau(y)$ is perceived as the state induced by a partial solution $y \in Y$ to *Problem P.* Thus, the relationship between $y \in Y$ and the state that it generates is expressed as follows:

$$s = \tau(y) = (\tau_f(y), \tau_o(y)), y \in Y \tag{11.90}$$

In keeping with this understanding of the state, in the previous section the objective was to identify the structure of τ_f. In this section it is to establish the structure of τ_o. Once this is accomplished we will have the complete structure of the state before us.

Our starting point is then that τ_f has been worked out and that the following relation holds:

$$\big(y, y' \in Y_n, \tau_o(y) = \tau_o(y')\big) \Longrightarrow Z^*(y) = Z^*(y') \tag{11.91}$$

where $Z^*(y)$ denotes the set of optimal solutions to *Problem C*(y).

Thus, our objective is to identify that element in the function τ_o that induces the above *equivalence relation* on Y, which entails that the sets of optimal solutions to *Problem* $C(y)$ and *Problem* $C(y')$ are equal if $\tau_o(y) = \tau_o(y')$.

The line that we pursue here is similar to the one pursued above, except that now we ask: What information about the conditioning sequence y is needed in order to determine the set of *optimal solutions* to *Problem* $C(y)$? Since this question is asked in a frame of mind that τ_f has already been determined via an investigation of the solution set X, the answer clearly should be worked out via an analysis of the objective function of *Problem* $C(y)$. And so, we now decompose the objective function into two elements, one affiliated with the partial solutions $y \in Y$ and one with their complement $z \in Z(y)$.

To be more specific, let us consider the case where the objective function is of the following form:

$$q(x_1, \dots, x_k) = \sum_{n=1}^{k} a_n(x_n) \tag{11.92}$$

where a_n is a real-valued function on X_n.

Then the decomposition would simply mean rewriting (11.92) as follows:

$$q(x_1, \dots, x_k) = \sum_{j=1}^{n-1} a_j(x_j) + \sum_{j=n}^{k} a_j(x_j) \tag{11.93}$$

$$= c + \sum_{j=n}^{k} a_j(x_j) \ , \qquad c := \sum_{j=1}^{n-1} a_j(x_j) \tag{11.94}$$

The conditional problem at $y = (x'_1, \dots, x'_{n-1})$ would therefore take the following form:

$$\begin{aligned} p(x'_1, \dots, x'_{n-1}) :&= \underset{(x_n,\dots,x_k)}{\text{opt}} \ q(x'_1, \dots, x'_{n-1}, x_n, \dots, x_k) \\ &= \underset{(x_n,\dots,x_k)}{\text{opt}} \left\{ c + \sum_{j=n}^{k} a_j(x_j) \right\}, \quad c = \sum_{j=1}^{n-1} x'_j \end{aligned} \tag{11.95}$$

subject to $(x_n, \dots, x_k) \in Z(x'_1, \dots, x'_{n-1})$.

Now, as c is independent of $(x_n, \dots, x_k)$, it follows from (11.95) that

$$p(x'_1, \dots, x'_m) = c + \underset{(x_n,\dots,x_k)}{\text{opt}} \sum_{j=n}^{k} a_j(x_j) \tag{11.96}$$

thereby rendering $Z^*(x'_1, \dots, x'_{n-1})$ independent of c. Formally then, we can set $\tau_o(y) = \varnothing$, $\forall y \in Y$, where $\varnothing$ denotes the empty sequence.

The conclusion to be drawn then is that in this case the structure of the state variable will be given entirely by τ_f so that its formal phrasing will be

$$s_n = \tau(x_1, \dots, x_{n-1}) = \tau_f(x_1, \dots, x_{n-1}) \tag{11.97}$$

Put another way, an objective function of the type defined by (11.92) will have no impact whatsoever on the definition of the state variable.

Indeed, this conclusion can be cast in broader terms to equip us with a general modeling guideline. To see how it would be arrived at, assume that we have already identified the structure of τ_f and that the objective function q allows the following representation:

$$q(x_1, \dots, x_k) = q_1(v_1, x_1) \oplus q_2(v_2, x_2) \oplus \cdots \oplus q_k(v_k, x_k) \tag{11.98}$$

where $\oplus$ is a composition operator and $v_n = \tau_f(x_1, \dots, x_{n-1})$, $1 < n \leq k$.

Then the conditional problem induced by $(x'_1, \dots, x'_{n-1})$ would be of the following form:

$$\underset{(x_n,\dots,x_k)}{\text{opt}} \left\{q_1(v_1, x'_1) \oplus \cdots \oplus q_{n-1}(v_{n-1}, x'_{n-1}) \oplus q_n(v_n, x_n) \oplus \cdots \oplus q_k(v_k, x_k)\right\} \tag{11.99}$$

subject to

$$(x'_1, \dots, x_{n-1}, x_n, \dots, x_k) \in X \tag{11.100}$$

The point to note, however, is that if $\oplus$ is *Markovian* (see *Section 10.2*), then by definition, this problem is equivalent to

$$\underset{(x_n,\dots,x_k)}{\text{opt}} \{q_n(v_n, x_n) \oplus \cdots \oplus q_k(v_k, x_k)\} \tag{11.101}$$

subject to (11.100), in that both have the same optimal solutions.

The general modeling guideline that can be offered then asserts the following: An objective function admitting a decomposition such as that shown in (11.98) with $\oplus$ being a Markovian composition operator would have no bearing on the structure of the state variable. In such cases one would seek to establish the structure of the state only through an analysis of the constraints imposed on *Problem P.*

It should be pointed out, though, that although this guideline (or a variant thereof) applies to the overwhelming majority of cases encountered in dynamic programming, certain important problems clearly violate it. One is certain to come across problems where the objective function not only influences the structure of the state variable, but its impact on the state is a deeply felt one. The following is a case in point.

11.3.4 Example

Consider the problem where the objective function q has this form:

$$q(x_1, \dots, x_k) = \sum_{n=1}^{k-1} b_n(x_n, x_{n+1}) \tag{11.102}$$

Here we would decompose q as follows:

$$q(x_1,\dots,x_k) = \sum_{j=1}^{n-2} b_j(x_j,x_{j+1}) + b_{n-1}(x_{n-1},x_n) + \sum_{j=n}^{k-1} b_j(x_j,x_{j+1}) \tag{11.103}$$

$$= c_n + b_{n-1}(x_{n-1},x_n) + \sum_{j=n}^{k-1} b_j(x_j,x_{j+1}) \tag{11.104}$$

where now

$$c_n = \sum_{j=1}^{n-2} b_j(x_j,x_{j+1}) \ , \ n > 2 \tag{11.105}$$

with $c_1 = c_2 = 0$.

Thus, the conditional problem at $y = (x'_1,\dots,x'_{n-1})$ would be of the following form:

$$p(x'_1,\dots,x'_{n-1}) = \underset{(x_n,\dots,x_k)}{\text{opt}} \left\{ c_n + b_{n-1}(x'_{n-1},x_n) + \sum_{j=n}^{k-1} b_j(x_j,x_{j+1}) \right\} \tag{11.106}$$

or using (11.105),

$$p(x'_1,\dots,x'_{n-1}) = c_n + \underset{(x_n,\dots,x_k)}{\text{opt}} \left\{ b_{n-1}(x'_{n-1},x_n) + \sum_{j=n}^{k-1} b_j(x_j,x_{j+1}) \right\} \tag{11.107}$$

Clearly, the value of c_n is irrelevant for working out the optimal solutions to this problem. Consequently, $(x'_1,\dots,x'_{n-2})$ would have no bearing on τ_o. However, the term $b_{n-1}(x'_{n-1},x_n)$ is very much relevant. This means that the value of x'_{n-1} might be required for finding the optimal solutions to *Problem* $C(x'_1,\dots,x'_{n-1})$. To provide for this we would have to set

$$\tau_o(x_1,\dots,x_{n-1}) := x_{n-1} \tag{11.108}$$

in which case

$$s_n : = \tau(x_1,\dots,x_{n-1}) \tag{11.109}$$

$$= (\tau_f(x_1,\dots,x_{n-1}), \tau_o(x_1,\dots,x_{n-1}) \tag{11.110}$$

$$= (\tau_f(x_1,\dots,x_{n-1}), x_{n-1}) \tag{11.111}$$

Thus, if, for example,

$$\tau_f(x_1,\dots,x_{n-1}) = r - \sum_{j=1}^{n-1} x_j \tag{11.112}$$

then

$$s_n = \left(r - \sum_{j=1}^{n-1} x_j \ , \ x_{n-1} \right) \ , \ n = 2, \dots, k \tag{11.113}$$

For $n = 1$ we would then set $s_1 = (r, \varnothing)$ where $\varnothing$ denotes the empty sequence.

In brief, stating a problem with an objective function such as the one defined by (12.73) in terms of a dynamic programming model will generally cause an *expansion of the state variable* induced by the solution set X as defined by τ_f. This is so because one will have to incorporate the value of the decision variable x_{n-1} which will be required for finding the optimal solutions of the conditional problem at $(x'_1, \dots, x'_{n-1})$.

An exception to this will be the case where the state induced by the solution set X will already contain this information. The repercussions of an expanded state, as we know, can be severe. It can adversely affect the computational complexity of the algorithm, to impede in many cases the solution of the resulting functional equation.

I wish to point out, though, that there are ways to keep the adverse effects of this type of objective functions at bay. I examine this question in *Chapter 12.*

At this point I want to look at the more basic question of the position that ought to be adopted with regard to the state variable, given the state's immediate implications for the solution of the functional equation.

It is important to keep in mind that an expanded state variable need not be treated as though it were an edict from above. Indeed, sometimes one would have some control over the type of state that eventually would form part of the dynamic programming model that one would formulate. This control would be in the type of condition (Markovian or the Weak-Markovian) one would postulate. Obviously, a state variable forming part of a dynamic programming model underpinned by the Markovian condition can be expected to be "larger" than a state variable forming part of a dynamic programming model underpinned by the Weak-Markovian condition (for the same problem). Recall that opting for either the Markovian condition or the Weak-Markovian condition would depend entirely on whether it would be necessary to recover the *entire set* of optimal solutions for the problem considered or, whether recovering *an* optimal solution would suffice.

And to illustrate, the state variable formulated above forms part of a model whose underlying condition, that is (11.59), imposes a *strong equivalence relation* between *Problem $C(y)$* and *Problem $C(y')$*.

That is, $\tau_o(y) = \tau_o(y')$ entails that *Problem $C(y)$* and *Problem $C(y')$* have the *same* set of optimal solutions, thereby assuring the recovery of *all* the optimal solutions to the problem in question. Or, to put it more rigorously:

$$Y_n(s) := \{ y \in Y_n : \tau(y) = s \} \tag{11.114}$$

for $1 \leq n \leq k, s \in S_{n+1} := \{\tau(y) : y \in Y_n\}$. Namely, let $Y_n(s)$ denote the subset of Y_n whose elements induce the state s.

Then (11.59), in conjunction with (11.61), ensure that if $z^* \in Z^*(y)$ for some $y \in Y_n$, then $z^* \in Z^*(y')$, for all $y' \in Y_n(s)$.

Suppose, however, that the following less stringent condition is postulated: for any pair (n, s) such that $1 \leq n \leq k$ and $s \in S_n$ there exists a z^* such that $z^* \in Z^*(y)$, for all $y \in Y_n(s)$.

Then, the resulting dynamic programming model will be *Weak-Markovian* (*Section 10.2*). Mathematically we can express this condition as follows:

$$\bigcap_{y \in Y_n(s)} Z^*(y) \neq \varnothing \ , \ 1 \leq n < k, s \in S_{n+1} \tag{11.115}$$

For example, consider the case where the objective function q is defined thus:

$$q(x_1, \dots, x_k) = \max_{1 \leq n \leq k} \{q_n(x_n)\} \tag{11.116}$$

Since max is an associative operation, it follows that

$$q(x'_1, \dots, x'_{n-1}, x_n, \dots, x_k) = \max\left\{\max_{1 \leq j \leq n-1}\{q_j(x'_j)\} \ , \ \max_{n \leq j \leq k}\{q_j(x_j)\}\right\} \tag{11.117}$$

$$= \max\left\{c, \max_{n \leq j \leq k}\{q_j(x_j)\}\right\} \tag{11.118}$$

where

$$c := \max_{1 \leq j \leq n-1}\{q_j(x'_j)\} \tag{11.119}$$

Therefore

$$q(x'_1, \dots, x'_{n-1}) = \max\left\{c, \max_{n \leq j \leq k}\{q_j(x_j)\}\right\} \tag{11.120}$$

$$= c \left\lceil \max_{n \leq j \leq k}\{q_j(x_j)\}\right. \tag{11.121}$$

recalling that $a \lceil b := \max(a, b)$.

Now, because finding the optimal solutions to this problem calls for the value of c to satisfy (11.91), we need to define

$$\tau_o(x_1, \dots, x_{n-1}) := \max_{1 \leq j \leq n-1}\{q_j(x_j)\} \tag{11.122}$$

However, since $\lceil$ is monotone non-decreasing with respect to its right argument, any optimal solution to

$$\max_{(x_n, \dots, x_k)} \left\{\max_{n \leq j \leq k} q_j(x_j)\right\} \tag{11.123}$$

subject to $(x_n,\dots,x_k) \in Z(x'_1,\dots,x'_{n-1})$, will also be optimal for (11.121). This implies that if we set $\tau = \tau_f$, then for each pair (n,s), there exists a z such that $z \in Z^*(y)$, for all $y \in Y_n(s)$.

The implication therefore is that the state induced by τ_f will not require an expansion in this case.

To sum up, if the recovery of *an* optimal solution suffices, one would invoke the Weak Markovian condition (11.115) rather than (11.91).

11.3.5 Consistency

The state variable must be able to withstand a CONSISTENCY test. Namely, it is important to make sure that it satisfies the following relation:

$$[y,y' \in Y_n, \tau(y) = \tau(y')] \Longrightarrow \tau(y,x) = \tau(y',x) \tag{11.124}$$

for all $x \in X_{n+1}$ such that $(y,x) \in Y_{n+1}$.

Note that because the feasibility condition (11.61)-(11.62) assures that

$$\big[y,y' \in Y_n, x \in X_{n+1}, \tau(y) = \tau(y'), (y,x) \in Y_{n+1}\big] \Longrightarrow (y',x) \in Y_{n+1} \tag{11.125}$$

it follows that postulating $(y,x) \in Y_{n+1}$ in the context of (11.125) is akin to postulating $(y',x) \in Y_{n+1}$ and vice versa.

In essence what (11.125) requires is that s_{n+1} be uniquely determined by the triplet (n,s_n,x_n), namely that the state variable be *Markovian*. One important implication of this requirement of course is that it guarantees that the transition function is Markovian. That is, it guarantees that a function T with $T(n,\tau(y),x) = \tau(y',x)$ for all $y,y' \in Y_{n-1}$, and $x \in X_n$ such that $\tau(y) = \tau(y')$ and $(y,x) \in Y_n$ exists.

To illustrate this point, suppose that in the context of a certain problem we define

$$\tau(x_1,\dots,x_{n-1}) := \sum_{j=1}^{n-1} x_j \ , \ 1 < n \leq k+1 \tag{11.126}$$

Then by definition,

$$\begin{aligned}\tau(x_1,\dots,x_n) &= \sum_{j=1}^{n} x_j & (11.127)\\ &= \sum_{j=1}^{n-1} x_j + x_n & (11.128)\\ &= \tau(x_1,\dots,x_{n-1}) + x_n & (11.129)\end{aligned}$$

Thus, the value of $\tau(x_1,\dots,x_n)$ will be uniquely determined by the values of $\tau(x_1,\dots,x_{n-1})$ and x_n. Furthermore, (11.129) entails that

$$s_{n+1} = T(n,s_n,x_n) := s_n + x_n \tag{11.130}$$

The same idea can be expressed by requiring that τ satisfy the following recursive relationship:

$$\tau(y,x) = T(n,\tau(y),x) \ , \ 1 \leq n \leq k, y \in Y_{n-1}, x \in X_n, (y,x) \in Y_n \tag{11.131}$$

In words, the next state of the system must derive entirely from the current stage, current state, and current decision.

11.3.6 Example

Consider the global constraint

$$\sum_{j=1}^{k} a_j x_j + \sqrt{\sum_{j=1}^{k} b_j x_j} \quad \leq C \tag{11.132}$$

and suppose that we contemplate using the following state variable to represent it:

$$s_n = \tau(x_1,\dots,x_{n-1}) := \sum_{j=1}^{n-1} a_j x_j + \sqrt{\sum_{j=1}^{n-1} b_j x_j} \quad , \quad n = 2,\dots,k \tag{11.133}$$

with $s_1 = \tau(\varnothing) = 0$.

This representation is typically *in*consistent because in general

$$A + \sqrt{B} = A' + \sqrt{B'} \tag{11.134}$$

does not imply that

$$(A + a) + \sqrt{B + b} = (A' + a) + \sqrt{B' + b} \tag{11.135}$$

for all a and b of interest.

For example, consider the instance where $A = 3$, $A' = 4$, $B = 25$ and $B' = 16$ in which case we have

$$A + \sqrt{B} = 8 = A' + \sqrt{B'}$$

But for $a = 1$ and $b = 11$ be have

$$\begin{aligned}(A + a) + \sqrt{B + b} &= (3 + 1) + \sqrt{25 + 11} = 4 + 6 = 10\\ (A' + a) + \sqrt{B' + b} &= (4 + 1) + \sqrt{16 + 11} = 5 + \sqrt{27} \neq 10\end{aligned}$$

Hence, in general the above representation is inconsistent.

It is easy to verify that the representation defined by

$$s_n = \tau(x_1,\dots,x_{n-1}) := \left(\sum_{j=1}^{n-1} a_j x_j \ , \ \sum_{j=1}^{n-1} b_j x_j\right) \ , \ n = 2,\dots,k \tag{11.136}$$

with $s_1 = \tau(\varnothing) = (0,0)$, is consistent. In fact, it is easy to see that in this case

$$T(n,s,x) = (s(1) + a_n x \ , \ s(2) + b_n x)) \ , \ n = 1,2,\dots,k \tag{11.137}$$

and

$$\tau(y,d) = \left(\sum_{j=1}^{n-1} a_j x_j + a_n d \ , \ \sum_{j=1}^{n-1} b_j x_j + b_n d\right) \ , \ y = (x_1,\dots,x_{n-1}) \tag{11.138}$$

$$= T(n,\tau(y),d) \ , \ 1 \le n \le k, y \in Y_{n-1}, d \in X_n, (y,x) \in Y_n \tag{11.139}$$

as required by the consistency test. □

In practice, a formal consistency test for a proposed defintion of the state variable is rarely required because the inconsistency in the state's formulation would normally be noticed, by inspection, in the construction of the transition function T.

All the same, dynamic programming novices may find it useful to put their formulation of the state variable to a formal consistency test.

The preceding analysis gives the impression that because the constraints are concerned with *feasibility* whereas the objective function is concerned with *optimality*, a problem's constraints are treated in a fundamentally different way than its objective function in the construction of the state variable.

The question therefore is: since in both cases the *conditional analysis* is designed to *decompose* the problem, why decompose only the objective function? Why not decompose the constraints with a decomposition scheme similar to that employed to decompose the objective function?

As we shall see, this is indeed a good idea!

Decomposing the constraints with the same basic technique employed to decompose the objective function will facilitate the construction of τ_f.

11.4 Decomposition Revisited

The idea is then to approach the *conditional analysis* that the constraints and the objective function undergo in the construction of the state as the basis for unifying their treatment. To this end let us adopt the following simple convention: *a global constraint is a real-valued function of the decision variables.*

Let us now examine the consequences of this simple proposition.

To do this let us formulate a "semi-formal" model. That is, let us assume that there are k decision variables. So, a feasible solution to our problem is a sequence $\mathbf{x} = (x_1, \dots, x_k)$ that is a member in some set X that is a subset of $X' := \mathbb{D}^k$, where $\mathbb{D}$ denotes the decision space of the problem.

In this framework the feasibility issue is viewed as a *membership* issue: $\mathbf{x} \in X$. So, if we have m constraints, we can let $X^{(j)}$ denote the subset of X' pertaining to the j-th constraint, in which case we would have

$$X = \bigcap_{j=1}^{m} X^{(j)} \tag{11.140}$$

namely, for $\mathbf{x} \in X'$ to be feasible, it must satisfy *all* the constraints — that is, m constraints.

Hence, focusing on the role of the global constraints we can thereby focus on the implication of the requirement $\mathbf{x} \in X^{(j)}$ where j corresponds to the global constraint under consideration.

However, it proves far more convenient, indeed more effective, to translate the *set theoretic* representation $\mathbf{x} \in X^{(j)}$ of a constraint into an equivalent *functional* representation using the *characteristic function* of $X^{(j)}$. Formally, such a function is defined as follows:

$$C^{(j)}(\mathbf{x}) = \begin{cases} 1 & , \quad \mathbf{x} \in X^{(j)} \\ 0 & , \quad \mathbf{x} \notin X^{(j)} \end{cases} \tag{11.141}$$

So formally $C^{(j)}$ is a real-valued function on X' with values in $\{0, 1\}$. By construction, $C^{(j)}(\mathbf{x}) = 1$ if and only if $\mathbf{x}$ satisfies the j-th constraint under consideration.

Note that in this setting, an $\mathbf{x}$ is feasible if and only if it *maximizes* $C^{(j)}(\mathbf{x})$ over X' for all $j = 1, 2, \dots, m$. The implication is clear then: the decomposition of global constraints is similar to the decomposition of objective functions.

Now, since our aim is to consider a single global constraint at a time, it would be convenient to drop the superscript j so that the global constraint would be represented by a characteristic function C on X'. So let,

$$C(\mathbf{x}) = \begin{cases} 1 & , \quad \mathbf{x} \in X \\ 0 & , \quad \mathbf{x} \notin X \end{cases} \quad , \quad \mathbf{x} \in X' \tag{11.142}$$

Also, it would be natural to view C as a generalization of standard constraints used in optimization theory and mathematical programming, for instance, $c(\mathbf{x}) \leq b$, or $c(\mathbf{x}) \geq b$, or $c(\mathbf{x}) = b$ where b is a given numeric scalar and c is a real valued function on X'.

Such a generalization regards $\leq$, $\geq$ and $=$ as specific instances of a binary relation $\diamond$ on some abstract set so that the constraint is expressed as follows:

$$c(\mathbf{x}) \diamond \gamma \ , \ \mathbf{x} \in X' \tag{11.143}$$

That is, $\mathbf{x} \in X'$ is feasible — with respect to the constraint under consideration — if and only if $c(\mathbf{x}) \diamond \gamma^*$.

Formally, the relationship between the real-valued function c and the characteristic function C can be expressed as follows:

$$C(\mathbf{x}) = \xi\left(c(\mathbf{x})\right) \ , \ \mathbf{x} \in X' \tag{11.144}$$

where c is a function on X' with values in some set Γ and ξ is the function on Γ with values in $\{0, 1\}$ defined as follows:

$$\xi(\gamma) = \gamma \diamond \gamma^* \ , \ \gamma \in \Gamma \tag{11.145}$$

For example, consider the global constraint

$$\sum_{n=1}^{k} a_n x_n \leq A \tag{11.146}$$

where all the parameters are positive integers and $X' = \mathbb{R}_+^k$.

To formulate this constraint in the framework of the $\xi(c(\mathbf{x}))$ format, the left hand side of the constraint is treated as a function on $\mathbb{R}_+^k$ and ξ as an *indicator function*.

In short, in this case we can set

$$c(x_1, \dots, x_k) = \sum_{n=1}^{k} a_n x_n \tag{11.147}$$

$$\diamond = \leq \tag{11.148}$$

$$\Gamma = \mathbb{R}_+ \tag{11.149}$$

$$\gamma^* = A \tag{11.150}$$

$$\xi(\gamma) = \begin{cases} 1 & , \ \gamma \leq A \\ 0 & , \ \gamma > A \end{cases} \ , \ \gamma \in \mathbb{R}_+ \tag{11.151}$$

It should be appreciated that the highly abstract nature of the $c(\mathbf{x}) \diamond \gamma^*$ format, hence the $C(\mathbf{x}) = \xi(c(\mathbf{x}))$ format, renders it an extremely powerful modeling tool.

11.4.1 Example

Consider the generic constraint

$$\{x_1, x_2, x_3, \dots, x_k\} = \mathcal{K} := \{1, 2, 3, \dots, k\} \tag{11.152}$$

which requires $\mathbf{x}$ to be a *permutation* of $(1, 2, 3, \dots, k)$.

We can formally set

$$X' = \{1, 2, \dots, k\}^k \tag{11.153}$$

$$c(x_1, \dots, x_k) = \bigcup_{j=1}^{k} \{x_j\} \tag{11.154}$$

$$\diamond = \doteq \tag{11.155}$$

$$\Gamma = 2^{\mathcal{K}} \tag{11.156}$$

$$\gamma^* = \mathcal{K} \tag{11.157}$$

■

where 2^K denotes the power set of $\mathcal{K}$ and $\doteq$ denotes the *equality relation*: $A \diamond B = 1$ iff $A = B$, else $A \diamond B = 0$.

Consequently,

$$\xi(c(\mathbf{x})) = c(\mathbf{x}) \diamond \mathcal{K} \ , \ \mathbf{x} \in X' \tag{11.158}$$

$$= \left(\bigcup_{j=1}^{k} \{x_j\} \right) \diamond \{1,2,3,\dots,k\} \tag{11.159}$$

$$= \{x_1,\dots,x_k\} \diamond \{1,2,3,\dots,k\} \tag{11.160}$$

$$= \begin{cases} 1 & , \ \mathbf{x} \text{ is a permutation of } (1,2,\dots,k) \\ 0 & , \ \text{otherwise} \end{cases} \tag{11.161}$$

For instance, for $k = 5$ we have $\mathcal{K} = \{1,2,3,4,5\}$ so for $\mathbf{x} = (1,3,2,1,3)$ we have

$$\xi(c(1,3,2,1,3)) = c(1,3,2,1,3) \diamond \mathcal{K} \tag{11.162}$$

$$= \{1\} \cup \{3\} \cup \{2\} \cup \{1\} \cup \{3\} \diamond \{1,2,3,\dots,k\} \tag{11.163}$$

$$= \{1,2,3\} \diamond \{1,2,3,\dots,k\} \tag{11.164}$$

$$= 0 \tag{11.165}$$

On the other hand, for $\mathbf{x} = (1,3,2,5,4)$ we have

$$\xi(c(1,3,2,5,4)) = c(1,3,2,5,4) \diamond \mathcal{K} \tag{11.166}$$

$$= \{1\} \cup \{3\} \cup \{2\} \cup \{5\} \cup \{4\} \diamond \{1,2,3,4,5\} \tag{11.167}$$

$$= \{1,2,3,5,4\} \diamond \{1,2,3,4,5\} \tag{11.168}$$

$$= 1 \tag{11.169}$$

I note in passing that this global constraint figures prominently in many area of combinatorial optimization. $\square$

This clear, let us begin the investigation of how the structure of the state variable is induced by global constraints of the form $C(\mathbf{x}) = \xi(c(\mathbf{x}))$ with the simple case where c can be represented by a binary scheme, namely

$$c(x_1,\dots,x_k) = v(1,x_1) \oplus v(2,x_2) \oplus \cdots \oplus v(k,x_k) \tag{11.170}$$

where $\oplus$ is a binary operation and v is a real-valued function on $\mathcal{K} \times \mathbb{D}$. In this case function c is said to be *separable.*

Now suppose that $\oplus$ is associative, namely

$$(a \oplus b) \oplus c = a \oplus (b \oplus c) \tag{11.171}$$

This dispenses with the parenthesis on the right hand side of (11.170) thus freeing the order of execution of this composite structure.

All we need to know about $(x_1,\dots,x_n), n \leq k$, in order to evaluate $c(x_1,\dots,x_k)$ is the value of

$$\sigma_n := v(1,x_1) \oplus \cdots \oplus v(n,x_n) \tag{11.172}$$

namely, by construction

$$c(x_1,\dots,x_k) = \sigma_n \oplus v(n+1,x_{n+1}) \oplus \cdots \oplus v(k,x_k) \ , \ 1 \leq n < k \tag{11.173}$$

This means that in such a case σ_n would be the state representing $(x_1,\dots,x_n)$, namely using our standard notation the state variable induced by this scheme is $s_n = \sigma_{n-1}$. More explicitly, we can let

$$s_n := v(1,x_1) \oplus \cdots \oplus v(n-1,x_{n-1}) \ , \ n = 2,\dots,k+1 \tag{11.174}$$

with $s_1 := L_\oplus$, where $L_\oplus$ denotes the left identity element of $\oplus$.

Thus, in the case of (11.146) we have

$$\oplus = + \tag{11.175}$$

$$v(j,x_j) = a_j x_j \ , \ j = 1,2,\dots,k \tag{11.176}$$

$$s_n = \sum_{j=1}^{n-1} a_j x_j \ , \ n = 2,3,\dots,k+1 \tag{11.177}$$

with $s_1 = 0$, and in the case of (11.152) we can let

$$\oplus = \cup \tag{11.178}$$

$$v(j,x_j) = \{x_j\} \ , \ j = 1,2,\dots,k \tag{11.179}$$

$$s_n = \bigcup_{j=1}^{n-1} \{x_j\} \ , \ n = 2,3,\dots,k+1 \tag{11.180}$$

with $s_1 = \varnothing$.

If $\oplus$ is not associative, the function would still be separable but its evaluation — hence decomposition — would depend on the pattern of parentheses used in the evaluation of the right hand side of (11.170).

If the order of execution would be from left to right, then the above analysis would still be valid. However, if the evaluation would be carried out from right to left then the above scheme would not work. The point is that for the function τ_f discussed in §*11.3.1* to be consistent we must be able to express $\tau_f(x_1,\dots,x_{n+1})$ in terms of $\tau_f(x_1,\dots,x_n)$.

11.4.2 Example

Consider the following exotic global constraint:

$$\log\left(a_1 + b_1x_1 + \log\left(a_2 + b_2x_2 + \log\left(\cdots + \log\left(a_k + b_kx_k\right)\right)\cdots\right)\right) \leq A \tag{11.181}$$

If we let

$$\alpha \overleftarrow{\oplus} \beta := \alpha + \log(\beta) \tag{11.182}$$

then the constraint can be rewritten as follows:

$$v(1,x_1)\overleftarrow{\oplus} v(2,x_2)\overleftarrow{\oplus} \cdots \overleftarrow{\oplus} v(k,x_k) \leq A \tag{11.183}$$

where

$$v(n,x_n) := a_n + b_n x_n \ , \ n = 1,\dots,k \tag{11.184}$$

and $\overleftarrow{\oplus}$ indicates that the expression is evaluated from right to left.

Thus, we can let $\diamond = \leq$ and set

$$c(\mathbf{x}) = v(1,x_1)\overleftarrow{\oplus} v(2,x_2)\overleftarrow{\oplus} \cdots \overleftarrow{\oplus} v(k,x_k) \tag{11.185}$$

But ..., although c is separable

$$\begin{aligned} s_n &= v(1,x_1)\overleftarrow{\oplus} \cdots \overleftarrow{\oplus} v(n-2,x_{n-2})\overleftarrow{\oplus} v(n-1,x_{n-1}) \ , \ n = 2,\dots,k & (11.186)\\ &= v(1,x_1)\overleftarrow{\oplus}\Big(\cdots \overleftarrow{\oplus}\Big(v(n-2,x_{n-2})\overleftarrow{\oplus} v(n-1,x_{n-1})\Big)\cdots\Big) & (11.187)\end{aligned}$$

is not a sound state variable in this case because this scheme does not satisfy the *consistency* requirement. To see that this is so, note that by construction

$$\begin{aligned} s_{n+1} &= v(1,x_1)\overleftarrow{\oplus} \cdots \overleftarrow{\oplus} v(n-2,x_{n-2})\overleftarrow{\oplus} v(n-1,x_{n-1})\overleftarrow{\oplus} v(n,x_n) & (11.188)\\ &= v(1,x_1)\overleftarrow{\oplus}\Big(\cdots \overleftarrow{\oplus}\Big(v(n-2,x_{n-2})\overleftarrow{\oplus}\Big(v(n-1,x_{n-1})\overleftarrow{\oplus} v(n,x_n)\Big)\Big)\cdots\Big) & (11.189)\end{aligned}$$

hence, if $\overleftarrow{\oplus}$ is not associative then, s_{n+1} cannot, in general, be obtained from s_n and x_n.

To see this more clearly, set $k = 3$, in which case we have

$$s_2 = \log(a_1 + b_1 x_1) \tag{11.190}$$

$$s_3 = \log(a_1 + b_1 x_1 + \log(a_2 + b_2 x_2)) \tag{11.191}$$

$$s_4 = \log(a_1 + b_1 x_1 + \log(a_2 + b_2 x_2 + \log(a_3 + b_3 x_3))) \tag{11.192}$$

Clearly, generally, s_4 cannot be expressed in terms of s_3 and x_3 nor can s_3 be expressed in terms of s_2 and x_2.

Note, however, that if the decision variables are renamed so that x'_1 becomes x_k, x'_2 becomes x_{k-1} and so on, the constraint can be rewritten as follows:

$$\log\left(a'_k + b'_k x'_k + \log\left(a'_{k-1} + b'_{k-1}x'_{k-1} + \log\left(\cdots + \log\left(a'_1 + b'_1 x'_1\right)\right)\cdots\right)\right) \leq A \tag{11.193}$$

then this would be a valid scheme, namely we would have

$$s'_{n+1} = v'(n, x'_n) \overleftarrow{\oplus} s'_n \ , \ n = 1, \dots, k \tag{11.194}$$

observing that

$$s'_2 = \log(a'_1 + b'_1 x'_1) \tag{11.195}$$
$$s'_3 = \log(a'_2 + b'_2 x'_2 + \log(a'_1 + b'_1 x'_1)) = \log(a'_2 + b'_2 x'_2 + s'_2) \tag{11.196}$$
$$s'_4 = \log(a'_3 + b'_3 x_3 + \log(a'_2 + b'_2 x'_2 + \log(a'_1 + b'_1 x'_1))) \tag{11.197}$$
$$= \log(a'_3 + b'_3 x_3 + s'_3) \tag{11.198}$$

and if we set $s'_1 = 0$ then we would also have $s'_2 = v'(1, x'_1) \overleftarrow{\oplus} s'_1$.

It is important to ensure that this type of variable renaming does not affect adversely the problem's other constructs, for instance other global constraints and the objective function.

We shall encounter similar situations in subsequent sections of this chapter and in *Chapter 14* in the discussion on *forward decomposition schemes*. □

11.5 Infeasible States and Decisions

The explicit identification of feasible states and feasible decisions, as well as the construction of the sets of feasible states and feasible decisions, can often prove a difficult task.

To set the scene for a discussion on this matter, recall that in the context of the multistage decision model, $D(n, s)$ denotes the set of feasible decisions at stage n when the process is at state s and S_n denotes the set of states that are feasible at stage n. Furthermore, the assumption is that each modified problem has at least one feasible solution, namely $X(n, s) \neq \varnothing$.

It follows then that the feasible states and decisions are required to satisfy two basic inter-related conditions

· Any state $s \in S_n$ must be reachable from the initial state via $(n - 1)$ feasible transitions.
· From any state $s \in S_n$ it is possible to reach the final stage via $N - n + 1$ feasible transitions.

What difficulties can be encountered then in view of these conditions?

Although in many problems both conditions can be easily handled, this is not always the case. For example, there are cases where coping with the second condition is as difficult as solving the problem itself.

The accepted practice therefore is to "violate" these conditions and admit (provisionally) states and decisions that are infeasible. The dynamic programming functional equation is adjusted accordingly to accommodate the incorporation of infeasible states and decisions in the analysis. Provisions are made to insure that these infeasible states and decisions do not affect the optimality of the solutions generated by the functional equation.

The upshot of this is that often the state space used in the solution of the dynamic programing functional equation is much larger than the state space required by the dynamic programming model. So it is important to strike a proper balance between two — often contradicting — objectives:

· Keeping the state space as small as possible so as to minimize the number of modified problems.

· Facilitating an efficient generation of the state space.

It is important to note that lurking behind this practical technical matter is the more basic methodological issue: how does dynamic programming deal with cases where the problem under consideration does not have a feasible solution?

The following examples illustrate some of the technical issues associated with this practical aspect of dynamic programming.

11.5.1 Example

Consider the following knapsack problem

$$z^* := \max_{x_1,\dots,x_k} \sum_{n=1}^{k} c_n x_n \tag{11.199}$$

$$\sum_{n=1}^{k} a_n x_n = A \tag{11.200}$$

$$x_n \in \{0,1,2,\dots\}\ , n = 1,\dots,k \tag{11.201}$$

where all the parameters are positive integers. Note that the global constraint (11.200) is an *equality* constraint.

Now, suppose that, as before, we formulate the state variable as follows:

$$s_n := A - \sum_{j=1}^{n-1} a_j x_j\ ,\ \ n = 2,3,\dots,k+1 \tag{11.202}$$

with $s_1 := A$ and $S := \{0,1,2,\dots,A\}$ so that

$$T(n,s,x) := s - a_n x\ ,\ \ 1 \le n \le k, s \in S, x \in D(n,s) \tag{11.203}$$

Our task is to determine the sets $\{D(n,s)\}$ for all relevant stage-state pairs. But this is no easy matter because, given that the global constraint is an equality constraint, the final state s_{k+1} must be equal to 0.

So, the question is what decisions are admissible in the set $D(n,s)$ for $n < k$? As a matter of fact, an even more serious question arises here: how do we know that the problem is feasible? Strictly speaking we do not!

There are a number of tactics to deal with this difficulty. Of these the most commonly used is a *penalty* scheme aimed to ensure that an infeasible decision will occur in the optimal solution if and only if there are no feasible solutions to the problem. This approach allows a measure of freedom in the definition of $D(n, s)$.

For example, in our case we may set:

$$D(n,s) := \left\{0, 1, 2, \dots, \left\lfloor \frac{s}{a_n} \right\rfloor\right\} \ , \ 1 \leq n \leq k, s \in S \tag{11.204}$$

But, strictly speaking this is incorrect.

For instance, this construction accepts the trivial solution $\mathbf{x} = (0, 0, \dots, 0)$ in spite of the fact that this solution is infeasible if $A > 0$.

So to overcome this irritant, we can introduce a penalty at the final stage of the process, $n = k + 1$, namely we let

$$W(s) := \begin{cases} 0 & , \quad s = 0 \\ -\infty & , \quad s > 0 \end{cases} \tag{11.205}$$

and re-write the objective function as follows:

$$g(s_1, x_1, x_2, \dots, x_k) = \sum_{n=1}^{k} c_n x_n + W(s_{k+1}) \tag{11.206}$$

The functional equation would then be as follows:

$$f_n(s) = \max_{x \in D(n,s)} \{x c_n + f_{n+1}(s - x a_n)\} \ , \ n = 1, 2, \dots, k \tag{11.207}$$

with $f_{k+1}(s) := W(s), s \in S$.

If we obtain $f_1(A) = -\infty$, the conclusion is that the problem is not feasible. Otherwise, the problem is feasible and $f_1(A) = z^*$.

Alternatively, the above definition of $D(n, s)$ is maintained for $1 \leq n < k$, with the following modification for $n = k$:

$$D(N,s) := \begin{cases} \varnothing & , \quad \left\lfloor \dfrac{s}{a_k} \right\rfloor \neq \dfrac{s}{a_k} \\ \left\{ \dfrac{s}{a_k} \right\} & , \quad \left\lfloor \dfrac{s}{a_N} \right\rfloor = \dfrac{s}{a_k} \end{cases} \tag{11.208}$$

Note that if s is a multiple of a_k then $D(k, s)$ is a singleton, namely $D(k, s) = \{s/a_k\}$, whereas if s is not a multiple of a_k then $D(k, s) = \varnothing$.

In this case we invoke the convention that *maximization over an empty set yields the value* $-\infty$. So again, if we obtain $f_1(A) = -\infty$, the conclusion is that the problem is not feasible. Otherwise, the problem is feasible and $f_1(A) = z^*$. □

The next example illustrates the difficulties arising in the pre-configuration of the sets of feasible states pertaining to the stages of the process.

11.5.2 Example

Consider the instance of the knapsack problem featured in the previous example where the global equality constraint is as follows:

$$1203x_1 + 4087x_2 + 5086x_3 + 9045x_4 + 8056x_5 = 50000 \tag{11.209}$$

Formally we can set $S_1 = \{50000\}$ and compute S_n for $n = 2, 3, 4, 5$ according to this formula

$$\begin{aligned} S_{n+1} &= \{T(n, s, x) : s \in S_n, x \in D(n, s)\} && (11.210)\\ &= \{s - xa_n : s \in S_n, x \in D(n, s)\} && (11.211)\end{aligned}$$

However, as explained above, it is no easy matter to pre-configure the sets $D(n, s)$ for the relevant stage-state pairs. Furthermore, even if the ploy discussed above is used to overcome this difficulty, we may still be reluctant to eliminate all the infeasible states because the computational resources required in this effort can be considerable.

So, an easy way out of this predicament would be to let $S = \{0, 1, 2, \ldots, 50000\}$ and approximate S_n by S for all $n > 1$. This, no doubt is wasteful, but in the case of small scale problems this easy-way-out can be justified. In situations where this slack approach cannot be contemplated, the strategy one can adopt is to generate the sets S_n in an impromptu manner, while the functional equation is being solved.

Such a method — called *Push* — is discussed in *Chapter 15.* The discussion in *Chapter 14* is also relevant to this case. □

In the next section I examine the construction of the state variable from a more general perspective. My aim is to outline a coordinated treatment of the objective function and each of the global constraints.

11.6 State Aggregation

There are problems where two or more global constraints generate the same state variable. In such cases the state variables can be aggregated into a single state.

Obviously, this can have a dramatically positive effect on the structure of the dynamic programming model and its functional equation.

11.6.1 Example: Modified Knapsack Problem

Consider the following familiar (unbounded) knapsack problem:

$$z^* := \max_{x_1,\ldots,x_k} \sum_{j=1}^{k} c_j x_j \tag{11.212}$$

$$\sum_{j=1}^{k} a_j x_j \leq b \tag{11.213}$$

$$x_j \in \{0,1,2,\dots\} \ , \ \forall j \in \mathcal{K} := \{1,2,\dots,k\} \tag{11.214}$$

where all the coefficient are positive integers.

The single global constraint can be handled by the following simple state variable

$$s_j := b - \sum_{m=1}^{j-1} a_m x_m \ , \ j = 2,\dots,k+1 \tag{11.215}$$

with $s_1 = b$.

The other elements of the dynamic programming model are as follows:

$$S = \{0,1,2,\dots,b\} \tag{11.216}$$

$$D(n,s) = \left\{0,1,2,\dots,\left\lfloor \frac{s}{a_n} \right\rfloor\right\} \tag{11.217}$$

$$T(n,s,x) = s - xa_n \tag{11.218}$$

Alternatively, if we use state variables of the form

$$s_j := \sum_{m=1}^{j-1} a_m x_m \ , \ j = 2,\dots,k+1 \tag{11.219}$$

with $s_1 = 0$, then the other elements of the model will be as follows:

$$S = \{0,1,2,\dots,b\} \tag{11.220}$$

$$D(n,s) = \left\{0,1,2,\dots,\left\lfloor \frac{b-s}{a_n} \right\rfloor\right\} \tag{11.221}$$

$$T(n,s,x) = s + xa_n \tag{11.222}$$

To all intents and purposes these two formulations are equivalent meaning that it makes no difference which one will be used.

Suppose, however, that in addition, the following constraints are imposed on this classic problem:

$$x_1 \leq \beta_1 \tag{11.223}$$

$$x_1 + x_2 \leq \beta_2 \tag{11.224}$$

$$x_1 + x_2 + x_3 \leq \beta_3 \tag{11.225}$$

$$x_1 + x_2 + x_3 + x_4 \leq \beta_4 \tag{11.226}$$

$$\vdots \quad \vdots \qquad \vdots \qquad \vdots$$

$$x_1 + x_2 + x_3 + x_4 + \cdots + x_k \leq \beta_k \tag{11.227}$$

where $\beta_{i+1} \geq \beta_j, j = 1,2,\dots,k-1$.

Observe that the first constraint is local, but the other $k-1$ are global. The implication is clear: additional $k-1$ state variables will have to be incorporated in the model. The complete state variable would thus be as follows:

$$s_j := (s_j(1), s_j(2), \dots, s_j(k)) \ , \ j = 2, 3, \dots, k+1 \tag{11.228}$$

where

$$s_j(1) := b - \sum_{m=1}^{j-1} a_m x_m \ , \ j = 2, \dots, k+1 \tag{11.229}$$

and

$$s_j(i) := \beta_i - \sum_{m=1}^{j-1} x_m \ , \ j = 2, 3, \dots, k; \ i = 2, \dots, k \tag{11.230}$$

with $s_1(1) = b$ and $s_1(i) = \beta_i, i = 2, 3, \dots, k$.

Obviously, this is not a very attractive proposition. So consider instead the much simpler state variable, namely $s_j = (s_j(1), s_j(2))$, where $s_j(1)$ is given by (11.228) and

$$s_j(2) := \sum_{m=1}^{j-1} x_m \ , \ j = 2, \dots, k \tag{11.231}$$

with $s_1(2) = 0$.

This state variable can handle all the additional $k-1$ global variables, as $\beta_i - s_j(2)$ provides all we need to know about the range of feasible values of x_j dictated by the above constraints, namely

$$x_j \in \left\{0, 1, 2, \dots, \min\left\{\left\lfloor \frac{b - s_j(1)}{a_j} \right\rfloor, \beta_j - s_j(2)\right\}\right\} \tag{11.232}$$

This means that we can use a model based on the state variable

$$s_j := \left(\sum_{m=1}^{j-1} a_m x_m \ , \ \sum_{m=1}^{j-1} x_m\right) \ , \ j = 2, \dots, k \tag{11.233}$$

with $s_1 = (0, 0)$, in which case the other elements of the dynamic programming model would be as follows:

$$D(n, s) = \left\{0, 1, 2, \dots, \left\{\left\lfloor \frac{b - s(1)}{a_n} \right\rfloor, \beta_j - s(2)\right\}\right\} \tag{11.234}$$

$$T(n, s, x) = (s(1) + x a_n \ , \ s(2) + x) \tag{11.235}$$

Clearly, this is a far more attractive model. □

The following example is somewhat more intricate.

11.6.2 Example: Modified Knapsack Problem

Consider the following (unbounded) knapsack problem featured in the previous example, namely

$$z^* := \max_{x_1,\dots,x_k} \sum_{j=1}^{k} c_j x_j \tag{11.236}$$

$$\sum_{j=1}^{k} a_j x_j \leq b \tag{11.237}$$

$$x_j \in \{0,1,2,\dots\} \ , \ \forall j \in \mathcal{K} := \{1,2,\dots,k\} \tag{11.238}$$

where all the coefficient are positive integers.

Now, suppose that this problem is additionally made subject to the following k constraints

$$x_1+x_2+x_3+x_4+\cdots+x_k \leq \beta_1 \tag{11.239}$$

$$x_2+x_3+x_4+\cdots+x_k \leq \beta_2 \tag{11.240}$$

$$x_3+x_4+\cdots+x_k \leq \beta_3 \tag{11.241}$$

$$x_4+\cdots+x_k \leq \beta_4 \tag{11.242}$$

$$\vdots \qquad \vdots \qquad \vdots \quad \vdots$$

$$x_k \leq \beta_k \tag{11.243}$$

observing that the first $k-1$ constraints are global and the last constraint is a simple local constraint imposing an upper bound on x_k.

Since the decision variables are required to be non-negative, we can assume, with no loss of generality, that $\beta_j \geq \beta_{j+1}$ for all $j = 1,\dots,k-1$. If this is not so, then some of the constraints are superfluous and can be ignored.

In any case, this system of constraints can be represented far more succinctly as follows:

$$\sum_{m=j}^{k} x_m \leq \beta_j \ , \ j \in \mathcal{K} \tag{11.244}$$

Since the first $k-1$ constraints are global in nature, each will require a component in the state variable. That is, taking a simplistic approach we would let the state variable be defined as follows:

$$s_j = \tau(x_1,\dots,x_{j-1}) := (s_j(0), s_j(1),\dots,s_j(k-1)) \tag{11.245}$$

for $j = 2,\dots,k+1$, where

$$s_j(0) := b - \sum_{m=1}^{j-1} b_m x_m \tag{11.246}$$

$$s_j(i) := \begin{cases} \beta_i & , \ j < i \\ \beta_i - \sum\limits_{m=i}^{j-1} x_m & , \ j \geq i \end{cases} , \ i = 1, 2, \dots, k-1 \tag{11.247}$$

with $s_1 = \tau(\varnothing) = (b, \beta_1, \beta_2, \dots, \beta_{k-1})$.

Again, not a very attractive proposition

Observe, however, that the $k-1$ additional global constraints are closely related, which suggests that it might be possible to formulate a far more compact representation of the state that they induce.

To see how this can be done, note that by definition the ith additional global constraint can be re-written as follows:

$$x_j + x_{j+1} + \cdots + x_k \leq s_j(i) \ , \ \forall j \in \mathcal{K}, j \geq i \tag{11.248}$$

This means that insofar as decision variable x_j is concerned, this constraint dictates that

$$x_j \leq s_j(i) \ , \ j \geq 1 \tag{11.249}$$

It follows therefore that the impact of the additional $k-1$ global constraints on the range of feasible values of x_j can be summarized as follows:

$$x_j \leq \min\{s_j(1), \dots, s_j(j)\} \ , \ j \in \mathcal{K} \tag{11.250}$$

The implication is then that insofar as the additional $k-1$ global constraints are concerned, at stage j of the multistage decision process it is not essential to know all the specific values of $s_j(1), \dots, s_j(k-1)$. The smallest of these values will suffice to determine the range of feasible values of x_j.

In short, the vector $(s_j(1), \dots, s_j(k-1))$ can be replaced by the scalar

$$\underline{s}_j := \min\{s_j(1), \dots, s_j(j)\} \tag{11.251}$$

This analysis suggests a state variable of the form $s_j = (u_j, v_j)$ where

$$u_j := b - \sum_{m=1}^{j-1} b_m x_m \tag{11.252}$$

$$v_j := \min\left\{\beta_1 - \sum_{m=1}^{j-1} x_m \ , \ \beta_2 - \sum_{m=2}^{j-1} x_m \ , \dots \ , \ \beta_{j-1} - \sum_{m=j-1}^{j-1} x_m \ , \ \beta_j\right\} \tag{11.253}$$

This means that the initial state would be

$$s_1 := (b, \beta_1) \tag{11.254}$$

and subsequent states would be defined as follows:

$$s_{j+1} = T(j, s_j, x_j) := (s(1) - a_j x_j \ , \ \min(s_j(2) - x_j, \beta_j)) \tag{11.255}$$

for $j \in \mathcal{K}$.

This state variable is manifestly more attractive than the simplistic one discussed above. But can we do even better than that?

Suppose that we reverse the labeling of the stages, so that stage 1 becomes stage k, stage 2 becomes stage $k-1$ and so on. The same problem would then take the following form

$$z^* := \max_{x'_1,\dots,x'_k} \sum_{j=1}^{k} c'_j x'_j \tag{11.256}$$

$$\sum_{j=1}^{k} a'_j x'_j \leq b \tag{11.257}$$

$$x'_j \in \{0,1,2,\dots\}\ ,\ \forall j \in \mathcal{K} := \{1,2,\dots,k\} \tag{11.258}$$

where $x'_j = x_{k-j+1}$, $a'_j = a_{k-j+1}$ and $c'_j = c_{k-j+1}$. The additional constraints would be as follows:

$$\begin{aligned}
x'_k+x'_{k-1}+x'_{k-2}+x'_{k-3}+\cdots+x'_1 &\leq \beta_1 && (11.259)\\
x'_{k-1}+x'_{k-2}+x'_{k-3}+\cdots+x'_1 &\leq \beta_2 && (11.260)\\
x'_{k-2}+x'_{k-3}+\cdots+x'_1 &\leq \beta_3 && (11.261)\\
x'_{k-3}+\cdots+x'_1 &\leq \beta_4 && (11.262)\\
\vdots \qquad \vdots \qquad \vdots \quad &\ \vdots\\
x'_1 &\leq \beta_k && (11.263)
\end{aligned}$$

so that it would be possible to rewrite them as follows

$$\begin{aligned}
&x'_1 && \leq \beta'_1 && (11.264)\\
&x'_1+x'_2 && \leq \beta'_2 && (11.265)\\
&x'_1+x'_2+x'_3 && \leq \beta'_3 && (11.266)\\
&x'_1+x'_2+x'_3+x'_4 && \leq \beta'_4 && (11.267)\\
&\vdots \quad \vdots \qquad \vdots && \vdots\\
&x'_1+x'_2+x'_3+x'_4+\cdots+x'_k && \leq \beta'_k && (11.268)
\end{aligned}$$

where $\beta'_i = \beta_{k-i+1}, i = 1,2,\dots,k$.

These additional constraints are identical to the constraints examined in *Example 11.6.1,* meaning that the k state variables that they induce can be aggregated into the single variable

$$s'_j := \sum_{i=1}^{j-1} x'_j\ ,\ j = 1,2,\dots,k \tag{11.269}$$

This formulation is, no doubt, more attractive (better?) then the two discussed above.

I come back to this type of constraints in *Chapter 14* in the discussion on *forward decomposition schemes.* □

The reader is cautioned, however, that the process of aggregating state variables induced by global constraints can be very tricky. It must therefore be carried out carefully.

The following example illustrates aggregation of state variables affiliated with the objective function.

11.6.3 Example

Consider the following problem:

$$\operatorname*{opt}_{x_1,\dots,x_k} \sum_{n=1}^{k} a_n x_n + \sqrt{\sum_{n=1}^{k} b_n x_n} \tag{11.270}$$

$$\sum_{n=1}^{k} b_n x_n \leq B \tag{11.271}$$

$$\sum_{n=1}^{k} c_n x_n \leq C \tag{11.272}$$

$$x_n \in \{0,1,2,\dots\} \ , \ n = 1,2,\dots,k \tag{11.273}$$

where all the parameters are positive integers.

There are two global constraints so, two components are required to represent them in the state variable. In the usual manner then let

$$s_n = \left(B - \sum_{j=1}^{n-1} b_j x_j \ , \ C - \sum_{j=1}^{n-1} c_j x_j \right) \ , \ n = 1,\dots,k \tag{11.274}$$

so that $s_n(1)$ denotes the amount of the "B" resource available at stage n and $s_n(2)$ denotes the amount of the "C" resource available at stage n and the initial state is $s_1 = (B, C)$.

In line with this set

$$S = \{0,1,2,\dots,B\} \times \{0,1,2,\dots,C\} \tag{11.275}$$

$$S_1 = \{(B,C)\} \tag{11.276}$$

$$D(n,s) = \left\{ 0,1,2,\dots, \left\lfloor \min\left\{ \frac{s(1)}{b_n} \ , \ \frac{s(2)}{c_n} \right\} \right\rfloor \right\} \tag{11.277}$$

$$T(n,s,x) = (s(1) - x b_n \ , \ s(2) - x c_n) = s - x(b_n, c_n) \tag{11.278}$$

for $n = 1,2,\dots,k \ ; \ s \in S, x \in D(n,s)$.

Since the objective function is non-separable, on the face of it, an additional component is required in the state variable to represent the sum under the $\sqrt{}$, that is

$$s_n(3) = \sum_{j=1}^{n-1} b_j x_j \ , \ j = 1, 2, \dots, k \tag{11.279}$$

However, this turns out to be unnecessary because this component of s is uniquely determined by $s(1)$, namely

$$s(3) = B - s(1) \tag{11.280}$$

This means that to complete the multistage decision model we can set

$$w(n, s, x) = xa_n \ , \ n = 1, 2, \dots, k, s \in S, x \in D(n, s) \tag{11.281}$$

$$W(s) = \sqrt{B - s(1)} \ , \ s \in S \tag{11.282}$$

$$\oplus = + \tag{11.283}$$

The affiliated functional equation would then be as follows:

$$f(s) = \underset{x \in D(n,s)}{\text{opt}} \ \{xa_n + f_{n+1}(s - x(b_n, c_n))\} \tag{11.284}$$

for $1 \leq n \leq k, s \in S$, with

$$f_{k+1}(s) := W(s) = \sqrt{B - s(1)} \ , \ s \in S \qquad \square \tag{11.285}$$

Still things are not always as nice and easy. The following example illustrates a trickier situation arising from a conflict between an objective function and a global constraint.

11.6.4 Example

Consider the problem

$$\underset{\mathbf{x}}{\text{opt}}\, g(\mathbf{x}) := a_1 x_1 + \big[a_2 x_2 + [a_3 x_3 + \cdots [\cdots + [a_k x_k]^p \cdots]^p \cdots]^p\big]^p \tag{11.286}$$

subject to

$$\beta(\mathbf{x}) := b_1 x_1 + \left[b_2 x_2 + \left[b_3 x_3 + \left[\cdots + [b_k x_k]^2 \cdots \right]^2 \right]^2 \right]^2 \leq B \tag{11.287}$$

$$x_n \in \{0, 1, 2, \dots\} \ , \ n = 1, 2, \dots, k \tag{11.288}$$

where all the parameters are positive integers.

By inspection, it is clear that the objective function g is separable and can be decomposed by the scheme

$$\rho(n, s_n, x_n, r) = (a_n x_n) \overleftarrow{\oplus} r := a_n x_n + r^p \ , \ n = 1, 2, \dots, k - 1 \tag{11.289}$$

observing that the binary operator $\overleftarrow{\oplus}$ is non-associative and is executed from right to left.

So far so good.

The global constraint (11.287) is decomposed in a similar fashion, using a non-associative binary operation that is executed from right to left, observing that

$$\beta(\mathbf{x}) = b_1 x_1 \overleftarrow{\otimes} b_2 x_2 \overleftarrow{\otimes} \cdots \overleftarrow{\otimes} a_k x_k \tag{11.290}$$

where

$$c \overleftarrow{\otimes} d := c + d^2 \tag{11.291}$$

However, remember that, as explained in *Example 11.4.2,*, to decompose (11.287), the indices of the decision variables must be reversed to satisfy the consistency condition. This condition requires that the state generated by $(x_1, \dots, x_{n+1})$ be expressed in terms of the state generated by $(x_1, \dots, x_n)$ and x_{n+1}.

But reversing the indices of the decision variables will render the objective function under consideration non-separable with respect to the state generated by this constraint.

The difficulty that will arise is that for the constraint to allow decomposition the composition function must operates from *left to right.* In contrast, the objective function requires a composition function that operates from *right to left.*

In this example, both the objective function and the constraint are decomposable from "right to left", hence reversing the indices of the decision variables will not do the trick.

Remark: As we shall see in *Chapter 14*, the objective function of a dynamic programming model can often be decomposed from "left to right" so the difficulty encountered above is not in fact pathologic. Indeed, in *Chapter 14* a dynamic programming model is formulated for the problem analyzed above. □

There are of course pathologic cases where no amount of modeling can alleviate the situation. The following example illustrates such a case.

11.6.5 Example: Modified Knapsack Problem

Consider the following interesting modified nonlinear knapsack problem:

$$z^* := \min_{x_1,\dots,x_k} \sum_{n=1}^{k} \left[a_n x_n - \sum_{j=1}^{k} c_j x_j \right]^2 \tag{11.292}$$

$$\sum_{n=1}^{k} b_n x_n = B \tag{11.293}$$

$$x_n \in \{0,1,2,\dots\}\ ,\ n=1,2,\dots,k \tag{11.294}$$

where all the parameters are positive integers.

At first sight the situation seems hopeless.

That is, there seems to be no obvious way to separate this objective function other than using the *trivial decomposition scheme* where the state amounts to the complete history of the process, namely $s_n = (x_1,\dots,x_{n-1})$. But this, needless to say, is highly impractical.

So, depending on how hard-pressed one is, or what computational resources are available, one may consider instead the following round-about approach.

Consider the following *parametric problem*:

$$z^*(C) := \min_{x_1,\dots,x_k} \sum_{n=1}^{k} [a_n x_n - C]^2\ ,\ C \in \mathbb{R}_+ \tag{11.295}$$

$$\sum_{n=1}^{k} b_n x_n = B \tag{11.296}$$

$$\sum_{n=1}^{k} c_n x_n = C \tag{11.297}$$

$$x_n \in \{0,1,2,\dots\}\ ,\ n=1,2,\dots,k \tag{11.298}$$

Now, suppose that we solve this problem for all values of C that can be generated by feasible solutions to the original problem. Let C^* denotes the value of C that yields the smallest value of $z^*(C)$. Then clearly C^* is the "optimal" C: any optimal solution for the parametric problem with $C = C^*$ is an optimal solution to the original problem.

A special case of interest is when $c_n = a_n/k$. Details on this case can be found in Sniedovich [1980a, 1983]. □

For illustrative purposes, perhaps the most edifying example of state aggregation is the standard shortest path problem discussed in *Chapter 10.* So, let us go back to this problem and reexamine it from this perspective.

11.6.6 Shortest Path Problem

Consider the following formulation of the standard shortest path problem

$$z^* := \min_{\substack{x_1,\dots,x_k \\ k}} \sum_{j=1}^{k-1} d(x_j, x_{j+1}) \tag{11.299}$$

$$(x_j, x_{j+1}) \in \mathit{Arcs}\ ,\ j=1,2,\dots,k-1 \tag{11.300}$$

$$x_1 = A \tag{11.301}$$

$$x_k = B \tag{11.302}$$

where:

· A = origin node.
· B = destination node.
· $Arcs$ = set of available arcs.
· x_j = j-th node on the path.
· $d(i,j)$ = length of arc (i,j).

In words, the objective is to find the shortest path from node A to node B on a network where each arcs is represented by a pair of successive nodes.

Note that there are $k-1$ global constraints each involving two successive nodes representing an arc on the path. Thus, conceptually we can let the state variable be a vector $\mathbf{s} = (s(1), \ldots, s(k-1))$ where $s(j)$ denotes the component of the state variable induced by the global constraint $(x_j, x_{j+1}) \in Arcs$.

However, because each such global constraint involves only two successive decision variables, the impact of each constraint on the state variable is in fact very limited. Indeed, the set of feasible values for x_j is completely independent of all $s(i), i = 1, 2, \ldots, k-1$, except s_{j-1}.

Thus, the $k-1$ global constraints can be aggregated into a single-component state variable, namely

$$s_j := x_{j-1} \text{ (j-th node on the path)}, j = 2, 3, \ldots, k \tag{11.303}$$

and written as follows:

$$x_j \in Suc(s_j) \ , \ j = 2, 3, \ldots, k \tag{11.304}$$

where $Suc(j)$ denotes the set of immediate successors of node j.

This choice satisfies the consistency conditions and produces the simple transition function $T(s,x) = x$. $\square$

11.7 Nodes as States

With the exception of the brief illustration above, to this point my discussion was conducted entirely from one perspective. The assumption was that the problem in question is to be formulated in terms of a multistage dynamic programming model. Recall, however, that in *Chapter 10* I introduced an alternative dynamic programming model — the shortest path model — and I illustrated that for certain problems the latter offers a far more suitable modeling framework.

The question obviously is whether the analysis of how to approach the construction of the state in a multistage dynamic programming model would also apply to the shortest path model. Generally speaking the answer to this is in the affirmative. It will indeed be possible to use the same strategy to construct the state variable in the context of the shortest path model.

However, it is important to note that the impact of global constraints and/or non-Markovian objective functions in the context of this model is far more severe. Here, the nodes stand to lose their original intuitive meaning.

To see why this is so, recall that the shortest path model is stated in terms of a directed graph where the nodes represent states and the arcs decisions. Thus, in the framework of this model the state space S is a finite set whose elements are the nodes of the graph; for each $s \in S$ the set $D(s)$ contains all the immediate successors of node s, and the transition function is of the form $T(s,x) = x$. The lengths of the arcs are stipulated by a function w such that $w(i,j)$ is interpreted as giving the length of arc (i,j). The length of a path $(s_1, x_1, x_2, \ldots, x_m)$ is given by

$$g(s_1, x_1, x_2, \ldots, x_m) = w(s_1, x_1) \oplus w(x_1, x_2) \oplus w(x_2, x_3) \oplus \cdots \oplus w(x_{m-1}, x_m). \quad (11.305)$$

with $\oplus$ being a Markovian or a Weak-Markovian composition operator.

Now, as we saw in the preceding sections of this chapter, global constraints and/or non-Markovian objective functions require taking into account an array of considerations that eventually show up in the state variable. The implication of this for the shortest path model is that this can result in a distortion of the nature of the nodes.

To illustrate this point let us first consider an expansion brought about by a global constraint and then an expansion caused by a non-Markovian objective function.

11.7.1 Example: Modified Shortest Path Problem

Suppose that the standard shortest path problem is additionally made subject to the following constraint:

> *The path from the node of origin to the terminal node must consist of no more than M arcs.*

Plainly, this is a global constraint. In keeping with the approach formerly delineated, to determine how this constraint might affect the structure of the state variable we would assume that a number of decisions have already been made — i.e. a number of arcs have already been traversed.

The question would then be: what information about these decisions (arcs) is required in order to determine the remaining feasible decisions?

The answer would be worked out as follows: as the original problem is a standard shortest path problem, the last node on the initial partial path must be identified, for otherwise it will be impossible to determine the next (feasible) transition.

Furthermore, to meet the above global constraint, the number of arcs still permitted, or the number of arcs already traversed, must also be known, otherwise it will be impossible to decide at what point to terminate the process. The latter consideration will be reflected in the state to affect an expansion of it.

Symbolically, the expanded state variable will be a pair $s = (v, i)$ where v is the current vertex (node) and i the number of additional transitions (arcs) allowed. The expanded transition function would therefore be

$$T((v, i), x) = (x, i - 1) \tag{11.306}$$

Of course, the initial state would be $s = (v^\circ, m)$ where v° is the node of origin. □

11.7.2 Example

Consider the case where the length of a path is equal to *the difference between the lengths of the longest and the shortest arcs on that path.* Formally, the objective function can be defined as follows:

$$\begin{aligned} g(s_1, x_1, x_2, \dots, x_k) = \max\{&w(s_1, x_1), w(x_1, x_2), \dots, w(x_{k-1}, x_k)\} \\ - \min\{&w(s_1, x_1), w(x_1, x_2), \dots, w(x_{k-1}, x_k)\} \end{aligned} \tag{11.307}$$

Reasoning along the lines of the preceding example, we would begin by postulating an initial segment, $y = (s_1, x'_1, x'_2, \dots, x'_{m-1})$, of a feasible path and then proceed to optimize the conditional problem induced by this sub-path.

Observe that because min and max are associative operations, we have

$$\begin{aligned} g(y, x_m, \dots, x_k) = \max\big\{&c, w(x'_{m-1}, x_m), w(x_m, x_{m+1}), \dots, w(x_{k-1}, x_k)\big\} \\ - \min\big\{&d, w(x'_{m-1}, x_m), w(x_m, x_{m+1}), \dots, w(x_{k-1}, x_k)\big\} \end{aligned} \tag{11.308}$$

where

$$c := \max\big\{w(s'_1, x'_1), w(x'_1, x'_2), \dots, w(x'_{m-2}, x'_{m-1})\big\} \tag{11.309}$$

and

$$d := \min\big\{w(s'_1, x'_1), w(x'_1, x'_2), \dots, w(x'_{m-2}, x_{m-1})\big\} \tag{11.310}$$

By inspection, it follows that to optimize g under these terms we need to have on hand the values of c, d and x_{m-1}. Because in the context of the standard shortest path problem x_{m-1} is the vertex reached after $m - 1$ transitions, the state will have to be expanded to absorb the triplet $s = (v, c, d)$ where v is the current vertex and c and d are the scalars defined above.

In other words, three items of information are required in order to find the best possible continuation to a given sub-path $y = (x_1, x_2, \dots, x_{m-1})$:

· The current vertex, x_{m-1}.
· The length of the longest arc traversed thus far, c.
· The length of the shortest arc traversed thus far, d.

The initial state would then be $s = (v^\circ, -\infty, \infty)$ and the expanded transition function would take this form:

$$T((v,c,d),x) = ((x, \max\{c, w(v,x)\}, \min\{d, w(v,x)\}) \qquad \square \quad (11.311)$$

In summary then, it is important to appreciate the effect of global constraints and/or non-Markovian objective functions on the standard shortest path problem when the latter is formulated in terms of a dynamic programming model. The point is that a priori there is no assurance that a shortest path problem stated in terms of a directed graph will be amenable to a valid dynamic programming formulation if the states are defined as the nodes of the graph.

More generally, there is a marked difference between the role that nodes and arcs have in the framework of graph theory and the role they have in the sequential decision models deployed by dynamic programming. Here, they often — but not always — represent states and decisions, respectively.

So, whereas in the context of graph theory nodes and arcs of a directed graph are abstract concepts devoid of any content, in the context of sequential decision models they can represent complex objects with "real meaning".

In particular, the nodes and arcs of a sequential decision model can be quite different from the nodes and arcs of a directed graph deployed to describe certain aspects of the problem.

11.7.3 Example: Assignment Problem

Consider an instance of the *Assignment Problem* defined by the following table of "assignment costs".

	Painting	*Cooking*	*Washing*	*Driving*
Jim	3	4	⋆	8
Anne	6	7	⋆	3
Mary	⋆	3	4	⋆
Adam	⋆	⋆	5	3

The numbers here stipulate the cost (AUD$) associated with the assignment of the 4 persons to the 4 jobs. The objective is to find the least costly assignment.

The problem can be described graphically as shown in Figure 11.1. The length of an arc represents the associated assignment cost.

To construct a dynamic programing model for this problem, let x_j denote the task assigned to person j. Then a feasible solution to the problem is a *permutation* of the tasks — with the exceptions implied by the $\star$ in the table, and missing arcs in the picture. For example, the solution $\mathbf{x} = (C, P, D, W)$

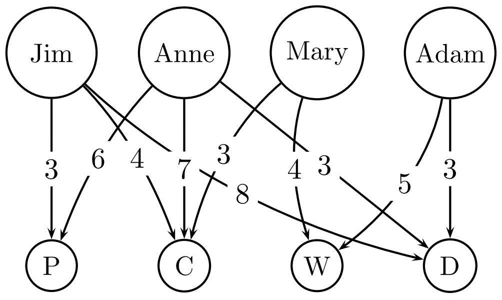


Figure 11.1: Assignment problem

assigns Cooking to Jim, Painting to Anne, Driving to Mary and Washing to Adam.

To ensure that the decisions are feasible, the tasks already assigned are recorded, to prevent re-assignment. Alternatively, the unassigned tasks are recorded. Let the state variable s denote the set of unassigned tasks.

Based on this preliminary informal analysis, we can set up the following sequential decision model (using our standard notation):

$$S = \text{ power set of } \{\text{P,C,W,D}\} \tag{11.312}$$
$$D(s) = s \tag{11.313}$$
$$T(s,x) = s\backslash\{x\} \tag{11.314}$$
$$\sigma = \{P, C, W, D\} \tag{11.315}$$

It is clear that the state $s = \varnothing$ is terminal. However, because of the exceptions, other states can also be terminal.

The state transition model representing the set of feasible decisions is a directed graph where each node represent a set of tasks. The complete graph is shown in Figure 11.2.

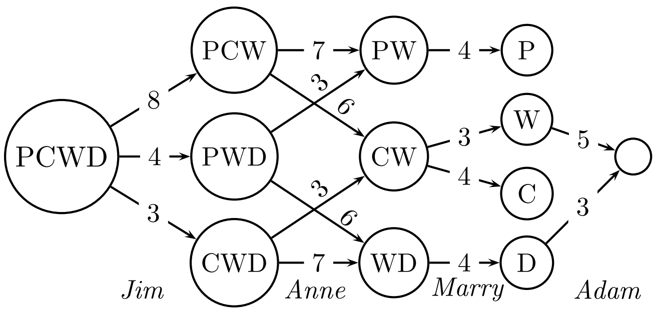


Figure 11.2: State transition graph for the assignment problem

The costs $w(s, x)$ shown on the arcs denote the cost of assigning job x to person $4 - |s| + 1$, where person 1 is *Jim*, person 2 is *Anne*, person 3 is *Mary* and person 4 is *Adam*. The cost of an inadmissible assignment is assumed to be equal to ∞ and is not shown on the graph.

Note that in addition to the obvious terminal state $s = \varnothing$, there are two other terminal states namely $s = \{P\}$ and $s = \{C\}$ (The Painting and Cooking jobs cannot be assigned to Adam). However, since these two states do not lead to a complete assignment, we let $S'' = \{\varnothing\}$ be the set of admissible terminal states.

The objective is to find the least costly sequence of decisions which begins at the initial state $\sigma = \{P, C, W, D\}$ and ends at the terminal state $S'' = \{\varnothing\}$.

Using our standard notation, the functional equation is as follows:

$$f(s) = \min_{x \in s} \{w(s,x) + f(s \backslash \{x\})\} \ , \ s \neq \sigma \tag{11.316}$$

with $f(\sigma) = 0$.

Here $f(s)$ denotes the least costly way of reaching state s from the initial state $\sigma = \{P, C, W, D\}$.

For the record, I should point out that, the *Assignment Problem* has other formulations. The most commonly encountered formulation is the following:

$$z^* := \min_x \sum_{i=1}^{k} \sum_{j=1}^{k} w_{i,j} x_{i,j} \tag{11.317}$$

$$\sum_{i=1}^{k} x_{i,j} = 1 \ , \ j = 1, 2, \ldots, k \tag{11.318}$$

$$\sum_{j=1}^{k} x_{i,j} = 1 \ , \ i = 1, 2, \ldots, k \tag{11.319}$$

$$x_{i,j} \in \{0, 1\} \ , \ i, j = 1, 2, \ldots, k \tag{11.320}$$

where

$$x_{i,j} := \begin{cases} 1 & , \ \text{assign job } j \text{ to person } i \\ 0 & , \ \text{otherwise} \end{cases} \tag{11.321}$$

and $w_{i,j}$ denotes the cost of assigning job j to person i.

What is more, there are highly efficient specialized algorithms available for this problem. I examine it here only to illustrate the modeling aspects of dynamic programming and to point out that dynamic programming is clearly unsuitable for this case: the size of the state space grows exponentially with the size of the problem (number of jobs), meaning the *Curse of Dimensionality*. $\square$

The following is no doubt the ultimate example of a shortest path problem where the nodes of the graph are not proper state variables.

11.7.4 Simple Path Problems

There are situations where the optimal shortest path sought is required to be *simple*, namely a path with no repeated nodes. If the graph under consideration is acyclic then obviously this requirement is superfluous. But if the graph is cyclic, then in general there is no assurance that the optimal path is simple. Furthermore, there are cases where a simple optimal path exists whereas a cyclic optimal path does not exist because the (cyclic) problem is unbounded below (see *Example 10.7.1*).

To ensure that the optimal path of a cyclic shortest path dynamic programming model is simple, we do as follows. For each node n that is part of a cycle, the state variable is defined as a pair $s = (n, v)$ where v denotes the set of nodes already visited that are on the same cycle as n. Thus, we let

$$D(n,v) = \{x \in Suc(n) : x \notin v\} \ , \ n \in A \tag{11.322}$$

$$T(n,v,x) = (x, v \cup \{x\}) \ , \ x \in D(n,v) \ , \ n \in A, x \in A \tag{11.323}$$

where A denotes the set of nodes on the cycles and $Suc(n)$ denotes the set of immediate successors of node n.

For other nodes we can nominally set $s = (n, \varnothing)$, where $\varnothing$ denotes the empty set, and

$$D(n,v) = Suc(n) \ , \ n \notin A \tag{11.324}$$

$$T(n,v,x) = (x,v) \ , x \in D(n,v), x \notin A \tag{11.325}$$

For example, consider the shortest path problem associated with the graph shown in Figure 11.3.

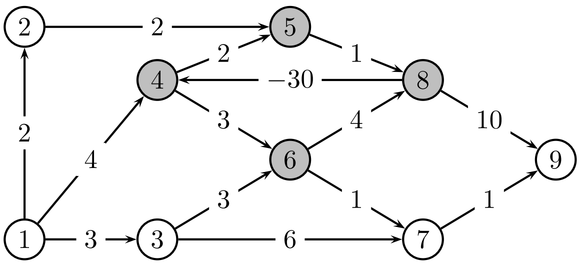


Figure 11.3: Acyclic shortest path problem

By inspection, the nodes in $A = \{4, 5, 6, 8\}$ are on the cycles. Thus, for these nodes a state is a pair $s = (n, v), v \subseteq A$, whereas for the other nodes a state is a pair $s = (n, \varnothing)$. Note that in this case v can take $2^{|A|} = 2^4 = 16$ values.

The difficulty is that often it is unknown in advance what nodes are on the cycles, so that v must be appended practically to all the nodes. However, . . . this can easily trigger the *Curse of Dimensionality!*.

If the graph is large but it is known in advance that only a small number of nodes are on the cycles, it may prove worthwhile to identify the cycles first (see *Example 16.4.4*). This will yield a relatively small set of possible values of v and v will be appended only to a relatively small number of nodes. □

11.8 Multistage vs Sequential Models

Recall that I pointed out in *Chapter 10,* that the choice between a multistage decision model and a sequential decision model is essentially a matter of convenience and style.

11.8.1 Example: Equipment Replacement Problem

Consider the following familiar generic *equipment replacement problem*: A machine is needed for the next K periods, say years. The life span of a new machine is $L < K$ periods and the total net cost of maintaining it for l years is \$$c(l)$. The cost of a new machine is \$$C$. The machine can be purchased at the beginning of each period. The salvage value of an old machine of age a (periods) is \$$B(a)$. Assume that a new machine is purchased at the beginning of the planning horizon. What is the optimal replacement policy?

This problem is concerned with a tradeoff between the cost of purchasing a new machine and the cost incurred by the maintenance of an aging machine.

We can approach the modeling of this problem in two different ways. One will result in a multistage model the other in a sequential decision model.

11.8.1.1 Multistage model

Here the question at the beginning of each period is as follows: should the old machine be kept, or should a new machine be purchased? Accordingly, we define the decision variable as follows:

$$x_j := \begin{cases} 1 & , \ \textit{if a new machine is purchased in period } j \\ 0 & , \ \textit{otherwise} \end{cases} \tag{11.326}$$

for $j = 1, 2, \dots, K$.

To ensure that a machine is not kept beyond its working age limit (L periods), we keep a record of the age of the current machine. This suggests the following state variable:

$$s_j := \textit{age of the current machine at the beginning of the j-th period} \tag{11.327}$$

Hence we let

$$S = \{1, 2, 3, \dots, L\} \tag{11.328}$$

$$D(j,s) = \begin{cases} \{0,1\} & , \quad s < L \\ \{1\} & , \quad s = L \end{cases} \tag{11.329}$$

$$T(j,s,x) = \begin{cases} 1 & , \quad x = 1 \\ s+1 & , \quad x = 0 \end{cases} \tag{11.330}$$

Let,

$$r(s) := C + c(s) - B(s) \ , \ s \in [1, L] \tag{11.331}$$

and

$f_j(s) :=$ *Total net cost from year j to year K, given that at the beginning of year j the operating machine is s years old.* (11.332)

Then the associated dynamic programming functional equation is as follows:

$$f_j(s) = \begin{cases} r(s) + f_{j+1}(1) & , \quad s = L \\ \min\{r(s) + f_{j+1}(1) \ , \ f_{j+1}(s+1)\} & , \quad 1 \leq s < L \end{cases} \tag{11.333}$$

for $1 \leq j \leq K$ with $f_{K+1}(s) = r(s), s \in [1, L]$.

11.8.1.2 Sequential Decision Model

This model views the problem under consideration as a *renewal process.* So the question here is: How long should a new machine be kept? Hence, let

$x_j :=$ *length of time that the j-th machine is kept.* (11.334)

In this framework, a record is kept of the period at which the new machine is purchased. So, the corresponding state variable is

$s_j :=$ *period at which the j-th machine is purchased.* (11.335)

This entails

$$S = \{1, 2, \ldots, K\} \tag{11.336}$$
$$D(s) = \{1, 2, \ldots, \min\{L, K - s + 1\}\} \tag{11.337}$$
$$T(s, x) = s + x \tag{11.338}$$

The associated dynamic programming functional equation is as follows:

$$f(s) = \min_{\substack{x \leq min\{L, K-s+1\} \\ x \ integer}} \{r(x) + f(s+x)\} \ , \ s = 1, 2, \ldots, K \tag{11.339}$$

with $f(s) = 0, s > K$.

Which version do you prefer?

11.9 Models vs Functional Equations

Since a dynamic programming functional equation embodies all the elements of the dynamic programming model, it is common practice to treat it as the formal model of the problem that is sought to be solved.

Indeed, it is often possible to identify the problem itself from the structure of the dynamic programming functional equation associated with it.

11.9.1 Example: Mystery Problem

Suppose that one fine morning you find on your front lawn a note similar to the one shown in Figure 11.4.

$$f_j(x) = \max_{\substack{y\in\{0,1\}\\ yc_j\le x}} \{ya_j + f_{j+1}(x - yc_j)\} \ , \ x \in X, j = 1,\dots,17 \tag{11.340}$$

$$f_{18}(x) = 0 \ , \ \forall x \in X := \{0,1,2,\dots,43\} \tag{11.341}$$

Figure 11.4: A note found on your lawn

A dynamic programming specialist will immediately recognize this as a dynamic programming functional equation and will quickly identify the problem giving rise to it as the well-known *0-1 knapsack problem.*

But for the purposes of our analysis let us pretend that we do not know this. So, how would we go about reconstructing the original problem?
The first steps in this detection process are straightforward:

· Notation used:
  - x denotes the state variable.
  - j denotes the stage variable.
  - y denotes the decision variable.
  - a_j and c_j are given parameters.

· It seems that the model is a multistage model with $N = 17$ decision stages.

· It seems that the state space is $S = \{0,1,2,\dots,43\}$.

· It seems that the initial state is $s = 43$.

· It seems that (using our standard notation) the sets of feasible decisions are given by:

$$D(j,s) = \begin{cases} \{0,1\} & , \ s \ge c_j \\ \{0\} & , \ s < c_j \end{cases} \tag{11.342}$$

· It seems that (using our standard notation) the transition function is given by

$$T(j,s,x) = s - xc_j \tag{11.343}$$

· It seems that (using our standard notation) the objective function is given by

$$g(s_1, x_1, x_2, \ldots, x_k) = \sum_{j=1}^{k} x_j a_j \tag{11.344}$$

And so, having identified all the elements of the multistage decision model, the conclusion must be that the problem is as follows:

$$\max_{x_1,\ldots,x_k} \sum_{n=1}^{k} a_n x_n \tag{11.345}$$

$$\sum_{n=1}^{k} c_n x_n \leq \sigma \tag{11.346}$$

$$x_n \in \{0,1\} \ , \ n = 1, 2, \ldots, k \tag{11.347}$$

where $k = 17$ and $\sigma = 43$.

Also, the modified problems associated with this multistage decision model are as follows:

$$f_n(s) := \max_{x_n,\ldots,x_k} \sum_{j=n}^{k} a_j x_j \ , \ n = 1, \ldots, k = 17; s \in S = \{0, 1, \ldots, 43\} \tag{11.348}$$

$$\sum_{j=n}^{k} c_j x_j \leq s \tag{11.349}$$

$$x_j \in \{0,1\} \ , \ j = n, n+1, \ldots, k \tag{11.350}$$

This problem is known as the *0-1 knapsack problem* because it is subject to the constraint that no more than one item is allowed to be selected from any given pile. □

With this as background try your hand at the following more taxing detection project.

11.9.2 Example

This time, the note on your front lawn has this message scribbled on it:

$$f(s) = 1 + \min_{\substack{x \ integer \\ 1 \leq x \leq s/2}} f(\max\{x, s - 2x\}) \ , \ s = 2, 3, \ldots, 9876543 \tag{11.351}$$

$$f(0) = f(1) = 0 \tag{11.352}$$

What is the DP story behind it? □

The following example illustrates another important aspect of the *Model vs Functional Equation* issue. Not always is it essential, indeed necessary, to conduct a full-blown formal formulation of the decision making process. Things can often be done informally, with the functional equation playing a key role in such an approach. In this setup, the functional equation is treated as a framework for the construction of the related dynamic programming model rather than the product of this model.

11.9.3 Example: Counterfeit Coin Problem

You are given a collection of k coins and a balance beam. All the coins except the counterfeit have the same weight. The counterfeit is slightly heavier than the other coins. Your task is to determine a weighing scheme that will identify the counterfeit coin in a minimum number of weighings under the *worst-case scenario.*

Note that the balance beam cannot measure the weight of items placed on it. It can only determine which "side" is heavier or whether the two "sides" have the same weight.

How would you go about formulating a dynamic programming functional equation for this problem?

A good starting point would be to consider the following situation: suppose that m coins remain to be inspected. Hence, if we place d coins on each side of the beam, then there are two possible outcomes: either the counterfeit coin is on the beam or it is not on the beam. In the first case d coins are yet to be inspected, in the second, $m - 2d$ coins are to be inspect.

So, if we let the state variable, s, denote the number of coins yet to be inspected, then given the current state and current decision, two new states are possible, either $s' = d$ or $s' = s - 2d$.

Specifically, the new state $s' = d$ will result if the counterfeit coin is on the beam and the state $s' = s - 2d$ will result if the counterfeit coin is not on the beam.

Your task is to determine a weighing strategy that will find the odd coin in a minimum number of weighings under the *worst case scenario* for each weighing. The *worst case scenario* clause simply means that when you place the coins on the scale, *Nature* hides the false coin in the *largest* collection of coins yet to be inspected. There are three collections: two on the scale and one off the scale.

For example, if 35 coins remain to be inspected and you decide to place 10 coins on each side of the scale, then you discover — after the weighing — that the false coin is off the scale. Whereas if you place 16 coins on each side of the scale, you discover — after the weighing — that the odd coin is on the scale. In the first case, $35 - 2(10) = 15$ coins remain to be inspected, in the second, 16 coins remain to be inspected.

What is the optimal weighing policy?

Guessing what the optimal policy is in this case is rather easy. But your task is far more demanding: it is to formulate a dynamic programming model for this problem. So, regard this exercise a dynamic programming modeling exercise, not a puzzle solving exercise.

One way to construct a dynamic programming formulation for this problem is to let

$$x_j := \textit{number of coins placed on each side of the scale at the } j\textit{-th weighting, } j = 1, 2, \dots, k \tag{11.353}$$

This means that you have to record the number of coins left for inspection before decision x_j is made. Thus, let

$$s_j := \textit{number of coins left for inspection before the } j\textit{-th weighting, } j = 2, 3, \dots, k \tag{11.354}$$

Observe that the number of required weighings is unknown, so k is treated as a *decision variable.*

And here is the crucial step: the recognition that the minimal number of weighings depends on the number of coins left for inspection, s_j. Once this is appreciated, it is only natural to let

$$f(s) := \textit{minimal number of weighings required to identify the odd coin if } s \textit{ coins are left for inspection, } s = 0, 1, 2, \dots, N \tag{11.355}$$

where N denotes the total number of coins. Similarly, let

$$f(s, x) := \textit{minimal number of weighings required to identify the false coin if } s \textit{ coins are left for inspection and } x \textit{ coins are placed on each side of the scale in the next weighting,} \tag{11.356}$$

for $s = 2, \dots, N, 1 \leq x \leq s/2$.

Then, clearly,

$$f(s) = \min_{1 \leq x \leq s/2} f(s, x) \ , \ s = 2, \dots, N \tag{11.357}$$

with $f(0) = f(1) = 0$.

So, all that remains is to express $f(s, x)$ in terms of $f(s')$ values pertaining to related values of s'. Hence, the critical question is: what is $f(s, x)$? What is the best we can do if s coins remain to be inspected and we decide to place x coins on each side of the scale?

The answer comes directly from *Nature*: if you place x coins on each side of the scale, the largest collection will contain $\max\{x, s - 2x\}$ coins. I shall therefore hide the false coin in that collection. This means that after the weighing, you'll have exactly $\max\{x, s - 2x\}$ coins to inspect.

Thus,

$$f(s, x) = 1 + f\left(\max\{x, s - 2x\}\right) \tag{11.358}$$

where the 1 represents the "cost" of the next weighing. This means that

$$f(s) = \min_{1\le x\le s/2} \{1 + f(\max\{x, s-2x\})\} \ , \ s = 2,3,\dots,N \tag{11.359}$$

$$= 1 + \min_{1\le x\le s/2} f(\max\{x, s-2x\}) \tag{11.360}$$

with, by definition, $f(0) = f(1) = 0$.

And if you insist on being pedantic, you would re-write this functional equation as follows:

$$f(s) = \begin{cases} 0 & , \quad s = 0,1 \\ 1 + \min\limits_{\substack{x \ integer \\ 1\le x\le s/2}} f(\max\{x, s-2x\}) & , \quad s = 2,3,\dots,N \end{cases} \tag{11.361}$$

Given this functional equation, it is not too difficult to work out that it represents a sequential decision model whose elements are as follows:

$$S = \{1, 2, 3, \dots, N\} \tag{11.362}$$

$$D(s) = \left\{1, 2, \dots, \left\lfloor \frac{s}{2} \right\rfloor \right\} \tag{11.363}$$

$$T(s, x) = \max\{x, s-2x\} \tag{11.364}$$

$$\sigma = N \tag{11.365}$$

$$w(s, x) = 1 \text{ (applied only when } x \ge 1) \tag{11.366}$$

observing that this model has one terminal node, namely $S'' = \{1\}$. $\square$

And this concludes our short detective story.

11.10 Easy Problems

To bring into sharp focus my exposition on the state and how to model it, or more generally, how to model in dynamic programming style, I am going to concentrate more directly on the challenges that are presented by problems that prove "difficult" to formulate and, to illustrate how these challenges can be handled. To set the stage, I formulate an abstract class of problems whose members are relatively "easy" to formulate in terms of a dynamic programming model. This class will serve as the benchmark for identifying those characteristics that render a problem more "difficult" to formulate.

Consider then the following optimization problem:

$$z^* = \operatorname*{opt}_{x\in X} g(x) \ , \ X \subseteq X' = X_1 \times X_2 \times \cdots \times X_k \tag{11.367}$$

$$s.t. \quad c(x) \diamond \gamma \tag{11.368}$$

where

$$g(x) = \bigoplus_{n=1}^{k} w(n, x_n) := w(1, x_1) \oplus w(2, x_2) \oplus \cdots \oplus w(k, x_k) \tag{11.369}$$

$$c(x) = \bigotimes_{n=1}^{k} v(n, x_n) := v(1, x_1) \otimes v(2, x_2) \times \cdots \times \otimes v(k, x_k) \tag{11.370}$$

and $\diamond$ is a binary relation as outlined in *Section 11.4.*

In cases where $g(x)$ is evaluated from right to left and $\oplus$ is Markovian, the state variable will be determined solely by the constraint $c(x) \diamond \gamma$. Therefore, if in addition to this $c(x)$ is evaluated from left to right, and $\otimes$ has a left-identity element ($L_{\otimes}$), the state variable would be defined as follows:

$$s_n := \bigotimes_{j=1}^{n-1} v(j, x_j) := v(1, x_1) \otimes v(2, x_2) \times \cdots \times \otimes v(n-1, x_{n-1}) \tag{11.371}$$

for $1 < n \leq k$, with $s_1 := L_{\otimes}$. Note that this implies that

$$s_{n+1} = T(n, s_n, x_n) := s_n \otimes v(n, x_n) \tag{11.372}$$

This done, all that remains is to work out the appropriate expressions for the decisions sets $D(n, s_n)$. Often this will depend on the specific properties of $\otimes$ and $\diamond$.

In summary then, the problems that are "easy" to formulate, in the sense that the identification of a sound state variable is carried out with relative ease, are characterized by:

· A separable objective function defined by a Markovian binary operation that is evaluated from "right to left".

· A constraint that is defined by a binary operation that is evaluated from "left to right".

The more taxing problems, those that are characterized by objective functions and/or constraints that are not amenable to such a straightforward decomposition, obviously require a more elaborate approach.

I add in passing that — as we shall see in *Chapter 14* — certain problems where the objective function is evaluated from "left to right" by means of a Markovian binary operation, can be handled with relative ease by a *"forward"* dynamic programming formulation.

11.11 Modeling Tips

In this section I outline a "rough guide" to modeling whose objective is to navigate the modeler in the modeling and formulation of so called "difficult" problems.

To begin however, the following disclaimer is in order. The set of instructions that I am about to propose is obviously not intended to be taken as a formula that is designed to ensure certain success in dynamic programming modeling. At this stage it ought to be clear that such a formula can hardly be envisioned. Still, as can be gathered from what we have seen so far, the set of instruction proposed below recapitulates the basic structure of the approach to dynamic programming that I have been outlining here. So, the idea is that following this set of instructions can steer the modeler/analyst in the right direction.

But it can do no more than this because, as I stated in the introduction, the crucial element in the modeling and formulation process would have to come from the modeler who would have to implement this "rough guide" with imagination, with the knowledge that comes from experience and so on.

A DP Modeling Guide

· Relax!
· Think about your problem as a multistage/sequential decision problem.
· Identify the decision variables.
· Identify the state variable.
· Identify the modified problems.
· Identify the conditional problems.
· Derive the functional equation.
· Construct the dynamic programming model.
· Double-check the model and the functional equation.
· Relax!

It can prove helpful to begin the modeling process by first formulating the problem that one seeks to solve as a "standard" optimization problem:

$$z^* = \underset{x\in X}{\operatorname{opt}}\, g(x) \ , \ \ X \subseteq X' = X_1 \times X_2 \times \cdots X_k \tag{11.373}$$

$$s.t. \quad c(x) \diamond \gamma \tag{11.374}$$

This formulation will call for the identification of proper decision variables and the formulation of an appropriate objective function and constraints which in turn may provide some helpful clues for the construction of decision and state variables for the dynamic programming model.

Specifically, formulating g and c as done in *Section 11.10* may enable to establish straightaway whether the problem is "easy" to solve and, if not, how in view of the difficulties encountered should it be decomposed.

It is important to remember, however, dynamic programming's specific outlook on optimization problems which entails that the dynamic programming model may not utilize the decision variables of the "standard " model. In a word, however helpful the "standard" formulation may prove to be, it is important to avoid being biased by it.

That said, let us now examine how our “rough guide” would be applied in the treatment of one of the most prominent problems in Combinatorial Optimization and Operation Research.

11.11.1 Example: Traveling Salesman Problem

A salesman has to visit k cities. Company regulations dictate that he start and finish the tour in his home city and that he visit all the cities on the tour exactly once. Also, he is required to make the shortest possible tour.

Your task is to develop a dynamic programming model for this well-known problem. If you consult the *Operations Research* literature, you'll find the following formulation for the *Traveling Salesman Problem* (TSP):

$$\min_x \sum_{i=1}^{k}\sum_{j=1}^{k} d_{i,j}x_{i,j} \tag{11.375}$$

$$s.t. \quad \sum_{i=1}^{k} x_{i,j} = 1 \ , \ j = 1,2,\dots,k \tag{11.376}$$

$$\sum_{j=1}^{k} x_{i,j} = 1 \ , \ i = 1,2,\dots,k \tag{11.377}$$

$$\sum_{i,j\in C} x_{i,j} \leq |C| - 1 \ , \ \forall C \subset \{1,2,\dots,k\}, |C| > 1 \tag{11.378}$$

$$x_{i,j} \in \{0,1\} \ , \ i,j = 1,2,\dots,k \tag{11.379}$$

where $d_{i,j}$ denotes the direct distance from city i to city j.

The binary decision variables are interpreted as follows:

$$x_{i,j} = \begin{cases} 1 & , \ \textit{from city } i \textit{ he goes directly to city } j. \\ 0 & , \ \textit{from city } i \textit{ he does not go directly to city } j. \end{cases} \tag{11.380}$$

The first two constraints are user-friendly: the first requires that each city be visited no more than once. The second indicates that he must go directly from any one city to only one city.

The third constraint is user-unfriendly. It indicates that no *sub-tours* are allowed and it is thus called a *sub-tour elimination constraint.*

The merit of this formulation is that it is LINEAR. Namely, the model formulates the TSP as a linear integer (binary) programming problem which is of course important because such problems are amenable to treatment (solution) by a variety of integer programming methods.

Another point to note about this formulation of the TSP is its a priori bias towards a linear model. This, however, is not very conducive to a dynamic programming formulation. It should be useful, therefore, to contemplate a simpler formulation, one that is not predisposed towards a linear model.

So consider the following formulation, assuming that there are $k+1$ cities, including the home city:

$$\min_{x_1,\dots,x_k} \left\{ d(0,x_1) + \sum_{j=1}^{k-1} d(x_j, x_{j+1}) + d(x_k, 0) \right\} \tag{11.381}$$

$$\{x_1,\dots,x_k\} = C := \{1,\dots,k\} \tag{11.382}$$

where $d(i,j)$ denotes the direct distance from city i to city j and city 0 denotes the home city.

The decision variables can be interpreted as follows:

$$x_j = \textit{the j-th city on the tour}\ ,\ j = 1,2,\dots,k \tag{11.383}$$

The constraint simply indicates that the sequence of decisions $(x_1,\dots,x_k)$ must be a PERMUTATION of the cities. Note that the implicit last decision is $x_{k+1} = 0$ meaning that the tour ends in the home city 0.

Remark:
Note that the decision variable $x_{i,j}$ in the formulation of the problem in (11.375)-(11.379), is significantly different from the decision variable, x_j in the formulation of the problem in (11.383). This is not unusual. Indeed, the decision variable in a dynamic programming formulation of a problem is often different from the decision variable in a non-dynamic programming formulation of the same problem. What is more, even two dynamic programming formulations of the same problem may feature significantly different decision variables.

In this discussion our frame of reference is x which denotes the decision variable of an optimization problem as such. So, the formal definition of x may vary with various formulations of the same problem. The relevant definition pertaining to a particular formulation should be obvious from the context of the discussion.

In short, don't be surprised to discover that the definition of the decision variable in the dynamic programming formulation of the TSP will be different from the above definition of $x_{i,j}$.

Given these preliminaries, let us now examine how our "rough guide" would be put to work in the modeling of the TSP.

Step 1: Relax!
Step 2: Thinks about your problem in terms of a multistage/sequential decision problem
That is, conceive of the tour as proceeding from one city to another in a sequential manner.

Step 3: Identify the decision variables.
Focusing on the "linear programming free" formulation, it is natural to define the decision variables as follows:

$$x_j := \textit{j-th city on the tour}, j = 1, 2, \ldots, k \tag{11.384}$$

Step 4: Identify the state variable.
To ensure that each city is not visited more than once and that all the cities are visited, a record must be kept of the set of cities already visited, or the set of cities yet to be visited. So, let

$$\begin{aligned} s_j := &\textit{ set of cities yet to be visited before the decision } x_j \\ &\textit{ is made}, j = 1, 2, \ldots, k \end{aligned} \tag{11.385}$$

This means that $s_j = C \backslash \{x_1, x_2, \ldots, x_{j-1}\}, j > 1$, with $s_1 = C$.
Step 5: Identify the modified problems.
To this end, you assume that the decision making process is at a point where the j-th decision is to be made. The problem then is this:

$$\begin{aligned} f(s) := &\textit{ length of the shortest tour through the remaining} \\ &\textit{ cities in } s \subseteq C. \end{aligned} \tag{11.386}$$

Step 6: Identify the conditional problems.
To to do this, assume that the tour is at state s and the decision is to go to city x. The question is then: what is the best decision under these conditions with regard to the remaining cities to be visited? So, consider this:

$$\begin{aligned} f(s,x) := &\textit{ length of the shortest tour through the cities in } s \textit{ given} \\ &\textit{ that } x \textit{ is the next city to be visited}, \ s \subseteq C, x \in s. \end{aligned} \tag{11.387}$$

Step 7: Derive the functional equation.
To accomplish this task you need to express $f(s,x)$ in terms of $f(s')$ where s' is the state resulting from (s,x), namely $s' = s \setminus \{x\}$. So, by definition, $f(s,x)$ is the length of a tour that starts at the current city, goes to city x and then proceeds along the shortest sub-tour through the remaining cities. That is,

$$f(s,x) = d(c,x) + f(s \setminus \{x\}) \tag{11.388}$$

where c denotes the current city. This in turn implies that

$$f(s) = \min_{x \in s} \{d(c,x) + f(s \setminus \{x\})\} \tag{11.389}$$

So far so good except for one thing: there is no record of the current city c. The state s is defined as the set of cities yet to be visited, but what element of the other cities in C is the current city?

So what should be done next?!

Relax!

Simply redefine the state variable, making sure that it specifies the current city. Hence,

Step $4'$**:** Identify the state variable.
To this end you need to keep a record of the cities yet to be visited and that of the current city. So, let

$$s = (c, V) \ , \ c \in C, V \subseteq C \setminus \{c\} \tag{11.390}$$

where c denotes the current city and V denotes the set of cities yet to be visited. The initial state is $\sigma = (0, C)$.

Step $5'$**:** Identify the modified problems.
So, again you assume that the decision making process is at a point where the next decision is about to be made. The problem is then this:

$$f(c,V) := \textit{length of the shortest tour through the cities in } V \textit{ given that the current city being visited is } c. \tag{11.391}$$

Step $6'$**:** Identify the conditional problems.
So, as the tour is at state $s = (c, V)$ and the decision is to go to city x, the question is what is best under these conditions with regard to the remaining cities to be visited? Hence:

$$f(c,V,x) := \textit{length of the shortest tour through the cities in } V \textit{ given that the tour is now in city } c \notin V \textit{ and the next visit is to city } x \in V. \tag{11.392}$$

As a matter of fact, you should have written $f((c, V), x)$. But it is not essential to be too meticulous in this regard.

Step $7'$**:** Derive the functional equation.
To do this express $f(c, V, x)$ in terms of $f(s')$ where s' is the state resulting from (c, V, x), namely $s' = s \setminus \{x\}$. So, by definition, $f(c, V, x)$ is the length of a tour that starts at the current city c, goes to city x and then proceeds along the shortest sub-tour of the remaining cities. That is,

$$f(c, V, x) = d(c, x) + f(x, V \setminus \{x\}) \tag{11.393}$$

This in turn implies that

$$f(c, V) = \min_{x \in V} \{d(c, x) + f(x, V \setminus \{x\})\} \tag{11.394}$$

for $c \in C, \ V \subseteq C \setminus \{x\}$ and $(c, V) = (0, C)$, with $f(c, \varnothing) = d(c, 0), c \in C$. The length of the shortest tour is $f(\sigma)$, where $\sigma = (0, C)$.

Well done!

Step 8: Construct the dynamic programming model.
Given the draft copy of the functional equation, set up a sequential decision model comprising the following ingredients, using our standard notation:

$$\sigma = (0, C) \tag{11.395}$$
$$S' = \{\sigma\} \cup \{(c, V) : c \in C, V \subseteq C \setminus \{x\} \tag{11.396}$$
$$S'' = \{(0, \varnothing)\} \tag{11.397}$$
$$S = S' \cup S'' \tag{11.398}$$
$$D(c, V) = \begin{cases} V & , \quad V \neq \varnothing \\ \{0\} & , \quad V = \varnothing \end{cases} \tag{11.399}$$
$$T(c, V, x) = (x, V \setminus \{x\}) \tag{11.400}$$
$$w(c, V, x) = d(c, x) \tag{11.401}$$
$$\oplus = + \tag{11.402}$$

This means that the modified problems are as follows:

$$f(c, V) = \min_{x_1, \dots, x_{m+1}} \sum_{j=1}^{m+1} d(c_j, x_j) \ , \ (c, V) \in S', m = |V| \tag{11.403}$$
$$x_j \in D(s_j) \ , \ j = 1, 2, \dots, m+1 \tag{11.404}$$
$$s_1 = (c, V) \tag{11.405}$$
$$s_{j+1} = T(s_j, x_j) \ , \ j = 1, 2, \dots, m+1 \tag{11.406}$$
$$s_{m+2} = (0, \varnothing) \tag{11.407}$$

The object of interest is $f(\sigma) = f(0, C)$.
You can therefore be more specific and rewrite the constraints (11.404)-(11.407) as follows:

$$V_1 = V \tag{11.408}$$
$$c_1 = c \tag{11.409}$$
$$x_j \in V_j \ , \ j = 1, 2, \dots, m \tag{11.410}$$
$$c_{j+1} = x_j \ , \ j = 1, 2, \dots, m \tag{11.411}$$
$$V_{j+1} = V_j \setminus \{x_j\} \ , \ j = 1, 2, \dots, m \tag{11.412}$$
$$x_{m+1} = 0 \tag{11.413}$$

Step 9: Double-check the model and the functional equation.

Of course, with experience come the skills and the confidence. So eventually you will not have to actively consult the "guide" to direct you in the formulation process. □

Still, with our "rough guide" as background, I shall now proceed to illustrate (in a less elaborate fashion) the formulation of a number of problem that prove "difficult" to formulate. My first illustration is that of a NON-SERIAL dynamic programming model.

In contrast to the models discussed thus far, here $T(j, s, d)$ is a *set* of states rather then a single state. That is, each decision generates a *family* of sub-problems, rather than one single subproblem. This means that after a decision is made the process *branches out* into a number of sub-processes.

11.11.2 Example: Chained Matrix Product

Consider a sequence of k matrices, $M^{(1)}, M^{(2)}, \dots, M^{(k)}$ of compatible sizes (for matrix multiplication). The objective is to determine the matrix product

$$p := M^{(1)}M^{(2)} \cdots M^{(k)} \tag{11.414}$$

Note that because matrix product is an *associative* operation, no parentheses are required to stipulate the order in which it is conducted. However, insofar as *computational efficiency* is concerned, this is a matter of paramount importance. For instance, consider the case where $k = 4$ and the sizes of the matrices are as follows:

A	B	C	D
13×5	5×89	89×3	3×34

Then the number of *scalar products* required to compute p — depending on the parentheses used — is as follows:

A(B(CD))	A((BC)D)	(A(BC))D	(AB)(CD)	((AB)C)D
$26,418$	$4,055$	$2,856$	$54,201$	$10,582$

As this small example vividly illustrates, significant savings can be achieved as a result of optimizing the manner in which the parentheses are executed in chained matrix products. The question is then: what is the best policy?

I shall now demonstrate how this important problem would be tackled by dynamic programming where it would be formulated as a sequential decision process where the transition function generates *sets of states* rather then singletons.

Let then

$$f(i,j) := \textit{Minimum number of scalar products required for multiplying} \\ \textit{the sub-chain } M^{(i)} \cdots M^{(j)} \ , \ 1 \leq i < j \leq k \tag{11.415}$$

If we break up this sub-chain into two sub-sub-chains, say $M^{(i)} \cdots M^{(d)}$ and $M^{(d+1)} \cdots M^{(j)}$, then the minimum total number of scalar products will be

$$f(i,j;d) = f(i,d) + f(d+1,j) + r(i)c(d)c(j) \tag{11.416}$$

where $r(i)$ denotes the number of matrix rows $M^{(j)}$ and $c(j)$ denotes the number of matrix columns. Thus, the optimal value of d is that which minimizes the right-hand side of (11.416). Hence,

$$f(i,j) = \min_{i \leq d < j} \left\{ f(i,d) + f(d+1,j) + r(i)c(d)c(j) \right\} \tag{11.417}$$

for $1 \leq i < j \leq k$ with $f(i,i) = 0, \forall i$.

Note that it follows from (11.416) that two new states are generated by a decision d, namely (i,d) and $(d+1,j)$. Formally, in this case we can set $T(i,j,d) = \{(i,d),(d+1,j)\}$, where the set of feasible decisions associated with state (i,j) is $D(i,j) := \{i, i+1, \dots, j-1\}$.

To ensure that the $f(\cdot)$ values needed on the right-hand side of the functional equation (11.417) to determine the value of $f(i,j)$ are available, the equation is solved for (i,j) pairs in an order determined by their $j-i$ values, recalling that with $f(i,i) = 0, \forall i$.

If the $f(i,j)$ values are organized in a table, then commencing with the elements of the main diagonal being set to 0, such an order will fill the upper-diagonal part of the table one diagonal at a time. □

By far the most important class of non-serial sequential decision processes is that of *Stochastic Processes*. These processes are characterized by a transition law according to which the states are *random variables* whose conditional probability functions are parameterized by the (current) stage, state and decision variables. In this case, each realization of the new state is observed with a given probability depending on the current stage, state and decision.

11.11.3 Example: Counterfeit Coin Problem

In contrast to the WORST-CASE ANALYSIS approach set out in §*11.9.3,* let us examine here an approach where the location of the false coin is determined PROBABILISTICALLY.

As before, let the state variable s denote the number of coins left for inspection and let the decision variable d denote the number of coins placed on each side of the beam.

Now, let $Pr[s' = n|s,d]$ denote the (conditional) probability that the new state s' is equal to n given that the current state is s and the current decision is d. If we assume that *Nature* is "impartial" — hence the conditional probabilities are *uniform* — we have $Pr[s' = d|s,d] = 2d/s$ and $Pr[s' = s-2d|s,d] = (s-2d)/s$ in the case where $d \neq s-2d$. If $d = s-2d$, then clearly $Pr[s' = d|s,d] = 1$.

If the objective is to minimize the *expected value* of the total number of weighings, we let

$$f(s) := \textit{minimum expected number of weighings required to identify the false coin if } s \textit{ coins are left for inspection, } s = 1,2,\dots,k \tag{11.418}$$

The associated dynamic programming functional equation is as follows

$$f(s) = \min_{\substack{1\le d\le s/2 \\ d \ integer}} \left\{ 1 + \frac{d}{s} f(d) + \frac{s-2d}{s} f(s-2d) \right\} \ , \ s > 1 \tag{11.419}$$

with $f(1) = 0$.

Observe that this formulation assumes that the next weighing is only of coins that were in the batch containing the false coin in the previous trial. As shown in Sniedovich [2003], it is sometime advantageous to "borrow" an additional genuine coin identified in previous trials.

This counter-intuitive "anomaly" is manifested in the fact that $f(s)$ is not monotone with s. In fact, solving the functional equation (11.419) we obtain

s	1	2	3	4	5	6	7	8	9	10
$f(s)$	0	1	1	$\frac{3}{2}$	$\frac{8}{5}$	2	$\frac{13}{7}$	2	2	$\frac{11}{5}$

Note that counter-intuitively $f(7) < f(6)$.

There is nothing mystifying about this result. It is a consequence of the fact that the decision variables are *discrete.*

With $n = 6$ coins to check, there are two optimal solutions, namely 2 or 3 coins can be placed on each side of the scale thus leaving 2 or 0 coins, respectively, off the scale. Suppose that 2 coins are placed on each side of the scale. Then, with probability one the next trial will feature 2 coins. This means that in this case the search will be completed in exactly 2 weighings regardless of where *Nature* places the false coin.

If 3 coins are placed on each side of the scale, then with probability one the next trial will feature 3 coins. This means that in this case, the search is completed in 2 weighings regardless of where *Nature* places the false coin. Thus, $f(6) = 2$.

With $n = 7$, 3 coins are placed on each side of the scale, in which case one coin is left off the scale. Hence, the probability is $1/7$ that the false coin will be identified in *the first trial!* This "opportunity" is not available in the case where $n = 6$.

Namely, with 7 being a *prime* number: indivisible by 2 and by 3, it enables identifying the false coin in one trial.

This "anomaly" is propagated via the functional equation to larger values of s. □

The next example is a variation on the conventional knapsack problem. It is a good illustration of how minor details can be important in dynamic programming modeling. I should add, though, that dynamic programming is not unique in this respect. Integer programming modeling exhibits similar characteristics.

11.11.4 Dual Knapsack Problem

Consider the following simple knapsack-like problem:

$$v^* := \min_{(x_1,\dots,x_k)} \sum_{n=1}^{k} v_n x_n \tag{11.420}$$

$$\sum_{n=1}^{k} w_n x_n \geq W \tag{11.421}$$

$$x_n \in \{0,1,2,3,\dots\} \ , \ n \in \mathcal{K} := \{1,2,\dots,k\} \tag{11.422}$$

where all the parameters are positive integers and x_n denotes the number of items of type n placed in the knapsack.

In words, the objective is to minimize the total value of the items in the knapsack, subject to the requirement that the total weight be equal to or greater than W.

The modeling task here is not too taxing, still ... my experience has shown that *students* often have great difficulties in modeling the "$\geq$" form of the total weight constraint.

To this I might add parenthetically that students have similar difficulties in modeling the "if then" constraints when required to give the standard knapsack problem an *integer programming* formulation. For instance, the constraint: if more than 3 items of type 6 are put in the knapsack then, no more than 4 items of type 2 are permitted.

And some even have difficulties coping with the "=" knapsack constraint, although they are familiar with the *Big M* method that is used extensively in the formulation of *linear programming* problems.

Back to the dual knapsack problem.

Consider the family of problems obtained by regarding W as an instance of a parameter, call it s. The parametric problem is then:

$$f(s) := \min_{(x_1,\dots,x_k)} \sum_{n=1}^{k} v_n x_n \quad , \ s \in \{0,1,2,\dots,W\} \tag{11.423}$$

$$\sum_{n=1}^{k} w_n x_n \geq s \tag{11.424}$$

$$x_n \in \{0,1,2,3,\dots\} \ , \ n \in \mathcal{K} := \{1,2,\dots,k\} \tag{11.425}$$

Based on our extensive study of the standard (unbounded) version of the problem, we can confidently propose that the functional equation in this case would be as follows:

$$f(s) = \min_{x \in \mathcal{K}} \{v_x + f(s - w_x)\} \tag{11.426}$$

observing that since exceeding the total weight s is not only permitted, but in fact required, the minimization on the right hand side of (11.426) is over the entire collection of piles $\mathcal{K}$. Note that here the decision variable x denotes the type of item selected, not the number of items of a given type selected which is used in (11.420)-(11.422).

The question is therefore: what should be the value of $f(s)$ for $s < 0$, given that our object of interest is the value of $f(W)$ defined by (11.423)-(11.425) with $s = W$?

The answer is of course: $f(s) = 0, \forall s < 0$.

But this gives rise to another question: what is the smallest (negative) value of s that should be considered?

Upon reflection we conclude that there is no need to consider values of s below $-w^*$, where $w^* = \max\{w_1, \dots, w_k\}$. However, to simplify the notation we shall let the smallest value be $-W$.

So formally, the sequential decision model for the dual knapsack problem is as follows:

$$S = \{-W, \dots, W\} \tag{11.427}$$

$$D(s) = \begin{cases} \varnothing & , \quad s \leq 0 \\ \mathcal{K} & , \quad s > 0 \end{cases} \tag{11.428}$$

$$T(s,x) = s - w_x \tag{11.429}$$

$$w(s,x) = v_x \tag{11.430}$$

$$\oplus = + \tag{11.431}$$

$$\text{opt} = \min \tag{11.432}$$

observing that the set of terminal states in $S'' = \{-W, -W+1, \dots, 0\}$, and $S' = \{1, 2, \dots, W\}$. The functional equation is as follows:

$$f(s) = \min_{x \in \mathcal{K}} \{v_x + f(s - v_x)\} \ , \ s \in S' \tag{11.433}$$

with $f(s) = 0, \forall s \in S''$. □

Finally, the following example is dedicated to students who may have wrestled with it in my DP exams over the past twenty years or so.

11.11.5 Equality Constrained Knapsack Problem

Consider the following simple knapsack problem:

$$v^* := \max_{(x_1,\dots,x_k)} \sum_{n=1}^{k} v_n x_n \tag{11.434}$$

$$\sum_{n=1}^{k} w_n x_n = W \tag{11.435}$$

$$x_n \in \{0, 1, 2, 3, \dots\} \ , \ n \in \mathcal{K} := \{1, 2, \dots, k\} \tag{11.436}$$

where all the parameters are positive integers and x_n denotes the number of items of type n placed in the knapsack. Observe that the problem is *infeasible* if $W < w^{\circ} := \min \{w_1, \dots, k\}$.

The associated modified problems are as follows:

$$f(s) := \max_{(x_1,\dots,x_k)} \sum_{n=1}^{k} v_n x_n \quad , \ s \in \{w^{\circ}, \dots, W\} \tag{11.437}$$

$$\sum_{n=1}^{k} w_n x_n = s \tag{11.438}$$

$$x_n \in \{0,1,2,3,\dots\} \ , \ n \in \mathcal{K} := \{1,2,\dots,k\} \tag{11.439}$$

In view of the discussion above, consider the following sequential decision model:

$$S = \{w^{\circ}, \dots, W\} \tag{11.440}$$

$$D(s) = \{x \in \mathcal{K} : w_x \leq s\} \tag{11.441}$$

$$T(s,x) = s - w_x \tag{11.442}$$

$$w(s,x) = v_x \tag{11.443}$$

$$\oplus = + \tag{11.444}$$

$$\text{opt} = \max \tag{11.445}$$

The dynamic programming functional equation deriving from this model is as follows:

$$f(s) = \max_{x \in \mathcal{K}} \{v_x + f(s - v_x)\} \ , \ s \in S \tag{11.446}$$

with $f(0) = 0$ and $f(s) = -\infty, \forall s \in [1, 2, \dots, w^{\circ} - 1]$.

Note that here the decision variable denotes the type of item selected, not the number of item selected from a certain pile, as in (11.434)-(11.436). Also observe that if $f(W) = -\infty$, the implication is that the problem defined by (11.420)-(11.422) is infeasible.

Remark: Compare the above sequential decision model with the multistage decision model formulated for this type of problem in *Example 11.5.1.* □

11.12 Concluding Remarks

To conclude this chapter I want to reiterate my position on the approach that I described here for tracking down the state variable and framing it mathematically. I do not propose that this approach be implemented religiously in every dynamic programming investigation as a means for obtaining a dynamic programming model and functional equation. Much less is it my intention to imply that experienced dynamic programming practitioners or scholars regularly engage in this kind of retrospective analysis, or perhaps in other similar analyses, to deduce the dynamic programming model.

My primary objective in proposing this approach is to suggest a way for identifying the state and constructing it mathematically, when this is not otherwise evident. Because, as I pointed out, identifying the state and constructing it mathematically constitutes a pivotal move in the formulation of an optimization problem in dynamic programming style, and because this often proves a difficult task to accomplish, particularly for novices.

That said, I wish to add another disclaimer. It is equally important to realize that I do not envisage the proposed approach to offer an infallible recipe that is certain to yield the state whenever it is applied in any specific case of *Problem P.* After all, what this approach offers is some tips, some general guidelines, and I hope, insight into how to proceed in the search for the state. But, the crucial input must come, from the modeler/analyst. The modeler must be able to decide, while working his/her way back from the conditional problem, what items of information are vital for determining the feasibility and optimality of the remaining decisions; and in so doing he/she must be able to work out the formulation of the state. Likewise, the modeler must be able to decide on the formulation that is best suited for a particular situation, should it turn out that more than one valid state formulation is possible, and so on. Of course for inspiration and advice on how to approach dynamic programming modeling one would have to read the master himself (Bellman 1957, p. 82):

> We have purposely left the description a little vague, since it is the spirit of the approach that is significant rather than the letter of some rigid formulation. It is extremely important to realize that one can neither axiomatize mathematical formulation nor legislate away ingenuity. In some problems the state variables and the transformations are forced on us; in others there is a choice in this matters and the analytic solution stands or falls upon this choice; in still others, the state variables and sometimes the transformations must be artificially constructed. Experience alone, combined with often trial and error, will yield suitable formulations of involved processes.

Although this statement was made more than fifty years ago, it remains as poignant and valid today; not as a sign of lack of progress but rather as testimony to the skills required in dynamic programming modeling.

11.13 Summary

I delineated a general approach for tracking down the state variable and establishing its mathematical form. The main features of this approach are postulating a conditioning sequence of decisions as a starting point and singling out the specific information about it that is essential for determining the optimality of the remaining decisions.

This type of retrospective analysis is based on an understanding of the state as bearing the imprint of two sources, the solution set X and the objective function q. Thus, an examination of the impact of these sources in this order unlocks the structure of the state.

I also demonstrated modeling and formulation in action through the analysis and formulation of a number of important problems.

For other approaches to the identification of the state variable in dynamic programming I refer the readers to Elmaghraby [1970], Hinderer [1970] and Pollock and Smith [1985], Chou et al. [2001].

12

Parametric Schemes

12.1 Introduction

In this chapter I take up again the state variable, addressing it here from a slightly different angle. Whereas in the previous chapter my main concern was to illustrate how to track down the state, here I focus on the question of how to fend off the adverse effects of a non-Markovian objective function and global constraints on it. As we have seen, such an objective function and/or constraints tend to enlarge the state space and thereby trigger the *Curse of Dimensionality.* So, in this chapter I delineate three schemes whose express aim is to avert this ill. The distinctive characteristic of these schemes is that they identify a structural feature in the dimensionality-prone problem of interest — the target problem — which allows tackling it through a surrogate problem, namely a simplified version of the former, which involves an exogenous *parameter.*

The underlying thesis here is that for a certain value of the parameter in question, the optimal solution obtained for the surrogate parametric problem is also optimal for the target problem. The resulting solution procedure consists of an algorithm which searches for the optimal solution for the target problem through an iterative solution of the parametric problem for various values of the exogenous parameter.

These parametric schemes (hybrid algorithms) amount to a collaborative effort between dynamic programming and

· Fractional programming (*Appendix B*).

· Composite concave programming (*Appendix C*).

· Lagrange multiplier methods.

The first two are designed to handle problems with non-Markovian objective functions, and the third is designed for problems subject to global constraints.

12.2 Background and Motivation

Let us begin with a statement of the problem that is of concern to us in this connection:

Problem $P(s), s \in S_1$:

$$p(s) := \operatorname*{opt}_{(x_1,\dots,x_N)} \ g(x_1,\dots,x_N) \tag{12.1}$$

$$x_n \in D(n,s_n)\ ,\ 1 \leq n \leq N \tag{12.2}$$

$$s_1 = s \tag{12.3}$$

$$s_{n+1} = T(n,s_n,x_n)\ ,\ 1 \leq n \leq N \tag{12.4}$$

Let $X(s)$ and $X^*(s)$ denote the set of feasible solutions and the set of optimal solutions to *Problem* $P(s)$ respectively.

Unless indicated to the contrary, all the computational examples assume that N is finite and that the state variable is induced by the global constraint

$$\sum_{n=1}^{N} x_n = c \tag{12.5}$$

and the local constraints

$$x_n \in \{0,1,2,\dots,c\} \ , \ 1 \leq n \leq N \tag{12.6}$$

where c is a given positive integer. Hence, we can set

$$S = \{0,1,\dots,c\} \tag{12.7}$$

$$S_1 = \{c\} \tag{12.8}$$

$$D(n,s) = \begin{cases} \{0,1,\dots,s\} & , \quad 1 \leq n < N \\ \{s\} & , \quad n = N \end{cases} \ , \quad s \in S \tag{12.9}$$

$$T(n,s,x) = s - x \ , \ 1 \leq n \leq N, s \in S, x \in D(n,s) \tag{12.10}$$

I pointed out in *Chapter 11* that difficulties will arise should it turn out that the objective function g is not Markovian. Or, to put it somewhat more accurately, should it turn out that a Markovian decomposition scheme for the objective function in question is not readily available. As it is extremely important to be clear on where precisely does the trouble lie here, let us backtrack to previous discussions to recall the role of a Markovian decomposition scheme in the formulation of a dynamic programming functional equation. Prior to this we need to recall the definition of a decomposition scheme.

A decomposition scheme in the framework of *Problem* $P(s)$ is a collection $(\rho, \{g_n : 1 \leq n \leq N\})$ such that:

- g_n is a real-valued function on $S \times \mathbb{D}^{N-n+1}$.
- $g_1(s,z) = g(s,z)$ for all $s \in S_1$, $z \in X(s)$.
- ρ is a real-valued function on $\mathbb{N} \times S \times \mathbb{D} \times \mathbb{R}$.
- $g_n(s,x,z) = \rho(n,s,x,g_{n+1}(T(n,s,x)))$ for all $1 \leq n < N$, $s \in S_n$, $x \in D(n,s)$ and $z \in X(n+1,T(n,s,x))$, where $X(n,s)$ denotes the set of all sequences $(x_n, x_{n+1}, \dots, x_N)$ such that

$$x_m \in D(m,s_m) \ , \ n \leq m \leq N \tag{12.11}$$

$$s_n = s \tag{12.12}$$

$$s_{m+1} = T(m,s_m,x_m) \ , \ n \leq m \leq N \tag{12.13}$$

  and $\mathbb{D}$ denotes the decision space of the multistage decision model.

If such a scheme exists, then the objective function g is said to be SEPERABLE. If no such scheme exists, then g is rendered NON-SEPARABLE. The following functions exemplify the latter case.

12.2.1 Example

Consider this very important generic function

$$g(s, x_1, x_2, \dots, x_N) = \frac{\displaystyle\sum_{n=1}^{N} w_n(s_n, x_n)}{\displaystyle\sum_{n=1}^{N} v_n(s_n, x_n)} \tag{12.14}$$

where $\{w_n\}$ and $\{v_n\}$ are real-valued functions on $S \times \mathbb{D}$. $\square$

12.2.2 Example

This is a "variance-like" function:

$$g(s_1, x_1, x_2, \dots, x_N) = \frac{1}{N} \sum_{n=1}^{N} \left[w_n(s_n, x_n) - \frac{1}{N} \sum_{k=1}^{N} w_k(s_k, x_k) \right]^2 \tag{12.15}$$

where $\{w_j\}$ are real-valued functions on $S \times \mathbb{D}$. $\square$

12.2.3 Example

And how about this intricate function:

$$g(s, x_1, x_2, \dots, x_N) = \sum_{n=1}^{N} w_n(s_n, x_n) + \beta \left[\sum_{n=1}^{N} v_n(s_n, x_n) \right]^{1/2} \tag{12.16}$$

where $\beta > 0$ and $\{w_n\}$ and $\{v_n\}$ are real-valued functions on $S \times \mathbb{D}$. $\square$

It should be noted that these objective functions are interesting not only from a theoretical point of view, but also because they arise in real world situations in economics, management, engineering, etc. Therefore, providing a viable solution scheme for them has important practical consequences.

Let us now remind ourselves of the property that gives a decomposition scheme its *Markovian* character. Recall then that the two factors that are at the center of the definition of a Markovian decomposition scheme are the family of *modified problems*

Problem $P(n, s), 1 \leq N, s \in S_n$:

$$f_n(s) := \underset{(x_n, \dots, x_N)}{\text{opt}} \; g_n(s, x_n, x_{n+1}, \dots, x_N) \tag{12.17}$$

subject to (12.11)-(12.13), and the family of *conditional problems*

Problem $P(n, s, x), 1 \leq N, s \in S_n, x \in D(n, s)$:

$$f_n(s, x) := \underset{(x_{n+1}, \dots, x_N)}{\text{opt}} \; g_n(s, x, x_{n+1}, \dots, x_N) \tag{12.18}$$

subject to (12.11)-(12.13) and

$$x_j \in D(j, s_j) \ , \ n+1 \leq j \leq N \tag{12.19}$$

Let $X^*(n, s)$ denote the set of optimal solutions to *Problem* $P(n, s)$ and let $X^*(n, s, x)$ denote the set of optimal solutions to *Problem* $P(n, s, x)$. Then, a decomposition scheme is said to be *Markovian* if

$$X^*(n, s, x) = X^*(n+1, T(n, s, x)) \tag{12.20}$$

for all $1 \leq n < N, s \in S_n$ and $x \in D(n, s)$.

In words, a decomposition scheme is Markovian if for any triplet (n, s, x) such that $1 \leq n \leq N$, $s \in S_n$ and $x \in D(n, s)$, *Problem* $P(n, s, x)$ and *Problem* $P(n+1, T(n, s, x))$ have the same set of optimal solutions.

As we already know, the most prevalent Markovian decomposition scheme is the one associated with additive objective functions of the form

$$g(s_1, x_1, \dots, x_N) = \sum_{n=1}^{N} q_n(x_n, s_n) \tag{12.21}$$

in which case

$$g_n(s_n, x_n, x_{n+1}, \dots, x_N) = \sum_{j=n}^{N} q_j(s_j, x_j) \tag{12.22}$$

and

$$\rho(n, s, x, a) = q_n(s, x) + a \ , \ a \in \mathbb{R} \tag{12.23}$$

That said, let us be clear on the precise meaning of the attribute non-separable and hence non-Markovian.

It is important to appreciate that when an objective function is dubbed non-separable and therefore non-Markovian, we do not mean this in absolute terms. The essential point here is that when we ascribe these properties to an objective function we do this with respect to a *specific formulation* of the problem in question.

So, while an objective function may not possess a Markovian decomposition scheme in terms of a specific formulation, and would thus be rendered non-seperable, the very same objective function would possess a Markovian decomposition scheme and would readily be rendered Markovian and separable in terms of a *different formulation.* What is more, we have the assurance (*Section 4.4*) that obtaining a Markovian formulation is possible *as a matter of principle.* The trouble is, however, that in the case of problems possessing objective functions such as those in *Example 12.1.1* – *Example 12.1.3*, this is accomplished at the expense of a substantial expansion of their state space, thus exposing the resulting functional equation to the *Curse of Dimensionality*.

And to illustrate, let us examine how this is manifested in the case of the ratio-type objective function defined by (12.14).

To continue with *Example 12.1.1,* note that reformulating (12.14) requires an expansion of the state variable which will yield the following:

$$s'_n = (s_n, a_n, b_n) \tag{12.24}$$

where

$$a_n := \sum_{j=1}^{n-1} w_j(s_k, x_j) \ , \ 1 < n \leq N \tag{12.25}$$

and

$$b_n := \sum_{j=1}^{n-1} v_j(s_j, x_j) \ , \ 1 < n \leq N \tag{12.26}$$

with $a_1 := b_1 := 0$.

Since by construction $a_{n+1} = a_n + w_n(s_n, x_n)$ and $b_{n+1} = b_n + v_n(s_n, x_n)$, the resulting transition function will have the form

$$T'(n, s', x) = (T(n, s, x), a + w_n(s, x), b + v_n(s, x)) \ , \ s' = (s, a, b) \tag{12.27}$$

with T denoting the transition function of *Problem* $P(s)$. Now, because by construction,

$$g(s', x_1, x_2, \dots, x_N) = \frac{a_N + w_N(s_N, x_N)}{b_N + v_N(s_N, x_N)} \ , \ s_N = (s_N, a_N, b_N) \tag{12.28}$$

we obtain the degenerate decomposition scheme

$$g_n(s'_n, x_n, x_{n+1}, \dots, x_N) = \frac{a_N + w_N(s_N, x_N)}{b_N + v_N(s_N, x_N)} \tag{12.29}$$

$$\rho(n, s', x, r) = r \ , \ r \in \mathbb{R} \tag{12.30}$$

where $s'_N = (s_N, a_N, b_N)$.

Consequently, the resulting dynamic programming functional equation will have the following form:

$$f_n(s, a, b) = \operatorname*{opt}_{x \in D(n,s)} f_{n+1}(T(n, s, x)\, , \, a + w_n(s, x)\, , \, b + v_n(s, x)) \tag{12.31}$$

for $1 \leq n \leq N$, $(s, a, b) \in S'_n$ and

$$f_{N+1}(s, a, b) := \frac{a}{b} \ , \ (s, a, b) \in S'_{N+1} \tag{12.32}$$

where S'_n is the feasible subset of the expanded state space associated with stage n. The point to observe then is that the sets $(S'_n : 1 \leq n \leq N + 1)$ can be exceedingly large, which of course means dimensionality. □

So, when one reflects on this difficulty it seems that the most sensible thing to do with regard to a dimensionality prone problem is to avoid tackling it with a *standard* dynamic programming solution plan and explore instead other avenues of approach.

One possibility that immediately springs to mind is to capitalize on those traits of the problem that would enable bringing in other optimization methods to assist dynamic programming where it comes to grief. We shall see in the following sections how pursuing this line leads to solution strategies based on a collaboration between dynamic programming and fractional programming, dynamic programming and c-programming, and dynamic programming and Lagrange multiplier techniques.

12.3 Fractional Programming Scheme

The rationale behind this scheme is very simple indeed. It argues that a problem with an objective function of the form

$$g(s, x_1, x_2, \ldots, x_N) = \frac{\displaystyle\sum_{n=1}^{N} w_n(s_n, x_n)}{\displaystyle\sum_{n=1}^{N} v_n(s_n, x_n)} \tag{12.33}$$

should be amenable to the solution techniques of *fractional programming*. For as shown in *Appendix B,* this is precisely the aim of fractional programming — optimizing ratio-type functions.

Recall then that Dinkelbach's method proposes to do this indirectly, that is, by solving a parametric problem with the following objective function:

$$g(s_1, x_1, x_2, \ldots, x_N; \lambda) := \sum_{n=1}^{N} w_n(s_n, x_n) - \lambda \sum_{n=1}^{N} v_n(s_n, x_n) \tag{12.34}$$

$$= \sum_{n=1}^{N} [w_n(s_n, x_n) - \lambda v_n(s_n, x_n)] \tag{12.35}$$

$$= \sum_{n=1}^{N} q_n(s_n, x_n; \lambda) \tag{12.36}$$

where

$$q_n(s_n, x_n; \lambda) := w_n(s_n, x_n) - \lambda v_n(s_n, x_n) \ , \ \lambda \in \mathbb{R} \tag{12.37}$$

Translating this proposition to our case would mean that the solution of the target problem would be sought through the solution of the parametric problem:

Problem $P(s;\lambda), s \in S, \lambda \in \mathbb{R}$:

$$f(s;\lambda) := \max_{(x_1,\dots,x_N)} \sum_{n=1}^{N} q_n(s_n, x_n; \lambda) \tag{12.38}$$

subject to (12.2)-(12.4).

As it is clear that for each $\lambda \in \mathbb{R}$ the parametric objective function is additive and separable, it follows that an additive Markovian decomposition scheme exists for *Problem* $P(s;\lambda)$. Hence, this problem will give rise to a dynamic programming functional equation of the following form:

$$f_n(s;\lambda) = \max_{x \in D(n,s)} \left\{ q_n(s,x;\lambda) + f_{n+1}(T(n,s,x);\lambda) \right\} \tag{12.39}$$

The important point here is that this equation involves no expansion of the state space, and should therefore be significantly easier to solve than the functional equation defined by (12.31).

Still, to make a convincing case for the proposed fractional/dynamic programming scheme, it is imperative to show that *Dinkelbach's algorithm* is indeed efficient in this case. That is, we need to show that the algorithm will not end going through a large number of parametric problems in the search for the optimal solution to the target problem. For recall that the main thrust of this algorithm is to find an optimal solution to the target problem through an iterative solution of the parametric problem for different values of the parameter λ. Since this is a problem dependent question, it is instructive to examine it in the context of a representative numerical example (for convergence properties of the algorithm see *Lemma B.11*).

12.3.1 Example

Consider the case where

$$v_n(s_n, x_n) = x_n^2 \tag{12.40}$$

$$w_n(s_n, x_n) = \frac{x_n}{n} \tag{12.41}$$

so that

$$q_n(s_n, x_n; \lambda) = \frac{x_n}{n} - \lambda x_n^2 \tag{12.42}$$

The functional equation is then as follows:

$$f_n(s;\lambda) = \max_{x \in \{0,1,2,\dots,s\}} \left\{ \frac{x}{n} - \lambda x^2 + f_{n+1}(s-x;\lambda) \right\} \tag{12.43}$$

with $f_{N+1}(s;\lambda) = 0$ for all s and λ. We shall consider the case where $N = c = 20$.

Table 12.1: Results from Dinkelbach's Algorithm

m	$\lambda^{(m)}$	$x^{(m)}$
0	0	20000000000000000000
1	0.05	9422111000000000000
2	0.117372134	4221111111111100000
3	0.1986545091	3211111111111111000
4	0.2121268758	3211111111111111000

The results yielded by Dinkelbach's algorithm are given in Table 12.1. Note that in this particular case the procedure was initiated with $\lambda = 0$.

The optimal solution yielded by this procedure is $x^{(3)}$, which in turn implies that the optimal value of the objective function is equal to $\lambda^{(4)} = 0.2121268758$. The point to note then is that the parametric problem was solved five times. To appreciate the full significance of these results the reader is encouraged to attempt to solve the problem using the functional equation given by (12.32). □

Let us now examine how this dynamic/fractional programming hybrid algorithm handles a sequential decision problem that is stated in terms of a directed graph.

12.3.2 Example

Consider the graph depicted in Figure 12.1 and assume that the objective function is of the form

$$q(x_1,\dots,x_k) = \frac{\displaystyle\sum_{n=1}^{k} w(x_n,x_{n+1})}{\displaystyle\sum_{n=1}^{k} v(x_n,x_{n+1})} \tag{12.44}$$

where $w(i,j)$ and $v(i,j)$ denote respectively the first and second components of the "length" of $arc(i,j)$. For instance, $w(1,2) = 1$ and $v(1,2) = 2$.

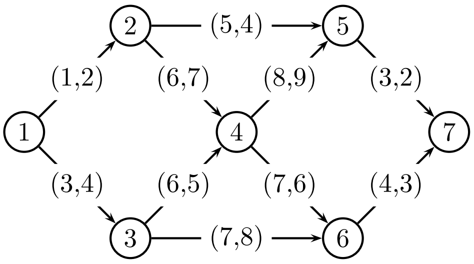


Figure 12.1: A shortest path problem

The object is to find a path from node 1 to node 7 which maximizes the above objective function. Note that if w and v are construed as say, benefit and time functions respectively, then our aim is to maximize the benefit per unit of time. In short, the object is to solve the following sequential decision problem:

$$f(s_1) := \max_{(x_1,\dots,x_k)} \frac{\displaystyle\sum_{n=1}^{k} w(s_n, x_n)}{\displaystyle\sum_{j=1}^{k} v(s_j, x_j)} \ , \ s_1 = 1 \tag{12.45}$$

$$x_n \in D(s_n) \ , \ 1 \leq n \leq k \tag{12.46}$$

$$s_{k+1} = 7 \tag{12.47}$$

$$s_{n+1} = T(s_n, x_n) = x_n \ , \ 1 \leq n \leq k \tag{12.48}$$

where $D(s)$ denotes the set consisting of the immediate successors of node s.

As a prelude I need to explain why opting for the fractional/dynamic programming scheme is called for in the first place.

Observe then that as the objective function is not readily separable, using the standard dynamic programming approach would require a reformulation that would yield state variables of the form $s = (i, a, b)$ where i denotes nodes and a and b respectively denote the first and second component of the length of the path from node 1 to node i.

For example, node 5 will produce the following three states: $s = (5, 6, 6)$, $s' = (5, 15, 18)$ and $s'' = (5, 17, 18)$. The first is induced by the path $(1, 2, 5)$, the second by $(1, 2, 4, 5)$ and the third by $(1, 3, 4, 5)$.

Clearly, no further argument is required to bring home the tremendous complexity of the state space in cases involving a large number of arcs and nodes. But even if one would somehow manage to construct and store the expanded state space, solving the resulting dynamic programming functional equation would still be a daunting task.

On the other hand, if Dinkelbach's algorithm is used, the parametric objective function would have this form

$$g(x_1, \dots, x_k; \lambda) = \sum_{m=1}^{k-1} \left[w(x_m, x_{m+1}) - \lambda v(x_m, x_{m+1}) \right] \ , \ \lambda \in \mathbb{R} \tag{12.49}$$

and the resulting parametric dynamic programming functional equation would be as follows:

$$f(s; \lambda) = \max_{x \in D(s)} \{ w(s, x) - \lambda v(s, x) + f(x; \lambda) \} \ , \ s \in \{1, 2, \dots, 6\} \tag{12.50}$$

starting with $f(7, \lambda) = 0$, $\forall \lambda$.

To illustrate the algorithm we begin with $\lambda^{(0)} = 0$. Solving the functional equation we obtain the optimal path $x^{(1)} = (1, 3, 4, 5, 7)$. Because

$$g(x^{(1)}; \lambda^{(0)}) = \sum_{m=1}^{4} \Big[w(x_m^{(1)}, x_{m+1}^{(1)}) - \lambda^{(0)} v(x_m^{(1)}, x_{m+1}^{(1)}) \Big] \neq 0 \tag{12.51}$$

we set

$$\lambda^{(1)} = g(y^{(1)}) = \frac{\sum_{m=1}^{k-1} w(x_m^{(1)}, x_{m+1}^{(1)})}{\sum_{m=1}^{k-1} v(x_m^{(1)}, x_{m+1}^{(1)})} \tag{12.52}$$

$$= \frac{3+6+8+3}{4+5+9+2} = 1 \tag{12.53}$$

and solve the parametric problem again, this time for $\lambda = \lambda^{(1)}$. The optimal path for this problem is $x^{(2)} = (1, 3, 4, 6, 7)$, yielding

$$g(x^{(2)}; \lambda^{(1)}) = \sum_{m=1}^{4} \Big[w(x_m^{(2)}, x_{m+1}^{(2)}) - \lambda^{(1)} v(x_m^{(2)}, x_{m+1}^{(2)}) \Big] \tag{12.54}$$

$$= 2 \neq 0. \tag{12.55}$$

We thus set

$$\lambda^{(2)} = g(y^{(2)}) = \frac{\sum_{m=1}^{k-1} w(x_m^{(2)}, x_{m+1}^{(2)})}{\sum_{m=1}^{k-1} v(x_m^{(2)}, x_{m+1}^{(2)})} \tag{12.56}$$

$$= \frac{3+6+7+4}{4+5+6+3} = \frac{10}{9} \tag{12.57}$$

and solve the parametric problem again, this time for $\lambda = \lambda^{(2)}$. The optimal path is now $x^{(3)} = (1, 2, 5, 7)$, yielding

$$g(x^{(3)}; \lambda^{(2)}) = \sum_{m=1}^{3} \Big[w(x_m^{(3)}, x_{m+1}^{(3)}) - \lambda^{(2)} v(x_m^{(3)}, x_{m+1}^{(3)}) \Big] \tag{12.58}$$

$$= \frac{1}{9} \neq 0 \tag{12.59}$$

whereupon we set

$$\lambda^{(3)} = g(x^{(3)}) = \frac{\sum_{m=1}^{k-1} w(x_m^{(3)}, x_{m+1}^{(3)})}{\sum_{m=1}^{k-1} v(x_m^{(3)}, x_{m+1}^{(3)})} \tag{12.60}$$

$$= \frac{1+5+3}{2+4+3} = \frac{9}{8} \tag{12.61}$$

and solve the parametric problem for $\lambda = \lambda^{(3)}$. This gives the optimal solution $x^{(4)} = (1,2,5,7)$. Because $x^{(4)} = x^{(3)}$ we stop. The optimal path is then $x^{*} = x^{(3)} = (1,2,5,7)$, and its length is equal to $g(x^{*}) = \lambda^{(3)} = 1.125$.

To sum up then, the fact that the procedure took only four iterations to recover the optimal solution attests to the unquestionable superiority of this scheme, for this case, over the conventional dynamic programming treatment. □

Let us now consider a scheme (hybrid algorithm) based on a dynamic programming/*c*-programming collaboration. I shall first describe its key tenets and then illustrate how it handles a problem with an objective function defined by (12.15).

12.4 C-programming Scheme

To explain the main ingredients of this scheme we can make do with a simple subcase of *Problem* $P(s)$ whose objective function g has this form:

$$g(s_1, z) = u_1(s_1, z) + \varphi(u_2(s_1, z)) \ , \ z = (x_1, x_2, \dots, x_N) \tag{12.62}$$

where

$$u_1(s_1, x_1, x_2, \dots, x_N) = \sum_{n=1}^{N} w_n(s_n, x_n) \tag{12.63}$$

$$u_2(s_1, x_1, x_2, \dots, x_N) = \sum_{n=1}^{N} v_n(s_n, x_n) \tag{12.64}$$

and φ is a nonlinear real-valued function on $\mathbb{R}$.

As required by *c*-programming (see *Appendix C*), it is assumed that φ is differentiable and either *convex* or *concave* depending on whether opt is *max* or *min* respectively (Observe the combination max-convex, min-concave). Again, I begin with the justification for this scheme.

Since φ is nonlinear, a conventional dynamic programming treatment of this problem will run into the same sort of trouble that it encounters with respect to (12.14). That is, with the objective function being non-separable a reformulation is required to construct a Markovian decomposition scheme. This will yield a state variable of the form

$$s'_n = (s_n, a_n) \tag{12.65}$$

where s_n denotes the original state variable and

$$a_n := \sum_{m=1}^{n-1} v_m(s_m, x_m) \ , \ 1 < n \leq m \tag{12.66}$$

with $a_1 := 0$.

The culprit in this case is the term a_n which is generated by the functions $\{v_m\}$ as specified by (12.66). This element can easily engender a state space that would be fundamentally larger than the original state space of the problem. On the other hand, using c-programming to solve this problem we would approach it via a parametric problem whose objective function is as follows:

$$\begin{aligned} g(s_1, x_1, x_2, \dots, x_N; \lambda) &= u_1(s_1, x_1, x_2, \dots, x_N) + \lambda u_2(s_1, x_1, x_2, \dots, x_N) \\ &= \sum_{n=1}^{N} w_n(s_n, x_n) + \lambda \sum_{n=1}^{N} v_n(s_n, x_n) \qquad (12.67) \\ &= \sum_{n=1}^{N} q_n(s_n, x_n; \lambda) \qquad (12.68) \end{aligned}$$

where

$$q_n(s_n, x_n; \lambda) = w_n(s_n, x_n) + \lambda v_n(s_n, x_n) \ , \ \lambda \in \mathbb{R} \qquad (12.69)$$

Clearly, as this is an additive separable function, no expansion of the state space will be required to construct a Markovian decomposition scheme for this case.

Having set out the essentials of the dynamic programming/c-programming scheme, I can now illustrate how it deals with a problem with an objective function of the type defined by (12.15).

12.4.1 Example

Consider the variance-type objective function in *Example 12.1.2*, namely,

$$g(s, x_1, \dots, x_N) = \sum_{n=1}^{N} \left[w_n(x_n) - \frac{1}{N} \sum_{m=1}^{N} w_m(x_m) \right]^2 \qquad (12.70)$$

As there seems to be no obvious way of formulating a Markovian decomposition scheme for this problem without causing an expansion in the state space, we treat it as a non-Markovian problem. That is, had we contemplated the regular dynamic programming treatment, we would have reformulated g to obtain the following:

$$\begin{aligned} g(x_1, x_2, \dots, x_N) = \sum_{n=1}^{N} [w_n(x_n)]^2 &- \left[\frac{2}{N} \sum_{m=1}^{N} w_m(x_m) \right] \sum_{n=1}^{N} w_n(x_n) \\ &+ \sum_{n=1}^{N} \left[\frac{1}{N} \sum_{m=1}^{N} w_m(x_m) \right]^2 \qquad (12.71) \end{aligned}$$

$$= \sum_{n=1}^{N} [w_n(x_n)]^2 - \frac{2}{N}\left[\sum_{m=1}^{N} w_m(x_m)\right]^2 + \frac{1}{N}\left[\sum_{m=1}^{N} w_m(x_m)\right]^2 \tag{12.72}$$

$$= \sum_{n=1}^{N} [w_n(x_n)]^2 - \frac{1}{N}\left[\sum_{m=1}^{N} w_m(x_m)\right]^2 \tag{12.73}$$

Thus, defining

$$a_n := \sum_{m=1}^{n-1} w_m(x_m) \ , \ 1 \leq n \leq N+1 \tag{12.74}$$

would have yielded

$$g(s, x_1, x_2, \dots, x_N) = \sum_{n=1}^{N} \left([w_n(x_n)]^2 - \frac{1}{N}\left[a_{N+1}\right]^2 \right) \tag{12.75}$$

whereupon we would have set $s'_n = (s_n, a_n)$ to enable expressing the objective function as follows:

$$g(x_1, x_2, \dots, x_N) = \sum_{n=1}^{N} b_n(s'_n, x_n) \tag{12.76}$$

where

$$b_n(s'_n, x_n) := \begin{cases} [w_n(x_n)]^2 & , \ n < N \\ [w_N(x_N)]^2 - \frac{1}{N}\left[a_N + w_N(x_N)\right]^2 & , \ n = N \end{cases} \ , \ s'_n = (s_n, a_n) \tag{12.77}$$

This transformation would have given an objective function that is manifestly Markovian with respect to the expanded state variable $s'_n = (s_n, a_n)$. But this expansion in the state space would have easily triggered dimensionality. To see that this is so, observe that the expanded state space S' is now defined thus:

$$S' = \bigcup_{n=1}^{N} S'_n \tag{12.78}$$

where $S'_1 = \{(c, 0)\}$ and

$$\begin{aligned} S'_{n+1} &= \left\{T(s_n, x_n), a_n + w_n(x_n)) : s_n \in S_n, (s_n, a_n) \in S'_n\right\} &(12.79)\\ &= \left\{(s - x, a + w_n(x)) : (s, a) \in S'_n\right\} &(12.80) \end{aligned}$$

It is immediately clear that for a large n these sets can be far larger than the original state space.

On the other hand, bringing in c-programming provides a way out of this predicament. Because, in compliance with the c -programming format we would set

$$u_1(x_1,\dots,x_N) = \sum_{n=1}^{N}[w_n(x_n)]^2 \tag{12.81}$$

$$u_2(x_1,\dots,x_N) = \sum_{n=1}^{N} w_n(x_n) \tag{12.82}$$

$$\varphi(z) = -\frac{z^2}{N} \ , \ z \in U := \mathbb{R} \tag{12.83}$$

so that by construction

$$g(x_1,\dots,x_N) = u_1(x_1,\dots,x_N) + \varphi(u_2(x_1,\dots,x_N)) \tag{12.84}$$

$$\varphi'(z) = -\frac{2z}{N} \ , \ z \in \mathbb{R} \tag{12.85}$$

Because in terms of this formulation opt = min and φ is differentiable and concave on $\mathbb{R}$, the problem under consideration can be classified as a one dimensional concave additive problem (see *Section C.5*). Consequently, the objective function of the parametric problem would have the following form:

$$g(x_1,\dots,x_N;\lambda) = u_1(x_1,\dots,x_N) + \lambda u_2(x_1,\dots,x_N) \ , \ \lambda \in \mathbb{R} \tag{12.86}$$

$$= \sum_{n=1}^{N}[w_n(x_n)]^2 + \lambda \sum_{n=1}^{N} w_n(x_n) \tag{12.87}$$

$$= \sum_{n=1}^{N} \left\{[w_n(x_n)]^2 + \lambda n(x_n)\right\} \tag{12.88}$$

$$= \sum_{n=1}^{N} q_n(x_n;\lambda) \tag{12.89}$$

where

$$q_n(x_n;\lambda) := [w_n(x_n)]^2 + \lambda w_n(x_n) \tag{12.90}$$

The parametric problem itself would be as follows:

$$\textit{Problem } V(\lambda): \ p(\lambda) := \min_{(x_1,\dots,x_N)} \sum_{n=1}^{N} q_n(x_n;\lambda) \ , \ \lambda \in \mathbb{R} \tag{12.91}$$

subject to (12.5)-(12.6).

This problem yields a dynamic programming functional equation of the following form:

$$f_n(s;\lambda) = \min_{x\in\{0,1,2,\dots,s\}} \{q_n(x;\lambda) + f_{n+1}(s-x;\lambda)\} \tag{12.92}$$

for $\lambda \in \mathbb{R}$, $1 \leq m < N$, $s \in \{0, \dots, c\}$, with

$$f_{N+1}(s; \lambda) = 0 \ , \ \forall s \in \{0, 1, \dots, c\} \ , \ \lambda \in \mathbb{R} \tag{12.93}$$

Observe that $p(\lambda) = f_1(c; \lambda)$.

As in the case of the fractional/dynamic programming scheme, the c-programming algorithm would involve an iterative solution of the functional equation (12.92)-(12.93) for various values of λ, until an optimal solution for the original problem is found. □

And here as well, to make a persuasive case for this approach it is necessary to show that the parametric problem will have to be solved only for a relatively small number of values of the parameter λ. For otherwise the advantage of using the dynamic programming/c-programming scheme as opposed to the standard dynamic programming strategy would remain doubtful.

At this stage I cannot provide results that make a categorical statement about the efficiency of the dynamic/c-programming scheme. I can confirm, though, that extensive numerical experiments with this scheme indicate that it is fundamentally more efficient than the standard dynamic programming algorithm based in an expanded state functional equation (see Domingo and Sniedovich [1991]). For an idea of the efficiency of the scheme let us consider the following example.

Continuing with *Example 12.3.1,* consider the case where $N = c = 20$ and $w_n(s, x) = x/n$. In preparation for deploying a c-programming algorithm, we first have to establish lower and upper bounds for the parameter λ. Observe then that we seek the value of the derivative of the function φ, given by (12.85), at the optimum. This means that we can confine the search to the interval $[\lambda_{\min}, \lambda_{\max}]$, where

$$\lambda_{\min} = \min_{(x_1,\dots,x_N)} \varphi'(x_1, \dots, x_N) \tag{12.94}$$

and

$$\lambda_{\max} = \max_{(x_1,\dots,x_N)} \varphi'(x_1, \dots, x_N) \tag{12.95}$$

both subject to (12.50). Since $\varphi'(z) = -2z/N$, it follows that

$$\lambda_{\min} = \min_{(x_1,\dots,x_N)} -\frac{2}{N} u_2(x_1, \dots, x_N) \tag{12.96}$$

$$= \min_{(x_1,\dots,x_N)} -\frac{2}{N} \sum_{n=1}^{N} w_n(x_n) \tag{12.97}$$

$$= -\left(\frac{2}{N}\right) \max_{(x_1,\dots,x_N)} \sum_{n=1}^{N} w_n(x_n \tag{12.98}$$

$$= -\left(\frac{2}{20}\right) \max_{(x_1,\dots,x_{20})} \sum_{n=1}^{20} \frac{x_n}{n} \tag{12.99}$$

Table 12.2: Results generated by the c-programming algorithm

λ	x	$g(x)$
$\lambda_{\text{min}} = -2.0$	12233322110000000000	2.398205
$\lambda_{\text{max}} = -0.1$	00000011111111 222222	0.053687

and

$$\lambda_{\text{max}} = \max_{(x_1,\dots,x_N)} -\frac{2}{N} u_2(x_1,\dots,x_N) \tag{12.100}$$

$$= \max_{(x_1,\dots,x_N)} -\frac{2}{N} \sum_{n=1}^{N} w_n(x_n) \tag{12.101}$$

$$= -\left(\frac{2}{N}\right) \min_{(x_1,\dots,x_N)} \sum_{n=1}^{N} w_n(x_n) \tag{12.102}$$

$$= -\left(\frac{2}{20}\right) \min_{(x_1,\dots,x_{20})} \sum_{n=1}^{20} \frac{x_n}{n} \tag{12.103}$$

both subject to (12.5)-(12.6). Hence, by inspection

$$\lambda_{\text{min}} = -\frac{2c}{N} = -2 \;\; ; \;\; \lambda_{\text{max}} = -\frac{2c}{N^2} = -0.1 \tag{12.104}$$

The inference is then that the optimal value of λ must be on the interval

$$V = [\lambda_{\text{min}}, \lambda_{\text{max}}] = [-2, -0.1] \tag{12.105}$$

Another point to note is that the dynamic programming functional equation (12.92) in our case takes the following form:

$$f_n(s;\lambda) = \min_{x\in\{0,1,\dots s\}} \left\{[w_n(x)]^2 + \lambda w_n(x) + f_{n+1}(s-x;\lambda)\right\} \tag{12.106}$$

where $\lambda \in V$, $1 \leq n < N$, and $s \in \{0,\dots,c\}$. The iterative procedure begins at $n = N+1 = 21$ with $f_{21}(s;\lambda) = 0$, $\forall s, \lambda$.

Let us now use this example to compare the performances of *Algorithm C.5.4.* and *Algorithm C.5.5* (see *Appendix C*). Table 12.2 gives the results obtained in Step 1 for both algorithms.

The results generated in Step 5 of *Algorithm C.5.5* are given in Table 12.3. In this case, the parametric problem was solved for $53 = 51 + 2$ values of λ. Table 12.4 gives the results generated at Step 5 of *Algorithm C.5.4*. Here the parametric problem was solved for $8 = 6 + 2$ values of λ. These results bear out my claim in *Appendix C* that *Algorithm C.5.4* should be expected to perform far more efficiently than *Algorithm C.5.5.* Indeed, my experiments have shown these results to be typical.

To throw more light on the difference between *Algorithm C.5.4* and *Algorithm C.5.5* I shall now analyze the results that they produce in the first two iterations. As both search the same interval, namely,

$$V = [\lambda_{\text{min}} = -2, \lambda_{\text{max}} = -0.1] \tag{12.107}$$

going into the first iteration both use the same values for λ_l and λ_u, namely

$$\lambda_l = \lambda_{\text{min}} = -2 \tag{12.108}$$
$$\lambda_u = \lambda_{\text{max}} = -0.1 \tag{12.109}$$

Consequently, in Step 5 of the first iteration, both generate the same λ, that is

$$\lambda = \frac{u_1(x') - u_1(x'')}{u_2(x'') - u_2(x')} \tag{12.110}$$

where x' and x'' are the optimal solutions found for *Problem* $T(\lambda'_m)$ and *Problem* $T(\lambda")$ respectively (these solutions are given in Table 12.2). This yields $\lambda^\circ = -0.959829$. Let x° denote the optimal solution for *Problem* $T(\lambda^\circ)$ (see Table 12.3).

Now, since neither x' nor x'' is optimal for *Problem* $T(\lambda^\circ)$, *Algorithm C.5.5* proceeds to search the interval $[-2.0, \lambda^\circ] = [-2, -0.959828]$. On the other hand, *Algorithm C.5.4* employs in Step 4 the coverage function defined by (C.80), that is, it computes the intervals already covered by (λ_l, x') and (λ°, x°), and checks whether these exhaust the interval $[-2.0, -0.959828]$.

Since

$$\varphi'(u(x')) = -\frac{2u(x')}{N} = -0.526349 \tag{12.111}$$
$$\varphi'(u(x^\circ)) = -\frac{2u(x^\circ)}{N} = -0.318810 \tag{12.112}$$

it follows that (λ_l, x') covers the interval

$$I(\lambda_l, x') = [\lambda_l, \varphi'(u(x'))] = [-2, -0.526349] \tag{12.113}$$

and (λ°, x°) covers the interval

$$I(\lambda^\circ, x^\circ) = [\lambda^\circ, \varphi'(u_2(x^\circ))] = [-0.955982, -0.318810] \tag{12.114}$$

Thus, the interval covered jointly by these pairs is

$$I(\lambda_l, x') \bigcup I(\lambda^\circ, x^\circ) = [-2, -0.318810] \tag{12.115}$$

The conclusion is then that these two pairs in fact cover the interval $[\lambda_l, \lambda^\circ]$. *Algorithm C.5.4* therefore discards this interval and proceeds to examine the interval $[\lambda^\circ, \lambda_u]$. On the other hand, it takes *Algorithm C.5.5* eight additional iterations to cover the interval $[\lambda_l, \lambda^\circ]$.

Since the pairs $\{(\lambda, x)\}$ generated by the two algorithms cover the entire

Table 12.3: Results for the c-programming scheme

m	λ	x	$g(x)$
1	-0.959829	0 1 1 1 2 2 2 2 2 2 2 1 1 1 0 0 0 0 0 0	0.471058
2	-0.351153	1 1 2 2 3 3 2 2 2 2 1 1 0 0 0 0 0 0 0 0	1.437248
3	-1.803800	1 1 2 2 3 3 2 2 2 1 1 0 0 0 0 0 0 0 0 0	1.654421
4	-1.856270	1 2 2 2 3 3 2 2 2 1 0 0 0 0 0 0 0 0 0 0	2.194882
5	-1.983333	1 2 2 2 3 3 2 2 2 1 0 0 0 0 0 0 0 0 0 0	2.194882
6	-1.813131	1 1 2 2 3 3 3 2 2 1 0 0 0 0 0 0 0 0 0 0	1.723567
7	-1.814286	1 1 2 2 3 3 3 2 2 1 0 0 0 0 0 0 0 0 0 0	1.723567
8	-1.805195	1 1 2 2 3 3 2 2 2 1 1 0 0 0 0 0 0 0 0 0	1.654421
9	-1.646970	1 1 2 2 2 3 2 2 2 2 1 0 0 0 0 0 0 0 0 0	1.531077
10	-1.700000	1 1 2 2 2 3 3 2 2 1 1 0 0 0 0 0 0 0 0 0	1.583245
11	-1.714286	1 1 2 2 2 3 3 2 2 1 1 0 0 0 0 0 0 0 0 0	1.583245
12	-1.680952	1 1 2 2 2 3 2 2 2 2 1 0 0 0 0 0 0 0 0 0	1.531077
13	-1.583333	1 1 2 2 2 2 2 2 2 2 1 1 0 0 0 0 0 0 0 0	1.437248
14	-1.104866	1 1 1 2 2 2 2 2 2 2 1 1 1 0 0 0 0 0 0 0	1.222754
15	-1.276923	1 1 1 2 2 2 2 2 2 2 1 1 1 0 0 0 0 0 0 0	1.222754
16	-1.064305	0 1 1 2 2 2 2 2 2 2 2 1 1 0 0 0 0 0 0 0	0.594931
17	-1.072727	0 1 1 2 2 2 2 2 2 2 2 1 1 0 0 0 0 0 0 0	0.594931
18	-1.021429	0 1 1 1 2 2 2 2 2 2 2 1 1 1 0 0 0 0 0 0	0.471058
19	-0.480455	0 0 1 1 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1 1	0.129821
20	-0.639183	0 1 1 1 1 1 1 2 2 2 2 1 1 1 1 1 1 0 0 0	0.315650
21	-0.786067	0 1 1 1 1 2 2 2 2 2 2 1 1 1 1 0 0 0 0 0	0.397122
22	-0.866667	0 1 1 1 1 2 2 2 2 2 2 1 1 1 1 0 0 0 0 0	0.397122
23	-0.728965	0 1 1 1 1 1 2 2 2 2 2 1 1 1 1 1 0 0 0 0	0.348973
24	-0.762500	0 1 1 1 1 1 2 2 2 2 2 1 1 1 1 1 0 0 0 0	0.348973
25	-0.687395	0 1 1 1 1 1 1 2 2 2 2 1 1 1 1 1 1 0 0 0	0.315650
26	-0.610171	0 1 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1 1 1 0	0.273447
27	-0.610171	0 1 1 1 1 1 1 1 2 2 2 1 1 1 1 1 1 1 0 0	0.291527
28	-0.630556	0 1 1 1 1 1 1 1 2 2 2 1 1 1 1 1 1 1 0 0	0.291527
29	-0.585965	0 1 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1 1 1 0	0.273447
30	-0.548647	0 0 1 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1 1 1	0.133695
31	-0.550000	0 0 1 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1 1 1	0.133695
32	-0.503497	0 0 1 1 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1 1	0.129821
33	-0.287237	0 0 0 1 1 1 1 1 1 1 1 11 1 1 1 1 2 2 2	0.075303
34	-0.371573	0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 1	0.122051
35	-0.423432	0 0 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 1 1 1	0.124855
36	-0.448774	0 0 1 1 1 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1	0.126975
37	-0.464286	0 0 1 1 1 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1	0.126975
38	-0.430769	0 0 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 1 1 1	0.124855
39	-0.389947	0 0 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 1 1	0.123260

Table 12.3: Continued

m	λ	x	$g(x)$
40	0.123260	0 0 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 1 1	0.123260
41	-0.376471	0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 1	0.122051
42	-0.364762	0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 1 1	0.077981
43	-0.366667	0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 1 1	0.077981
44	-0.336462	0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 1	0.076484
45	-0.345395	0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 1	0.076484
46	-0.326471	0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2	0.075303
47	-0.221251	0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2	0.058112
48	-0.272624	0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2	0.058112
49	-0.179898	0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2	0.052692
50	-0.205682	0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2	0.052692
51	-0.144444	0 0 0 0 0 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2	0.053687

Table 12.4: Results for the c-programming scheme

m	λ	x	$g(x)$
1	-0.959828	0 1 1 1 2 2 2 2 2 2 2 2 1 1 1 0 0 0 0 0 0	0.4710579
2	-0.480455	0 0 1 1 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1 1	0.1298207
3	-0.287236	0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2	0.0753033
4	-0.221250	0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2	0.0581120
5	-0.179897	0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2	0.0526924
6	-0.144444	0 0 0 0 0 0 1 1 1 1 1 1 1 1 2 2 2 2 2 2	0.0536871

interval $[\lambda_{\min}, \lambda_{\max}]$, it follows that the optimal solution is the one generated by $\lambda^* = -0.179897$, namely

$$x^* = (0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,2,2,2,2,2) \tag{12.116}$$

The optimal value of the objective function is then $g(x^*) = 0.0526924$. □

I conclude with a short discussion on a dynamic programming / Lagrange multiplier scheme. As schemes of this nature are discussed in detail in most texts on dynamic programming I shall not go into it in depth but shall only note its essentials and comment on certain difficulties that it may run into.

12.5 Lagrange Multiplier Scheme

The rationale for turning to this scheme is similar to the one prompting the fractional programming/dynamic programming and the *c*-programming/dynamic programming schemes, except that here the object is to circumvent an expansion of the state variable induced by *global constraints*.

To illustrate it assume that *Problem* $P(s)$ as defined by (12.1)-(12.4) is additionally subject to the global constraint

$$c(x_1, \ldots, x_N) \leq 0 \tag{12.117}$$

where c is some real-valued function on $X(s)$. As shown in *Chapter 11,* such a constraint will force an expansion in the state space. To avoid this difficulty one would bring in the Lagrange multiplier method which would entail seeking the solution of (12.1)-(12.4), (12.117), via a problem involving the following parametric objective function:

$$L(s, x_1, \ldots, x_N, \lambda) := g(s_1, x_1, \ldots, x_N) + \lambda c(x_1, \ldots, x_N) \ , \ \lambda \in \mathbb{R} \tag{12.118}$$

where g is the objective function of the target problem and λ is an exogenous parameter. Note that the parametric problem is not subject to the global constraint (12.117).

The idea would be then to recover a λ such that any optimal solution for the surrogate parametric problem given by this value of λ is also optimal for the target problem. Usually the procedure would involve an iterative solution of the parametric problem until such a λ is found. I shall not go into the question of what conditions assure that such a λ exists and how to identify it. However, I want to call attention to the following points.

If the intention is to solve the parametric problem with dynamic programming, one must ensure that the parametric objective function is separable because otherwise one difficulty would be traded for another.

To illustrate this point assume that the objective function of the target problem has this form

$$g(s, x_1, \ldots, x_N) = \sum_{n=1}^{N} q_n(x_n) \tag{12.119}$$

and that the additional global constraint is given by

$$c(x_1, \ldots, x_N) = \max_{1 \leq n \leq N} \{d_n(x_n)\} \tag{12.120}$$

Then the parametric objective function would be defined as follows:

$$L(s, x_1, \ldots, x_N, \lambda) := \left\{ \sum_{n=1}^{N} q_n(x_n) \right\} + \lambda \max_{1 \leq n \leq N} \{d_n(x_n)\} \ , \ \lambda \in \mathbb{R} \tag{12.121}$$

Since this objective function is non-separable, giving the parametric problem a dynamic programming formulation will produce an expanded state space. But this is precisely what was sought to be prevented in the first place! The inference therefore is that employing the Lagrange multiplier method in the case of the parametric objective function given by (12.118) will make sense only if g and c are *additive.*

More generally, the Lagrange multiplier/dynamic programming scheme, and for that matter, any other dynamic programming scheme deploying *penalty functions*, will be effective only if the objective function of the parametric problem will be separable. Of course, one would be able to use other approaches (eg. fractional programming, c-programming) to deal with the non-separable objective function of the parametric problem.

Another word of caution is in order. It concerns the idea of using the Lagrange multiplier method with the view to dispense with the state variable altogether. That such an idea should be contemplated is quite obvious.

To see why this is so, suppose that the state variable is induced by a *single* global constraint. Then, on using the Lagrange Multiplier scheme this constraint would be incorporated in the objective function of the resulting parametric problem, to thereby free the target problem from the global constraint. Building on this, one might go a step further to conclude that should the parametric objective function have a Markovian decomposition scheme which is independent of the state variable originally induced by the global constraint, then there would be no need for a state variable at all!

Whatever the appeal of such a "trick", it should be noted that this would normally give rise to a serious difficulty known as "duality gaps". Namely, no $\lambda's$ will exist for which the optimal solution for the parametric problem will be optimal for the target problem. The following example illustrates this point.

12.5.1 Example

Consider the following (unbounded) knapsack problem:

$$\max_{(x_1,\dots,x_N)} \sum_{n=1}^{N} a_n x_n \tag{12.122}$$

$$\sum_{n=1}^{N} b_n x_n \leq r, \ r \geq 0 \tag{12.123}$$

$$x_n \in \{0, 1, 2, \dots, r\} \tag{12.124}$$

where $\{a_n\}$, $\{b_n\}$ and r are positive integers.

The global constraint (12.123) induces in this case a state variable of the form

$$s_n = r - \sum_{m=1}^{n-1} b_m x_m \tag{12.125}$$

whose transition function is of the form

$$T(n, s_n, x_n) = s_n - b_n x_n \tag{12.126}$$

We would thus set

$$D(n, s) = \left\{0, 1, 2, \dots, \left\lfloor \frac{s}{b_n} \right\rfloor \right\} \tag{12.127}$$

This means that the state space is the set $S = \{0, 1, 2, \dots, r\}$.

Now, suppose that we use the Lagrange multiplier method. As our aim is to maximize the objective function it would be convenient to define the parametric objective function thus,

$$L(s, x_1, \dots, x_N, \lambda) = \sum_{n=1}^{N} a_n x_n - \lambda \left(\sum_{n=1}^{N} b_n x_n - r \right) \ , \ \lambda > 0 \tag{12.128}$$

$$= \lambda r + \sum_{n=1}^{N} (a_n - \lambda b_n) x_n \tag{12.129}$$

So, if the constant λr is ignored, the parametric problem would be stated as follows:

$$\max_{(x_1,\dots,x_N)} \sum_{n=1}^{N} (a_n - \lambda b_n) x_n \ , \ \lambda > 0 \tag{12.130}$$

$$x_n \in \left\{ 0, 1, 2, \dots, \left\lfloor \frac{r}{a_n} \right\rfloor \right\} \ , \ 1 \leq n \leq N \tag{12.131}$$

Observe that since the parametric objective function is additive with respect to $\{x_n\}$ and the problem is free of global constraints, it follows that the dynamic programming model would require no state variable at all!

To be sure, in this example the parametric problem is trivially easy, for the optimal solution, as a function of the parameter λ, would be determined by inspection as follows:

$$x_n(\lambda) = \begin{cases} \left\lfloor \frac{r}{a_n} \right\rfloor & , \ a_n - \lambda b_n \geq 0 \\ 0 & , \ \text{otherwise} \end{cases} \ , \ \lambda > 0, n = 1, 2, \dots, N \tag{12.132}$$

But the catch is that by varying λ the Lagrange multiplier method will at best recover N distinct solutions, most of which will not be feasible with respect to the target problem. It follows therefore that one is not assured to obtain an optimal solution to the problem considered.

In short, success in diminishing the state space, indeed in getting rid of it altogether, does not necessarily spell an effective solution procedure for the problem in question. $\square$

As a final note I wish to make it abundantly clear that these comments are not meant to suggest that the Lagrange multiplier/dynamic programming scheme will generally run into trouble. Not at all. Indeed, generally, subject to its inherent limitations, this scheme will perform extremely well.

12.6 Summary

The main point of this chapter is that in many cases a computationally intensive dynamic programming functional equation stemming from an expanded state space, can be staved off through the use of an appropriate alternative solution strategy.

Depending on whether the problem considered involves a non-Markovian objective function and/or is subject to a global constraint, one would be able to opt for a solution scheme that consists of a joint effort between dynamic programming and the suitable parametric method.

12.7 Bibliographic Notes

Collaboration schemes between dynamic programming and the Lagrange multiplier method are discussed in most dynamic programming texts, e.g. Bellman [1957a], White [1969], Denardo [1982, 2003]. Such schemes seem to be particularly well suited for stochastic problems that are subject to chance constraints, e.g. White [1969], Sniedovich [1980a].

More details on dynamic/c-programming schemes can be found in Sniedovich [1987a, 1989a], and details on fractional programming and dynamic programming schemes can be found in Lawler [1976] and Ibaraki [1987]. A tripartite dynamic/fractional/c-programming scheme is described in Sniedovich [1989b].

Separation schemes for dynamic programming models based on multiple objective techniques are described in Henig [1986] and Carraway et al. [1990].

13

The Principle of Optimality

13.1 Introduction

In this chapter I take a close look at the *Principle of Optimality*. I elaborate on the questions that were merely touched on in *Chapter 4,* namely the principle's meaning and validity and its place and role in dynamic programming. I deferred discussion on these issues to this part of the book because it is my view that only against the background of a complete account of the derivation of the functional equation of dynamic programming can one hope to do full justice to these matters.

So, having attended to all the fine details that were still in need of clarification — the Weak-Markovian and monotonicity conditions and having shed more light on the function of the concept of *state*, I can now take up this task. My preeminent objective in expounding these matters is to make clear Bellman's understanding of dynamic programming and to explain the style of exposition that he pursued in setting it out. I also look at the interpretation that the principle is commonly accorded in the literature and at the criticism that has been directed at it.

You will recall that in *Chapter 4* I pointed out two things about the question of how to approach the assessment of the *Principle of Optimality*. First, that no appraisal of the principle's validity can be done in isolation from the multistage decision model in the context of which it was formulated. Second, that to be able to bring out its precise meaning and role in dynamic programming one has to examine it from the standpoint of the principle's place in the derivation of the functional equation. At this stage, we have the necessary background to go into these questions more deeply. I begin with the second point.

The discussion is conducted within the framework of a dynamic programming model whose definition involves a multistage decision model (N, S, D, T, S_1, g) and a decomposition scheme $(\rho, \{g_n : n \in \mathbb{N}\})$ yielding the functional equation:

$$f_n(s) = \operatorname*{opt}_{x \in D(n,s)} \rho(n, s, x, f_{n+1}(T(n, s, x))) \ , \ \forall 1 \leq n < N, s \in S_n \qquad (13.1)$$

The functions $\{f_n\}$ are defined in the context of what we call a family of *modified problems*, namely:

Problem $P(n, s), n \in \mathbb{N}, s \in S_n$:

$$f_n(s) := \operatorname*{opt}_{(x_n,\dots,x_N)} g_n(s, x_n, x_{n+1}, \dots, x_N) \qquad (13.2)$$

$$x_m \in D(m, s_m) \ , \ n \leq m \leq N \qquad (13.3)$$

$$s_n = s \qquad (13.4)$$

$$s_{m+1} = T(n, s_m, x_m) \ , \ n \leq m \leq N \qquad (13.5)$$

We referred to *Problem* $P(n,s)$ as the MODIFIED PROBLEM AT (N,S) and we let $X^*(n,s)$ denote the set of optimal solutions for *Problem* $P(n,s)$.

The decomposition scheme is required to satisfy the following condition:

$$g_n(s,x,x_{n+1},\dots,x_N) = \rho(n,s,x,g_{n+1}(T(n,s,x),x_{n+1},\dots,x_N)) \tag{13.6}$$

for any collection $(n,s,x,x_{n+1},\dots,x_N)$ such that $1 \le n < N$, $s \in S_n$, $x \in D(n,s)$, $(x_{n+1},\dots,x_N) \in X(n+1,T(n,s,x))$, the latter denoting the set of a feasible solution for *Problem* $P(n+1,T(n,s,x))$.

Turning now to the derivation process itself. As we have seen, an appeal to the *Principle of Conditional Optimization* enables deducing from the family of modified problems, the following family of CONDITIONAL PROBLEMS:

Problem $P(n,s,x), 1 \le n < N, s \in S_n, x \in D(n,s)$:

$$f_n(s,x) := \underset{(x_{n+1},\dots,x_N)}{\text{opt}} \; g_n(s,x,x_{n+1},\dots,x_N) \tag{13.7}$$

$$x_m \in D(m,s_m) \;,\; n+1 \le m \le N \tag{13.8}$$

$$s_n = s \tag{13.9}$$

$$s_{m+1} = T(m,s_m,x_m) \;,\; n \le m \le N \tag{13.10}$$

with

$$f_N(s,x) := g_N(s,x) \;,\; s \in S_N \;,\; x \in D(N,s) \tag{13.11}$$

if N is finite.

The important thing to note here is that, irrespective of the particular nature of the decomposition scheme, we always obtain the following equation:

$$f_n(s) = \underset{x \in D(n,s)}{\text{opt}} \; f_n(s,x) \;,\; \forall 1 \le n < N, s \in S_n \tag{13.12}$$

The immediate implication is then that, in view of (13.12) being always true, and considering that by definition any decomposition scheme satisfies (13.6), it is only logical to regard

$$f_n(s,x) = \rho(n,s,x,f_{n+1}(T(n,s,x))) \tag{13.13}$$

for $1 \le n < N, s \in S_n, x \in D(n,s)$, as a *sufficient condition* for (13.1).

In other words, as it is clear that (13.12)-(13.13) imply (13.1), the validity of the functional equation is clearly guaranteed by the validity of (13.13).

Now, the question naturally arising is: *in what does one anchor the validity of (13.13)?* It need hardly be pointed out that an answer to this question depends entirely on one's *fundamental* approach to this equation. Thus, one may seek to tie the validity of (13.13) to the properties of the composition function ρ, arguing that by inspection of (13.2)-(13.10), the validity of (13.13) is contingent on the composition function ρ being *monotone non-decreasing with its last argument* (See *Section 10.1*).

However, approaching (13.13) from the viewpoint of the wider conceptual context of which this equation forms part, one would most naturally trace its validity to

$$X^*(n,s,x) \subseteq X^*(n+1,T(n,s,x)) \tag{13.14}$$

for all $1 \leq n < N, s \in S_n, x \in D(n,s)$, or better yet to

$$X^*(n,s,x) \cap X^*(n+1,T(n,s,x)) \neq \varnothing \tag{13.15}$$

for all $1 \leq n < N, s \in S_n, x \in D(n,s)$, where $\varnothing$ denotes the empty set.

Or in other words, one would choose either (13.14) or (13.15) as a sufficient condition for (13.13), both being equally compelling. The difference between them is that while (13.14) requires *any* optimal solution to *Problem* $P(n,s,x)$ to be also optimal for *Problem* $P(n+1,T(n,s,x))$, (13.15) imposes a weaker requirement. It makes do with there being, for each instance of (n,s,x), *at least one* optimal solution for *Problem* $P(n,s,x)$ that is also optimal for *Problem* $P(n+1,T(n,s,x))$.

As we saw in *Section 10.1,* the main consequence of basing the functional equation on the less stringent requirement is that such an equation is not assured to recover *all* the optimal solutions for *Problem P.* However, from a purely methodological point of view this is in essence a technical matter. The important thing is that both (13.14) and (13.15) guarantee the optimality of the recovered solutions, tying as they do the optimal solutions of *Problem* $P(n,s,x)$ to those of the modified problem it generates, namely *Problem* $P(n+1,T(n,s,x))$.

Now, recall that we saw in *Chapter 4* that (13.14) implies a stronger relation, namely

$$X^*(n,s,x) = X^*(n+1,T(n,s,x)) \tag{13.16}$$

for all $1 \leq n < N, s \in S_n, x \in D(n,s)$.

Also note that from (13.6) it is clear that (13.15) entails that

$$X^*(n+1,T(n,s,x)) \subseteq X^*(n,s,x) \tag{13.17}$$

for all $1 \leq n < N, s \in S_n, x \in D(n,s)$.

So, as (13.16) clearly implies (13.14), the two statements can be regarded equivalent and, for similar reasons, (13.17) and (13.15) can also be regarded equivalent.

The point then is that any one of the above requirements can aptly serve as a sufficient condition for the validity of the dynamic programming functional equation (13.1). However, in view of (13.14) and (13.16) establishing a 'strong' relation whereas (13.15) and (13.17) a 'weak' relation, I labeled the first pair the *Markovian Conditions* and the second the *Weak-Markovian Conditions.*

Let us now remind ourselves of the trait that gives these conditions their

Markovian nature. To this end it is instructive to incorporate the state $s' = T(n,s,x)$ in the statement of the conditional problem at (n,s,x).

So let us define

Problem $P(n,s,x,s'), 1 \leq n < N, s \in S_n, x \in D(n,s), s' = T(n,s,x)$:

$$f_n(s,x,s') := \underset{(x_{n+1},\dots,x_N)}{\text{opt}} \; g_n(s,x,s',x_{n+1},x_{n+2},\dots,x_N) \tag{13.18}$$

subject to (13.8)-(13.11), and let $X^*(n,s,x,s')$ denote the set of optimal solutions for *Problem* $P(n,s,x,s')$.

Since by construction *Problem* $P(n,s,x,s')$, $s' = T(n,s,x)$, is in fact identical to *Problem* $P(n,s,x)$, it clearly follows that $X^*(n,s,x,T(n,s,x)) = X^*(n,s,x)$. This allows rewriting (13.14)-(13.15) as

$$X^*(n,s,x,T(n,s,x)) \subseteq X^*(n+1,T(n,s,x)) \tag{13.19}$$

$$X^*(n,s,x,T(n,s,x)) \cap X^*(n+1,T(n,s,x)) \neq \varnothing \tag{13.20}$$

and (13.16)-(13.17) as

$$X^*(n,s,x,T(n,s,x)) = X^*(n+1,T(n,s,x)) \tag{13.21}$$

$$X^*(n+1,T(n,s,x)) \subseteq X^*(n,s,x,T(n,s,x)) \tag{13.22}$$

respectively. These relations apply for all $1 \leq n < N, s \in S_n, x \in D(n,s)$.

The Markovian character of (13.19) and (13.21) lies then in their implying that the set of optimal solutions to *Problem* $P(n,s,x,s'), s' = T(n,s,x)$, is uniquely determined by the state $s' = T(n,s,x)$ resulting from (n,s,x). Thus, for any two pairs (s,x) and (s°,x°) such that $x \in D(n,s)$, $x^\circ \in D(n,s^\circ)$ and $T(n,s,x) = s' = T(n,s^\circ,x^\circ)$, the respective conditional problems, namely *Problem* $P(n,s,x,s')$ and *Problem* $P(n,s^\circ,x^\circ,s')$, are equivalent in that both have the same set of optimal solutions.

Similarly, (13.20) and (13.22) imply that any two or more conditional problems at stage n that generate the *same state* at stage $n+1$ have *at least one* common optimal solution. Again, this is a typical Markovian condition, although it is not as strong as the one articulated by (13.19) or (13.21).

In summary, the inference to be drawn from the analyses in *Chapter 4* and *Chapter 10* is that any optimization problem possessing a decomposition scheme satisfying any one of the above *Markovian Conditions* will yield a valid functional equation of the form specified by (13.1).

The functional equation is thus viewed as an immediate consequence of

· The *Principle of Conditional Optimization* as manifested in (13.12).
· Either one of the *Markovian Conditions*.

With this as background let us now examine the *Principle of Optimality*.

13.2 Bellman's Principle of Optimality

It is only fitting to begin the examination of the famous statement that Bellman entitled *The Principle of Optimality* by first reminding ourselves of its wording. Prior to this I wish to point out that although I follow the accepted practice of associating it with Bellman's first book on dynamic programming [1957a, p. 83], the principle was in fact introduced by Bellman in 1953 and reiterated in the very same phrasing, in numerous other publications:

PRINCIPLE OF OPTIMALITY. *An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.*

Now, as already indicated, it is impossible to work out the principle's exact import without imputing precise meaning, or in other words, mathematical content, to the key phrases 'initial state', 'initial decision', 'the state resulting from' etc. This, needless to say, requires postulating a mathematical model. The model that I shall use for this purpose will naturally be the one that I set out in the preceding chapters.

Thus, reading the principle in terms of this model the objects 'initial state' and 'initial decision' are seen to pertain to a *generic conditional problem* associated with some *assumed decomposition scheme*. Any non final stage is denoted by n, any element of S_n by s and any element of $D(n,s)$ by x.

The policy in question is understood to be capable of generating optimal decisions for *any* problem imaginable. Hence, applying this policy to *Problem* $P(n,s,x)$ yields a sequence of decisions $z = (x_{n+1},\dots,x_N) \in X^*(n,s,x)$, where $X^*(n,s,x)$ denotes the set of optimal solutions for *Problem* $P(n,s,x)$.

The principle's contention is then that z must also be an optimal solution for the modified problem generated by (n,s,x), namely the modified problem at $(n+1,T(n,s,x))$, or as we call it, *Problem* $P(n+1,T(n,s,x))$.

That is, read in the context of our multistage decision model, the principle asserts the following:

> *If z is an optimal solution for Problem $P(n,s,x)$ then it must also constitute an optimal solution for the modified problem resulting from (n,s,x), namely Problem $P(n+1,T(n,s,x))$.*

It is clear then that in this setting the principle makes the very same claim that is made by the *Markovian Condition*, namely by (13.14). Or in other words, (13.14) and the principle amount to the same idea.

This established, the question naturally arises whether as in the case of the *Markovian Condition* one would be able, here as well, to formulate a weaker version of the original. That this is indeed possible is immediately evident. All one needs to do to this end is to read "An optimal policy" as affirming "There exists an optimal policy". This rendition has the effect of producing a less demanding version of the *Principle of Optimality* that is totally equivalent to the *Weak-Markovian Condition* (13.15). I shall therefore refer to (13.15) also as the *Weak Principle of Optimality*. This principle makes the following assertion:

> *Whatever the stage $n < N$, the state $s \in S_n$ and the decision $x \in D(n,s)$ are, at least one of the optimal solutions for Problem $P(n,s,x)$ is also optimal for the modified problem resulting from the triplet (n,s,x), namely Problem $P(n+1, T(n,s,x))$.*

What conclusions can be drawn then in view of these findings? Having determined that the *Markovian Condition* and *Bellman's Principle of Optimality* are in fact one, clearly the *Principle of Optimality* and its weaker version — the *Weak Principle* — emerge as the backbone of dynamic programming's theoretical foundation. This proposition can be summed up formally by means of the following result:

Corollary 13.2.1 *Any decomposition scheme satisfying the Principle of Optimality yields a valid functional equation.*

Before we can proceed I wish to reassure the reader of the principle's validity, considering the foundational role just attributed it. At this stage, the argument adduced for this purpose is an interim measure. Later in this chapter I come back to this question and I discuss it in greater detail. For now it is sufficient to keep in mind that having equated the *Markovian Condition* with the *Principle of Optimality*, the simple *monotonicity conditions* pertaining to the decomposition function ρ — which as shown in *Section 10.1* can be appealed to in order to insure the validity of the *Markovian Conditions* — apply to the principle as well.

As for the relation between the principle and the optimization problem itself namely, *Problem P*. First consider the following:

Definition 13.2.1 *An instance of Problem P is said to* OBEY *the Principle of Optimality if it can be formulated as a dynamic programming problem whose decomposition scheme satisfies the principle, namely (13.14). Similarly, an instance of Problem P is said to obey the Weak Principle of Optimality if it can be formulated as a dynamic programming problem whose decomposition scheme satisfies the Weak Principle, namely (13.15).*

Considering then that the *Principle* has been shown to be the equivalent of the *Markovian Condition*, the following is an immediate consequence of *Lemma 4.4.5*.

Corollary 13.2.2 *Any instance of Problem P obeys the Principle of Optimality.*

The conclusion to be drawn on the strength of this analysis is that the guiding idea in dynamic programming is to formulate any given instance of *Problem P* so that the formulation meets the precepts of the *Principle of Optimality* or its weaker version the *Weak Principle*.

Let us now examine how the *Principle of Optimality* is commonly construed in the literature.

13.3 Prevailing Interpretation

The interpretation of the *Principle of Optimality* that I set out above differs from the interpretation that one is likely to encounter in the literature. This disparity is due to different readings of the term *'initial decision'*. In my interpretation this term is understood to mean "any *feasible* initial decision" whereas the prevailing interpretation construes this to mean "any *optimal* initial decision". Thus, applying the terminology of our model to the prevailing interpretation, Bellman's principle would read as follows:

> *If $(x_n,\dots,x_N)$ is an optimal solution for Problem $P(n,s)$ then the sequence $(x_{n+1},\dots,x_N)$ must constitute an optimal solution for the modified problem induced by (n,s,x_n), namely Problem $P(n+1,T(n,s,x_n))$.*

This means that whereas my interpretation involves a *conditional problem and the modified problem that it generates*, the prevailing interpretation involves a *modified problem and the modified problem that it generates in conjunction with its optimal decisions*. Or to put it more succinctly, on the prevailing interpretation the principle contends that

$$(x_n,\dots,x_N)\in X^*(n,s)\Longrightarrow (x_{n+1},\dots,x_N)\in X^*(n+1,T(n,s,x_n)) \tag{13.23}$$

Now, the point to note here is that although it is possible to show that (13.23) guarantees the validity of the functional equation (13.1), the proof is not as direct as that based on (13.14). The reason for this is that (13.23) does not imply that

$$f_n(s,x)=\rho(n,s,x,f_{n+1}(T(n,s,x)))\ ,\ \forall x\in D(n,s) \tag{13.24}$$

but rather only that

$$f_n(s,x)=\rho(n,s,x,f_{n+1}(T(n,s,x)))\ ,\ \forall x\in \{z(1):z\in X^*(n,s)\} \tag{13.25}$$

To illustrate the issue involved assume that opt $=$ max and that $z^\circ = (x^\circ_n,\dots,x^\circ_N)$ is an optimal solution for *Problem* $P(n,s)$ for some $1\le n <$

N and $s \in S_n$. The proof of (13.1) by means of (13.23) consists of two arguments. First, invoking (13.23) we obtain

$$g_n(s, z^\circ) = f_n(s) = \rho(n, s, x, f_{n+1}(T(n, s, x_n^\circ))) \tag{13.26}$$

Second, if contrary to the assertion we assume that (13.1) does not hold, it will follow that

$$f_n(s) < \rho(n, s, x^*, f_{n+1}(T(n, s, x^*))) \tag{13.27}$$

for some $x^* \in D(n, s)$.

Set then $s^* = T(n, s, x^*)$ and let $z^* = (x_{n+1}^*, \dots, x_N^*)$ be any optimal solution for *Problem* $P(n+1, s^*)$. Since this entails that

$$f_{n+1}(s^*) = g_{n+1}(s^*, z^*) \tag{13.28}$$

we conclude that

$$g_n(s, z^\circ) = f_n(s) < \rho(n, s, x^*, g_{n+1}(s^*, z^*)) = g_n(s, x^*, z^*) \tag{13.29}$$

However, since by construction the sequence $z' = (x^*, z^*)$ constitutes a feasible solution for *Problem* $P(n, s)$, (13.29) contradicts the assertion that z° is an optimal solution for *Problem* $P(n, s)$. We therefore conclude that (13.23) entails (13.1).

On the other hand, in terms of my interpretation, the proof of (13.1) amounts to no more than the following: The *Principle of Conditional Optimization* implies that

$$f_n(s) = \underset{x \in D(n,s)}{\text{opt}} \; f_n(s, x) \; , \; \forall 1 \leq n < N, s \in S_n \tag{13.30}$$

and the *Principle of Optimality* implies that

$$f_n(s, x) = \rho(n, s, x, f_{n+1}(T(n, s, x))) \tag{13.31}$$

for all $1 \leq n < N, s \in S_n, x \in D(n, s)$. Hence, (13.30)-(13.31) imply

$$f_n(s) = \underset{x \in D(n,s)}{\text{opt}} \; \rho(n, s, x, f_{n+1}(T(n, s, x))) \tag{13.32}$$

for all $1 \leq n < N, s \in S_n, x \in D(n, s)$.

My interpretation of Bellman's principle has another important advantage in that it allows a straightforward extension of the principle to *stochastic* models with *nondenumerable state spaces*. The point is that the prevailing interpretation of the principle requires a technical correction in such models. This issue is discussed in *Appendix D*.

Also, an examination of Bellman's writings leaves no room for speculation as to the intended meaning of the term 'initial decision'. From the way Bellman used to invoke the principle it is clear that he consistently took the term 'initial decision' to connote 'any feasible initial decision'. This is patently evident, for instance, in his derivation of the functional equation in pp. 83-84 of his 1957 book.

13.4 Variations on a Theme

Perhaps one of the most telling things about the *Principle of Optimality* is that from the start it became the focal point of the commentary on dynamic programming. As it was immediately recognized that a satisfactory explanation of the theoretic foundation of dynamic programming was bound to a satisfactory explanation of the principle, elucidating the principle's precise meaning and role became part of the study of dynamic programming.

Also, alongside efforts to make clear its purport, attempts were made to revise it. These grew out of the view, expressed either covertly or overtly, that in Bellman's formulation the principle was either too loosely phrased or too vague or downright imprecise, and that it therefore required revision. Finally, Bellman's principle also proved a source of inspiration for the formulation of other principles.

So, over the years a number of principles were put forward in dynamic programming, to serve either as sufficient conditions for the dynamic programming functional equation or to describe fundamental properties of the optimal policy. I briefly mention two such principles to which I refer as *versions.*

13.4.1 Version # 1

This version was proposed by Denardo and Mitten [1967] as an alternative to Bellman's principle. Using the terminology of our model it reads as follows:

> *There exists a Markovian policy which is simultaneously optimal for all the modified problems.*

Although it can serve as a sufficient condition for the optimality equation, as in the case of the prevailing interpretation, the functional equation cannot be deduced from it as immediately as from Bellman's principle. In fact, proving the validity of this version of the principle for nontruncated problems requires postulating additional convergence properties (see Denardo [1965]). Its principal merit is that it brings out an important fundamental Markovian property of dynamic programming policies.

13.4.2 Version # 2

This version, it appears, aims to highlight the structural feature of optimal policies pertaining to *nonserial* sequential decision processes (see *Section 9.2.3*). Roughly, it states the following (See Mitten [1964, p. 414]):

> *Any subsection of an optimal sequence of decisions must be optimal with respect to its end-points.*

It is interesting to note that this version was rediscovered some thirty years later in the context of a *directed decision hypergraph* model (See Martin, Rardin and Campbell [1990, p. 131]), where it reads as follows:

> *An optimal structure in the composition graph must be formed from optimal structures in the composed subgraphs.*

Here again the *Principle of Optimality* is not intended to link the *Principle of Conditional Optimization* to the dynamic programming functional equation, but rather to provide a composition and decomposition principle for the analysis of non-serial processes.

13.5 Criticism

Much of the comment evoked by the *Principle of Optimality* has been in the form of criticism that focused on three key issues: the principle's *exact meaning*, its *validity* and its *status in the theory.* In the ensuing sections I examine this criticism and I reflect on it.

13.5.1 Meaning

The main complaint about the principle's purport has argued that it is *ambiguous.* My position on this is that such arguments arise from a reading of the principle that takes no cognizance of the fact that the principle was stated in a certain context and that to make sense of it one must read it within this context. Indeed, the literature is spotted with interpretations of the principle that treat it as though it were some context-independent general statement enunciating some universal optimality axiom. To appreciate the difficulties one might encounter in construing the principle as a context-independent optimality criterion, consider the following version of the principle:

> ...*If you don't do the best you can with what you happen to have you'll never do the best you might have done with what you should have had* ...
>
> Aris [1964, p. 27]

But, as demonstrated above, read in terms of the dynamic programming model formulated in this book, the *Principle of Optimality* is tantamount to (13.14) and as such its purport, intent, and role are crystal clear.

13.5.2 Validity

The most serious criticism directed at the *Principle of Optimality* has called into question its validity. The main thrust of this criticism is that the principle does not always hold. Given the ramifications of this objection for the very validity of the dynamic programming paradigm that one adopts, we need to examine it carefully. The point is this. Suppose that one formulates a dynamic programming model whose decomposition scheme defies the *Principle of Optimality.* What are the implications of such an example?

Let us consider this question by means of a problem that appears to *contradict* Bellman's *Principle of Optimality.*

13.5.2.1 Example

Consider the graph depicted in Figure 13.1. The problem is to find the *shortest* path(s) from node 1 to node 7, the length of a path being equal to the length of its *longest* arc. For example, the length of the path specified by the nodes $(1, 2, 5, 7)$ is equal to 4 because the longest arc on this path is $(1, 2)$ and its length is equal to 4.

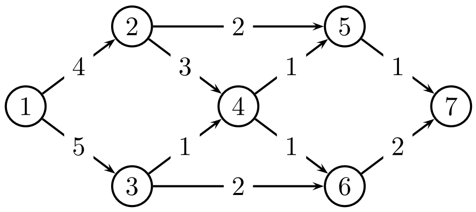


Figure 13.1: Alleged counter-example

Now, suppose that the nodes are viewed as states, and that $s = 1$. It is immediately clear that the path $p = (1, 2, 4, 6, 7)$ is optimal with respect to $s = 1$ because node 7 cannot be reached from node 1 via a path whose longest arc is smaller than 4.

Hence, in keeping with the prevailing interpretation of the principle, the sub-path $p' = (2, 4, 6, 7)$ should constitute an optimal path with respect to node $s' = 2$ which is the node generated by $s = 1$ and the first arc on p. But clearly p' is not an optimal path for $s' = 2$ as it is plain that the path $p' = (2, 5, 7)$ is shorter. The essential point is then that the *Principle of Optimality* does not hold in this case. □

The point of this and similar arguments is that there are valid dynamic programming formulations involving valid decomposition schemes, and hence valid dynamic programming functional equations, which nevertheless do not obey the *Principle of Optimality.* How then does one reconcile this fact with the central role that the principle seems to play in a conception of dynamic programming, or a dynamic programming methodology, such as the one that I have advanced here? Clearly the implication is that the status of the *Principle of Optimality* in such a conception is unclear.

My reply to the objection expressed through counter-examples of this type is as follows. To begin with, the sequential decision model describing the problem in the above example *fully obeys* the *Weak Principle of Optimality.* But what is more, this model can easily be reformulated so as to obey the

Principle of Optimality. Indeed, all one needs to do to this end is to extend the state variable so that it will consist of a pair $s = (k, v)$ where k denotes a node and v is the length of the longest arc traversed on the path from node 1 to node k.

To state this more generally, granted *Corollary 13.2.3*, it is clear that any discrete optimization problem can be formulated so that it will comply with the *Principle of Optimality.* Should one insist, however, that for the sake of completeness, formulations such as that featured in the example must also be accounted for, then clearly such formulations would fall *directly* under the weaker version of the principle, that is, the *Weak Principle of Optimality*.

The conclusion therefore is that a dynamic programming paradigm based on the *Principle of Optimality* is as sound as can be, which brings us to an examination of the principle's status in the theory.

13.5.3 Status in the Theory

Having rebutted the claims against the principle's validity and thus dispelled the misgivings about a dynamic programming methodology grounded on it, let us now examine what status would the principle have in such a methodology. It is important to appreciate then that the principle can play two distinct roles in a dynamic programming investigation.

It can either have the status of a *mathematical truth*, say a *Theorem*, *Lemma*, etc., or that of a *postulate* or an *assumption.*

Thus, if ascribed the status of a Theorem, the principle would be taken to describe a property that a *given* dynamic programming formulation can be shown to have. For example, the following is a valid argument.

Assumption 13.5.1 *The composition function ρ satisfies the condition $\rho(n, s, x, a) = a, \forall 1 \leq n < N, s \in S_n, x \in D(n, s), a \in \mathbb{R}$.*

Theorem 13.5.1 *Principle of Optimality:) Subject to the above assumption $X^*(n, s, x) \subseteq X^*(n + 1, T(n, s, x))$, $\forall 1 \leq n < N$, $s \in S_n$, $x \in D(n, s)$.*

Of course, in this capacity the validity of the principle must be proved, formally or otherwise. But if ascribed the role of a *postulate* or an *assumption*, the principle would be understood in the following manner.

Assumption 13.5.2 *Principle of Optimality: The decomposition scheme satisfies the condition $X^*(n, s, x) \subseteq X^*(n + 1, T(n, s, x))$, $\forall 1 \leq n < N$, $s \in S_n$, $x \in D(n, s)$.*

Theorem 13.5.2 *If the Principle of Optimality holds then the decomposition scheme yields a valid functional equation, namely*

$$f_n(s) = \underset{x \in D(n,s)}{\text{opt}} \rho(n, s, x, f_{n+1}(T(n, s, x))) \ , \ \forall 1 \leq n < N, s \in S_n \qquad (13.33)$$

The implication is therefore that it is of the utmost importance to be clear on how the principle is being treated in a given context.

It should be noted in this regard that in general it is advisable to avoid treating the principle as an assumption. For one thing, this will prevent situations in which one will unwittingly postulate the principle in contexts where it would turn out to be at variance with other assumptions. Also, in most *concrete* situations, it is a straightforward matter to determine whether or not the principle holds.

To sum up then, *attributing the principle the status of a postulate should be reserved exclusively for legitimate methodological purposes.*

13.6 So What Is Amiss?

At this point there remains only one question. Having seen that the validity of the *Principle of Optimality* is not at issue, and hence by implication that a dynamic programming methodology resting on the principle is equally sound, why is it then that Bellman's explanation of dynamic programming has been described by some as heuristic or lacking in rigor?

Obviously one would have to speculate to work out a reply to this question. Still, I believe that the following two explanations answer it, at least in part.

First, an incautious treatment of the *Principle of Optimality*, as we have seen, is bound to give rise to difficulties. This may easily be put down, erroneously, to a fundamental difficulty in the theory itself.

Secondly, *it seems that Bellman's descriptive, non-technical exposition of the key tenets of dynamic programming has been mistaken for a lack of mathematical rigor.*

In the next two sections I take up these two points with the view to explain Bellman's conception of dynamic programming and his approach to its formulation.

13.7 The Final State Model Revisited

You will recall that I argued in *Chapter 4* that in virtue of being the most rudimentary and general model imaginable, the final state model furnishes an ideal framework for proving the validity of dynamic programming's fundamental concepts. I now wish to illustrate how this can be put to good effect in the *exposition of dynamic programming.*

The point of this exercise is first of all to emphasize that this is precisely the model that Bellman used as a setting for the formulation of the basics of dynamic programming — a point that seems to have escaped critics and non-critics alike. And second, to lend support to his decision to use this model for this purpose.

Suppose then that instead of employing the multistage decision model (N, S, D, T, S_1, g) as a medium of discourse, I would have used the final state model. Then the discussion in *Chapter 4* would have taken the following course. Proceeding from the assumption that the objective function g depends only on the final state of the process, I would have postulated the following:

Assumption 13.7.1 *The objective function g is defined on the set $\cup_{n=0}^{N} S \times \mathbb{D}^{N-n}$ and it has the following property: let s_1 be any element of S_1, and let $(x_1, \dots, x_N)$ be any feasible solution to Problem $P(s_1)$. Then,*

$$g(s_1, x_2, x_3, \dots, x_N) = g(s_n, x_n, x_{n+1}, \dots, x_N) \ , \ \forall n \in \mathbb{N} \tag{13.34}$$

where s_n denotes the state generated by $(s_1, x_1, x_2, \dots, x_{n-1})$.

This would have rendered a decomposition scheme unnecessary because the objective function g would have applied to all the modified and conditional problems. That is, the set of modified problems would have had this form

Problem $P(n, s), 1 \leq n \leq N, s \in S:$

$$f_n(s) := \underset{(x_n, \dots, x_N)}{\text{opt}} \ g(s, x_n, x_{n+1}, \dots, x_N) \tag{13.35}$$

$$x_k \in D(k, s_k) \ , \ n \leq k < N \tag{13.36}$$

$$s_{k+1} = T(k, s_k, x_k) \ , \ n \leq k < N \tag{13.37}$$

$$s_n = s \tag{13.38}$$

Similarly, the conditional problems would have taken this form:

Problem $P(n, s, x), 1 \leq n < N, s \in S, x \in D(n, s):$

$$f_n(s, x) := \underset{(x_{n+1}, \dots, x_N)}{\text{opt}} \ g(s, x, x_{n+1}, x_{n+2}, \dots, x_N) \tag{13.39}$$

$$x_k \in D(k, s_k) \ , \ n + 1 \leq k < N \tag{13.40}$$

$$s_{k+1} = T(k, s_k, x_k) \ , \ n + 1 \leq k < N \tag{13.41}$$

$$s_n = s \tag{13.42}$$

$$x_n = x \tag{13.43}$$

The *Principle of Conditional Optimization* would have then yielded

Corollary 13.7.1

$$f_n(s) = \underset{x \in D(n,s)}{\text{opt}} \ f_n(s, x) \ , \ \forall 1 \leq n < N, s \in S_n \tag{13.44}$$

Whereupon, by inspection, we would have obtained

Corollary 13.7.2 *Let (n, s, x) be any triplet (n, s, x) such that $1 \leq n < N, s \in S_n$, and $x \in D(n, s)$. Then Problem $P(n, s, x)$ is identical to Problem $P(n + 1, T(n, s, x))$, namely both have the same objective function and the same feasible solutions.*

Therefore,

Corollary 13.7.3 *Principle of Optimality:*

$$X^*(n, s, x) \subseteq X^*(n + 1, T(n, s, x)) \tag{13.45}$$

for all $1 \leq n < N, s \in S_n, x \in D(n, s)$ which in turn would have yielded

Corollary 13.7.4 *Functional Equation:*

$$f_n(s) = \underset{x \in D(n,s)}{\text{opt}} \ f_{n+1}(T(n, s, x)) \ , \ \forall 1 \leq n < N, s \in S_n \tag{13.46}$$

This, in a word, is the essence of dynamic programming.

Although I have chosen not to use the final state model as described above, plainly this is the simplest framework in which to introduce the *Principle of Optimality*, its validity being immediately obvious here or, to quote Bellman [1957a, p. 83]:

"... a proof by contradiction is immediate ..."

The inescapable conclusion is then that much of the adverse comment about Bellman's formulation of dynamic programming is due to a failure to recognize that the discussion in Chapter 3 of Bellman's book was based on the premise that the underlying model is a *final state model* (see Bellman [1957a, pp. 81-82]). With this in mind let us now examine Bellman's approach.

13.8 Bellman's Treatment of Dynamic Programming

The most striking feature of the style of exposition that Bellman pursued in setting out dynamic programming, more precisely its conceptual dimension, was a simplicity of presentation. In stark contrast to my method of presentation in this book, Bellman never engaged in a painstaking *technical* analysis of the theoretical foundation of dynamic programming, but rather condensed it into a short outline which he articulated verbally.

This style of exposition, it seems, was an expression of Bellman's conviction that the cluster of ideas underlying dynamic programming is profoundly simple, immediately intelligible and of an intuitively compelling validity. This comes out time and again in his early articles on dynamic programming. One imagines therefore that it did not occur to him that an explanation of this set of simple ideas called for a laborious *technical* discussion.

Thus, stress is continually laid in Bellman's early writings on the fact that the essence of dynamic programming boils down to the triad

Multistage Decision Process

Principle of Optimality

Functional Equation

which lends itself to a concise verbal statement.

To illustrate Bellman's conception of dynamic programming and his approach to its exposition, one could do no better than cite one of Bellman's own formulations, in which the totality of dynamic programming is presented in a nutshell. It reads as follows:

> A GENERAL DESCRIPTION OF DYNAMIC PROGRAMMING PROBLEMS
> Having given some examples of dynamic-programming problems, let us now see if we can, in some general way, characterize these problems. They possess the following common features:
>
> (a) Multi-stage processes are involved.
> (b) At each stage, the state of the process is described by a small number of parameters.
> (c) The effect of a decision at any stage is to transform this set of parameters into a similar set.
>
> We have purposely left the description a bit vague, since we feel that it is the spirit of the problem rather than the letter that is significant. A certain amount of ingenuity is always required in attacking new questions, and no amount of axiomatics and rigid prescriptions can ever banish it.
>
> Add to the above the following simple
>
> PRINCIPLE OF OPTIMALITY. *An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision,*
> and we have the basic ingredients of the theory of dynamic programming.
>
> The rest is mathematics and experience.
>
> Bellman [1954c, p. 285][1]

[1]Reprinted by permission. Copyright 1954 INFORMS. Bellman, R. 1954. Some applications of the theory of dynamic programming – A review. *Operations Research,* 2(3) 275–288. The Institute for Operations Research and the Management Sciences, 7240 Parkway Drive, Suite 300, Hanover, Maryland 21076, USA.

It is interesting that Bellman's position on dynamic programming has crystallized right from start. Thus tracing his thinking on the subject through his numerous books and articles, one is struck by the fact that as early as 1953 his view on dynamic programming was already fully formed, and that the principle already assumed its central position in the theory.

The principle was first introduced in a technical report (Bellman 1953, p. 5) as a *requirement*, and the functional equation was deduced as a consequence of this requirement. Then, in two papers published in 1954, the principle was no longer treated as a requirement, but as an immediately perspicuous mathematical truth, whose validity is guaranteed by the structure of the *assumed model*, which I was at pains to stress, was a *final state model.* In both cases the functional equation was presented as a consequence of the principle:

> *...As stated above, the basic idea of the theory of dynamic programming is that of viewing an optimal policy as one determining the decision required at each time in terms of the current state of the system. Following this line of thought, the basic functional equations given below describing the quantitative aspects of the theory are uniformly obtained from the following intuitive Principle of Optimality ...*
>
> Bellman [1954a, p. 504][2]

> *...To obtain a functional equation governing the process we use the following obvious Principle of Optimality ...*
>
> Bellman [1954b, p. 47]

The same framework was used again in a later paper, Bellman [1957b, p. 281].

Surely, Bellman's persistent employment of the final state model as a mathematical setting for the formulation of the *Principle of Optimality* could not have been an accident. Indeed, this was a deliberate choice, as amply borne out by the fact that it was precisely the *final state model* that served as a medium for the presentation of the *Principle of Optimality* and dynamic programming in his first book; a book which no doubt amounts to Bellman's credo on the subject. The reason that the final-state model was used in this capacity is quite simple. As shown in the preceding section, the fundamental validity of the *Principle of Optimality* is immediately evident in this context and the simplicity of the model enables a straightforward formulation of the essential ingredients of the theory.

It must be emphasized, however, that despite it having been formulated in the framework of the final-state model, the *Principle of Optimality* was nevertheless construed as holding for other models as well. This is unmistakably brought out by the assertion

[2]Quoted by permission from the *American Mathematical Society,* 201 Charles Street, Providence, RI 0294-2294, USA.

. . . The mathematical transliteration of this simple principle will yield all the functional equations we shall encounter throughout the remainder of the book . . .

Bellman [1957a, p. 83]

and reinforced later in the discussion, where in the course of analyzing a multistage stochastic game problem, Bellman states

. . . That this principle is valid for one-person processes where we are attempting to maximize a return, or minimize a 'cost' is clear by contradiction. Since its validity may not be so obvious for game processes, let us present a brief proof for the sake of completeness . . .

Bellman [1957a, p. 291]

Bellman then proceeds to invoke the principle in the derivation of the dynamic programming functional equation in question.

It is also worth noting that Bellman was fully aware that the simplicity of the idea articulated by the *Principle of Optimality* might raise doubts about its validity. Thus, interrupting the discussion in which the derivation of the functional equation was demonstrated for the first time in the book, Bellman suggests the following:

. . . This observation, simple as it is, is the key to all our subsequent mathematical analysis. It is worthwhile for the reader to pause here a moment and make sure that he really agrees with this observation which has the deceptive simplicity of a half-truth . . .

Bellman [1957a, p. 8]

Another typical trait of Bellman's style of presentation is the direct manner in which optimization problems were translated into functional equations. Bellman rarely went into the intermediate modelling stages outlined in this book. Normally he would derive the functional equation almost directly from the classical formulation of the problem. For example, given a problem of the form

$$\underset{(x_1,\dots,x_N)}{\text{opt}} \sum_{n=1}^{N} q_n(x_n) \tag{13.47}$$

$$\sum_{n=1}^{N} a_n x_n \leq b \ , \ x_n \geq 0, \ n = 1, 2, \dots, N \tag{13.48}$$

Bellman would typically argue as follows.

Let $f_n(s)$ denote the optimal return from stage n on, assuming that there are s units of resource available. Now, suppose that at that point the decision is to allocate x units of the amount of resource in question.

Then, the *Principle of Optimality* implies that the best return that would be yielded by the remaining stages is equal to $f_{n+1}(s - a_n x_n)$, which means that the total return given s and x would be equal to

$$q_n(x_n) + f_{n+1}(s - a_n x) \tag{13.49}$$

Now, naturally, the decision x is made with a view to optimize this return. Thus,

$$f_n(s) = \operatorname*{opt}_{x} \{q_n(x) + f_{n+1}(s - a_n x)\} \tag{13.50}$$

where the optimization is over all the feasible values of x pertaining to state s at stage n.

It should be pointed out that the transition from (13.49) to (13.50), which is regarded by Bellman as natural and obvious, in fact involves an implicit appeal to what I term in this book the *Principle of Conditional Optimization.* That is, if $f_n(s, x)$ is defined as the optimal return from stage n onwards — given that at stage n the state is s and the decision is x — then clearly the *Principle of Optimality* entails that

$$f_n(s, x) = q_n(x) + f_{n+1}(s - a_n x) \tag{13.51}$$

Since the *Principle of Conditional Optimization* yields

$$f_n(s) = \operatorname*{opt}_{x} f_n(s, x) \tag{13.52}$$

the inference is that (13.50) is true.

I also call attention to the fact that this outline clearly indicates that Bellman understood the term ‘initial decision’ as ‘initial feasible decision’, that is, the decision x in (13.49) is any *feasible* decision pertaining to state s at stage n.

To conclude this discussion, I wish to point out that what is perhaps most intriguing in all this is that, despite the critical comment leveled at the *Principle of Optimality* and his formulation of dynamic programming in general, Bellman never modified the phrasing of the principle, nor his treatment of the theory nor, as can best be established, did he “publicly” react to this criticism. Why Bellman did not comment on this criticism in his books and articles is anybody’s guess. In any event, this is most unfortunate because in private conversations Bellman’s response to the criticism not only shed interesting light on his conception of dynamic programming, but also on his thinking on the philosophy of mathematics and science. The reader is referred to Bellman’s [1984] autobiography for an account touching on these issues.

13.9 Summary

The *Principle of Optimality* gives expression to the guiding idea underlying dynamic programming as an optimization method. Stated in terms of the multistage decision model outlined in this book, the principle links — through the mediation of the *Principle of Conditional Optimization* — the optimal solutions of the conditional problems with the optimal solutions of the modified problems that they generate. This relation is mirrored in the functional equation of dynamic programming. In this sense the principle enunciates the Markovian property of the optimal solutions.

I emphasized that the *Principle of Optimality* does not make a statement about some general optimality criterion. This statement makes sense only in the context of the assumed multistage decision model and its decomposition scheme. Thus, any attempt at the principle's interpretation, or assessment of its validity, must be confined to this framework. I come back to this issue in *Chapter 16.*

We have seen that in this framework the *Principle of Optimality* emerges as the natural sufficient condition for the validity of the dynamic programming functional equation. Moreover, having established a formal link between dynamic programming's formulation of an optimization problem and the classic prototype — *Problem P*— I established that any instance of *Problem P* has a dynamic programming formulation which obeys the *Principle of Optimality*. I thereby demonstrated the unquestionable rigor of a dynamic programming paradigm based on the *Principle of Optimality.*

I also discussed Bellman's approach to dynamic programming notably his exposition of it. I believe that a discussion on this matter firmly belongs in the broader discussion on the "art" aspects of dynamic programming because, in my view Bellman's formulation of dynamic programming, with the pair *Final State Model* — *Principle of Optimality* as its cornerstone, is indeed a work of art.

13.10 Post Script: Pontryagin's Maximum Principle

On a number of occasions I was asked to state my position on how *Pontryagin's Maximum Principle* would be understood in the context of a sequential decision process and, to be sure to include a discussion on this topic, should I ever contemplate a second edition of the book. So here it is in a nutshell.

Stated in the context of the multistage decision problem formulated in this book *Pontryagin's Maximum Principle* is concerned with the optimality conditions satisfied by the decision variables. Specifically, it addresses the question: what optimality conditions must be met by the problem

$$f_n(s) = \operatorname*{opt}_{x\in D(n,s)} f_n(s,x) \ , \ n \in \mathbb{N}, s \in S_n \tag{13.53}$$

recalling that

$$\begin{aligned} f_n(s) &:= \text{optimal return associated with the modified problem} \\ &\quad\ \text{at } (n,s). \qquad (13.54) \\ f_n(s,x) &:= \text{optimal return associated with the conditional problem} \\ &\quad\ \text{at } (n,s,x). \qquad (13.55) \end{aligned}$$

In other words, the question is: what conditions should the optimal value of x in (13.53) satisfy?

If the problem defined by (13.53) is a "classical" (unconstrained) optimization problem, then obviously the optimal value of x is required to satisfy the classical *first-order optimality condition*: the derivative of $f_n(s,x)$ with respect to x must be equal to zero at the optimum. Hence,

$$\frac{d}{dx} f_n(s,x) = 0 \qquad (13.56)$$

Now, if the *Principle of Optimality* holds, then

$$f_n(s,x) = \rho(n,s,x,f_{n+1}(T(n,s,x))) \qquad (13.57)$$

in which case (13.56) yields

$$\frac{d}{dx}\rho(n,s,x,f_{n+1}(T(n,s,x))) = 0 \qquad (13.58)$$

The *Maximum Principle* then asserts that, under the above conditions, the optimal value of x must satisfy the first-order optimality conditions specified by (13.58).

Consider for example the following familiar additive dynamic programming functional equation:

$$f_n(s) = \max_{x\in D(n,s)} \{w(n,s,x) + f_{n+1}(T(n,s,x))\} \ , \ n \in \mathbb{N}, s \in S_n \qquad (13.59)$$

In this case

$$\rho(n,s,x,f_{n+1}(T(n,s,x))) = w(n,s,x) + f_{n+1}(T(n,s,x)) \qquad (13.60)$$

hence the first order condition is

$$\frac{d}{dx}\{w(n,s,x) + f_{n+1}(T(n,s,x))\} = 0 \qquad (13.61)$$

Applying the chain rule we obtain

$$\frac{d}{dx}\{w(n,s,x) + f_{n+1}(T(n,s,x))\} = w'(n,s,x) + f'_{n+1}(y)T'(n,s,x) \quad (13.62)$$

where

$$w'(n,s,x) := \frac{d}{dx}w(s,x) \tag{13.63}$$

$$T'(n,s,x) := \frac{d}{dx}T(n,s,x) \tag{13.64}$$

$$f'_{n+1}(y) := \frac{d}{dy}f_{n+1}(y)\ ,\ y = T(n,s,x) \tag{13.65}$$

In short,

Maximum Principle:
If the problem defined by (13.59) is a "classical" optimization problem, then the first order optimality condition is

$$\frac{d}{dx}w(n,s,x) + \frac{d}{dx}T(n,s,x)\frac{d}{dy}f_{n+1}(y) = 0\ ,\ y = T(n,s,x) \tag{13.66}$$

For instance, consider the functional equation examined in *Example 5.3.3* namely

$$f_n(s) = \max_{0\le x\le s}\{\beta^{n-1}\sqrt{x} + f_{n+1}(s-x)\} \tag{13.67}$$

for $1 \le n < N, s \in S_n = [0,r]$, with

$$f_N(s) := \beta^{N-1}\sqrt{s}, \forall s \in [0,r] \tag{13.68}$$

So here $w(n,s,x) = \beta^{n-1}\sqrt{x}$, $D(n,s) = [0,s]$ and $T(n,s,x) = s - x$. Hence,

$$\frac{d}{dx}w(n,s,x) = \frac{\beta^{n-1}}{2\sqrt{x}}\ ,\ x > 0 \tag{13.69}$$

$$\frac{d}{dx}T(n,s,x) = -1 \tag{13.70}$$

Thus, for $n = N-1$ the first-order optimality condition is

$$\frac{\beta^{N-2}}{2\sqrt{x}} + (-1)\frac{d}{dy}\beta^{N-1}\sqrt{y} = 0\ ,\ y = s - x \tag{13.71}$$

$$\frac{1}{2\sqrt{x}} = \frac{\beta}{2\sqrt{y}} \tag{13.72}$$

$$\frac{1}{x} = \frac{\beta^2}{s-x} \tag{13.73}$$

$$x^* = \frac{s}{1+\beta^2}\ ,\ s \in [0,r] \tag{13.74}$$

hence

$$f_{N-1}(s) = f_{N-1}(s,x^*)\ ,\ s \in [0,r] \tag{13.75}$$

$$= \beta^{N-2}\sqrt{x^*} + f_N(s-x^*) \tag{13.76}$$

$$= \beta^{N-2}\sqrt{\frac{s}{1+\beta^2}} + \beta^{N-1}\sqrt{s - \frac{s}{1+\beta^2}} \tag{13.77}$$

$$= \beta^{N-2}\sqrt{s(1+\beta^2)} \tag{13.78}$$

This exercise is repeated for $n = N - 2$, and then for $n = N - 3$ ... until a pattern is identified following which we continue by induction.

And so, if we compare this approach with the approach employed in *Example 5.3.3* to solve this problem, it turns out that the role of the *Maximum Principle* is to deploy the *chain rule* of differentiation when the right hand side of the functional equation is solved analytically.

The point to note then is that the *Maximum Principle* in effect assumes the validity of a property that is enunciated by ... the *Principle of Optimality.* Or, in other words, the *Maximum Principle* makes an implicit appeal to the *Principle of Optimality.* Also note that it applies to cases where first-order optimality conditions are relevant.

I should point out, though, that the above version of the *Maximum Principle* is rather simple because, in the above example, no *functional constraints* are imposed on the decision variable, and the transition function T is very simple indeed.

In cases where $D(n, s)$ and T are more intricate, it might prove difficult to obtain a simple closed-form expression for the first-order optimality conditions. Also, certain convexity conditions will have to be imposed to insure that the required optimality conditions are also sufficient for a *global* optimum.

The *Maximum Principle* has a prominent role in optimal control theory where the literature on it is extremely rich (e.g. Pontryagin et al. 1962, Kirk 1970, Sethi and Thompson 2000, Geering 2007, Ross 2009). There is also a rich literature on its "discrete time" version (e.g. Hwang and Fan 1967, Blot 2009).

14

Forward Decomposition

14.1 Introduction

The family of modified problems — forming part of the dynamic programming models studied thus far — gives expression to the optimization of the *remaining* part of the sequential decision process where a number of decisions have already been made. Consequently, the functional equations that we encountered to this point, relate modified problems to their immediate *successors.*

In this framework an immediate successor of state $s \in S$ is a state s' that can be reached from state s in a single state transition. Thus, in the context of the multistage model, if the process is at stage n and the state s is observed, then an immediate successor of s is any state $s' \in S$ such that $s' = T(n, s, x)$ for some decision $x \in D(n, s)$.

Similarly, in the context of our sequential decision model, an immediate successor of state $s \in S$ is any state s' such that $s' = T(s, x)$ for some decision $x \in D(s)$.

This orientation is clearly manifested in the structure of the two generic functional equations associated with these two types of models:

Multistage decision model:

$$f_n(s) = \operatorname*{opt}_{x \in D(n,s)} \rho(n, s, x, f_{n+1}(T(n, s, x))) \ , \ n \in \mathbb{N}, s \in S \tag{14.1}$$

Sequential decision model:

$$f(s) = \operatorname*{opt}_{x \in D(s)} \rho(s, x, f(T(s, x))) \ , \ s \in S' \tag{14.2}$$

Expressing as they do the optimal return associated with a modified problem as the optimal returns associated with its immediate *successors*, the orientation of these functional equations is unmistakably "backward". Note that the orientation is "backward" relative to the direction stipulated by the state transition function T which depicts the dynamics of the decision-making process. And so, if the functional equation is seen as the recipe for computing say the value of $f_j(s)$ according to (14.1), then the values of $f_{j+1}(s'), s' = T(j, s, x), x \in D(j, s)$ are computed first. Effectively then, the functional equation is solved for $j = N, N-1, \dots, 1$ — in a "backward" order.

The roots of this "backward" orientation lie of course in the manner in which we decompose the *objective function* of the decision process, and in the property of *separability.* To have a clearer understanding of this point let us take another look at the concepts of *decomposition* and *separability.* This will shed more light on the *backward* decomposition scheme and it will establish a framework for the formulation of a "forward" decomposition scheme.

However, prior to this, I have to explain why the need for a *forward* scheme arises at all. To this end I consider a simple example, which illustrates that a *forward* scheme complements the function of a *backward* scheme.

14.1.1 Example

Consider the problem

$$\underset{\mathbf{x}}{\operatorname{opt}}\, g(\mathbf{x}) := \left[a_1x_1 + \left[a_2x_2 + [a_3x_3 + [a_4x_4 + [a_5x_5]^p]^p]^p\right]^p\right]^p \tag{14.3}$$

subject to

$$\beta(\mathbf{x}) := \left[b_1x_1 + \left[b_2x_2 + \left[b_3x_3 + \left[b_4x_4 + [b_5x_5]^2\right]^2\right]^2\right]^2\right]^2 \leq B \tag{14.4}$$

$$x_n \in \{0, 1, 2, \dots\}\ ,\ n = 1, 2, \dots, k \tag{14.5}$$

where all the parameters are positive integers.

Recall that since the obvious decomposition scheme for β relies on a non-associative binary operation that is executed from right to left, we need to reverse the indices of the decision variables and rewrite the problem as follows:

$$\underset{\mathbf{y}}{\operatorname{opt}}\, h(\mathbf{y}) := \left[c_5y_5 + \left[c_4y_4 + [c_3y_4 + [c_2y_2 + [c_1y_1]^p]^p]^p\right]^p\right]^p \tag{14.6}$$

subject to

$$\gamma(\mathbf{y}) := \left[d_5y_5 + \left[d_4y_4 + \left[d_3y_3 + \left[d_2y_2 + [d_1y_1]^2\right]^2\right]^2\right]^2\right]^2 \leq B \tag{14.7}$$

$$y_n \in \{0, 1, 2, \dots\}\ ,\ n = 1, 2, \dots, k \tag{14.8}$$

where $c_j := a_{5-j+1}$, $d_j := b_{5-j+1}, j = 1, 2, \dots, 5$, and y_j plays the role of x_{5-j+1}.

So, rearranging the terms, we obtain

$$\underset{\mathbf{y}}{\operatorname{opt}}\, h(\mathbf{x}) = \left[[[[[c_1y_1]^p + c_2y_2]^p + c_3y_3]^p + c_4y_4\right]^p + c_5y_5\right]^p \tag{14.9}$$

subject to

$$\gamma(\mathbf{y}) := \left[\left[\left[\left[[d_1y_1]^2 + d_2y_2\right]^2 + d_3y_3\right]^2 + d_4y_4\right]^2 + d_5y_5\right]^2 \leq B \tag{14.10}$$

To decompose the constraint $\gamma(\mathbf{y}) \leq B$, we define

$$s_{n+1} = T(n, s_n, y_n) := [s_n]^2 + d_ny_n\ ,\ n = 1, 2, \dots, 5 \tag{14.11}$$

with $s_1 := 0$, observing that by construction $[s_6]^2 = \gamma(\mathbf{y})$, hence the constraint can be written as $[s_6]^2 \leq B$.

Now, it is immediately clear that solving this problem is no easy task.

Because, as explained in §*11.10,* to lend itself to a straightforward "backward" decomposition, an objective function ought to allow an evaluation from "right to left". But here the objective function calls for an evaluated from "left to right".

So, the rationale for proposing the concept of "forward dynamic programming" is obvious. It is important to have on hand a means for decomposing the objective function in a forward — and not only a backward — direction, namely from "left to right".

14.2 Function Decomposition

Functions acting as objective functions and as constraints of optimization problems can often be represented as *composite functions*. In turn a *decomposition scheme* is a device describing the structure of a composite function.

Stated formally, let u be a real-valued function on Y^k where Y is some given set and let $\mathcal{K} := \{1, 2, \dots, k\}$. Then, u can often be expressed by means of a *binary operation* $\oplus$ and a real-valued function v on $\mathcal{K} \times Y$ as follows:

$$u(y_1, \dots, y_k) = v(1, y_1) \oplus v(2, y_2) \oplus \cdots \oplus v(k, y_k) \tag{14.12}$$

The most prevalent case is that where the function u is *additive,* namely where $\oplus = +$. Other common instances — discussed in *Chapter 10* — are $\oplus = \times$, $\oplus = \lfloor$ and $\lceil$.

An important property of these four instances is that $\oplus$ is *associative*, that is,

$$(a \oplus b) \oplus c = a \oplus (b \oplus c) \tag{14.13}$$

This dispenses with the need to parenthesize the right hand side of (14.12).

One of the implication of this property is that the function u can be decomposed in two different ways, that is, we can use two decomposition schemes:

$$\vec{u}_j\ (y_1, \dots, y_j) := v(1, y_1) \oplus \cdots \oplus v(j, y_j)\ ,\ j \in \mathcal{K} \tag{14.14}$$

$$\overleftarrow{u}_j\ (y_j, \dots, y_k) := v(j, y_j) \oplus \cdots \oplus v(k, y_k)\ ,\ j \in \mathcal{K} \tag{14.15}$$

where $\vec{u}_j$ is a real valued function on Y^k and $\overleftarrow{u}_j$ is a real valued function on Y^{k-j+1}.

By construction,

$$\vec{u}_k = u = \overleftarrow{u}_1 \tag{14.16}$$

$$\vec{u}_j\ (y_1, \dots, y_j) = \vec{u}_{j-1}\ (y_1, \dots, y_{j-1}) \oplus v(j, y_j)\ ,\ j \in \mathcal{K} \backslash \{1\} \tag{14.17}$$

$$\overleftarrow{u}_j\ (y_j, \dots, y_k) = v(j, y_j) \oplus \overleftarrow{u}_{j+1}\ (y_{j+1}, \dots, y_k)\ ,\ j \in \mathcal{K} \backslash \{k\} \tag{14.18}$$

For obvious reasons I refer to $(\oplus, \vec{u})$ as a *forward* scheme and to $(\oplus, \overleftarrow{u})$ a *backward* scheme.

Note that if $\oplus$ is non-associative, a convention for the evaluation of a composite expression is required otherwise (14.12) is not well-defined. For example, *division* $(\oplus = \div)$ is non-associative, hence the expression $2 \div 3 \div 4$ is not well-defined:

$$2 \div (3 \div 4) = \frac{8}{3} \neq (2 \div 3) \div 4 = \frac{1}{6} \tag{14.19}$$

And so, if u admits of the representation given in (14.12), we say that u is *separable* and we term $\oplus$ a *composition function.*

In short, a separable u and an associative composition function $\oplus$ provide two decomposition schemes. And a separable but non-associative u, provides only one decomposition scheme. The scheme's orientation would depend on the rule invoked to evaluate the composite expression. Thus, if the expression is evaluated from *right to left*, then a *backward* scheme would be used. Whereas, if the expression is evaluated from *left to right*, then a *forward* scheme would be used.

There are cases where u cannot be decomposed by means of a "standard" binary function, but it can be decomposed by means of slightly more elaborate schemes, such as

$$\vec{u}_k = u = \overleftarrow{u}_1 \tag{14.20}$$
$$\vec{u}_j\ (y_1, \dots, y_j) = \vec{\rho}\ (j, y_j, \vec{u}_{j-1}\ (y_1, \dots, y_{j-1}))\ ,\ j \in \mathcal{K} \backslash \{1\} \tag{14.21}$$
$$\overleftarrow{u}_j\ (y_j, \dots, y_k) = \overleftarrow{\rho}\ (j, y_j, \overleftarrow{u}_{j+1}\ (y_{j+1}, \dots, y_k))\ ,\ j \in \mathcal{K} \backslash \{k\} \tag{14.22}$$

where $\vec{\rho}$ and $\overleftarrow{\rho}$ are real valued functions on $K \times Y \times \mathbb{R}$.

Note that the binary representations are instances where

$$\vec{\rho}\ (j, y_j, a) = a \oplus v(j, y_j)\ ,\ j \in \mathcal{K} \setminus \{1\} \tag{14.23}$$
$$\overleftarrow{\rho}\ (j, y_j, a) = v(j, y_j) \oplus a\ ,\ j \in \mathcal{K} \setminus \{k\} \tag{14.24}$$

There are cases lending themselves to only one of the above schemes. For instance, consider this beauty:

$$u(y_1, y_2, y_3, y_4, y_5) = \left[\left[\left[\left[[y_1]^1 + y_2\right]^2 + y_3\right]^3 + y_4\right]^4 + y_5\right]^5 \tag{14.25}$$

The construction of a forward decomposition scheme is straightforward here:

$$\vec{u}_1\ (y_1) := [y_1]^1 \tag{14.26}$$
$$\vec{u}_2\ (y_1, y_2) := \left[[y_1]^1 + y_2\right]^2 \tag{14.27}$$

$$\vec{u}_3\,(y_1,y_2,y_3) := \left[\left[[y_1]^1 + y_2\right]^2 + y_3\right]^3 \tag{14.28}$$

$$\vec{u}_4\,(y_1,y_2,y_3,y_4) := \left[\left[\left[[y_1]^1 + y_2\right]^2 + y_3\right]^3 + y_4\right]^4 \tag{14.29}$$

$$\vec{u}_5\,(y_1,y_2,y_3,y_4,y_5) := u(y_1,y_2,y_3,y_4,y_5) \tag{14.30}$$

in which case

$$\vec{\rho}\,(j,y_j,a) = [a+y_j]^j \ \ ,\ j=2,3,4,5 \tag{14.31}$$

But, there seems to be no obvious way (other than employing the trivial decomposition scheme) to construct a backward decomposition scheme here. Still, by renaming the variables we reverse their "order" so that a backward scheme is readily available. To see how this works, let $z_j = y_{k-j+1}$. Then by construction,

$$u(y_1,y_2,y_3,y_4,y_5) = u(z_5,z_4,z_3,z_2,z_1) \tag{14.32}$$

$$= \left[\left[\left[[z_5+z_4]^2 + z_3\right]^3 + z_2\right]^4 + z_1\right]^5 \tag{14.33}$$

$$= \left[z_1 + \left[z_2 + \left[z_3 + [z_4+z_5]^2\right]^3\right]^4\right]^5 \tag{14.34}$$

So, we can set

$$\overleftarrow{u}_5\,(z_5) := z_5 \tag{14.35}$$

$$\overleftarrow{u}_4\,(z_4,z_5) := [z_4+z_5]^2 \tag{14.36}$$

$$\overleftarrow{u}_3\,(z_3,z_4,z_5) := \left[z_3 + [z_4+z_5]^2\right]^3 \tag{14.37}$$

$$\overleftarrow{u}_2\,(z_2,z_3,z_4,z_5) := \left[z_2 + \left[z_3 + [z_4+z_5]^2\right]^3\right]^4 \tag{14.38}$$

$$\overleftarrow{u}_1 := u \tag{14.39}$$

in which case

$$\overleftarrow{\rho}\,(j,z_j,a) = [a+z_j]^{k-j+1} \ \ ,\ j=1,2,3,4 \tag{14.40}$$

Recall that the separability requirement dictates that the objective function g of the multistage process be decomposed by means of a scheme $(\rho,\{g_n\})$ such that

$$g_1 \equiv g \tag{14.41}$$

$$g_n(s_n,x_n,x_{n+1},\dots,x_N) = \rho(n,s_n,x_n,g_{n+1}(s_{n+1},x_{n+1},\dots,x_N)) \tag{14.42}$$

observing that here we express g_n in terms of g_{n+1}, hence the appellation "backward".

To sum up, in the case of backward decomposition, the modified problems are concerned with the remaining part of the original process, namely the section extending from some postulated current state to the "end" of the process. In contrast, in the case of a forward decomposition, the modified problems are concerned with the part of the original process which extends from the initial state to some postulated current state.

So, by dynamic programming conventions, a decomposition where the modified problems are defined in terms of the *initial* segments of the decision process, is carried out in the "opposite direction". In turn, the functional equation of a forward formulation relates the modified problems to their immediate *predecessors*.

Thus, in the case of the multistage decision process, when we say that our objective is to formulate a forward decomposition scheme we mean a scheme $(\vec{\rho}, \{\vec{g}_n : n = 1, 2, \dots, N+1\})$ such that

$$\vec{g}_{N+1} = g \tag{14.43}$$

$$\vec{g}_n(s_1, x_1, \dots, x_{n-1}) = \vec{\rho}(n, s_{n-1}, x_{n-1}, g_{n-1}(s_1, x_1, \dots, x_{n-2})) \tag{14.44}$$

where as usual $s_n = T(n-1, s_{n-1}, x_{n-1})$.

Observe that by construction $\vec{g}_n$ is defined on $S_1 \times \mathbb{D}^{n-1} \times S_n, n > 1$, whereas $\vec{g}_1$ is defined on S_1.

We construe $\vec{g}_n(s_1, x_1, x_2, \dots, x_{n-1})$ as the return accumulated through the first $n-1$ stages of the process given that the initial state of the process was s_1 and that decision x_j was made in stage j, $1 \leq j < n$. Since s_n is uniquely determined by $T(n-1, s_{n-1}, x_{n-1})$, it follows that s_n is implied by $s_1, x_1, x_2, \dots, x_{n-1}$.

For example, if the objective function is additive, namely

$$g(s_1, x_1, x_2, \dots, x_N) = \sum_{n=1}^{N} w(n, s_n, x_n) \tag{14.45}$$

the obvious forward decomposition scheme is

$$\vec{g}_n(s_1, x_1, x_2, \dots, x_{n-1}) = \sum_{j=1}^{n-1} w(j, s_j, x_j) \ , \ 1 < n \leq N+1 \tag{14.46}$$

$$\vec{\rho}(n, s_{n-1}, x_{n-1}, a) = w(n-1, s_{n-1}, x_{n-1}) + a \tag{14.47}$$

for $1 < n \leq N+1$, with $\vec{g}_1(s) = 0, \forall s \in S_1$.

In words, the return accumulated over the first $n-1$ stages is the sum of (a) the return accumulated over the first $n-2$ stages and (b) the return generated at stage $n-1$, namely $w(n-1, s_{n-1}, x_{n-1})$.

My main thrust in this chapter is to analyze the relationship between the two decomposition paradigms. To explain why the two paradigms are, for the most part, equivalent and why they can occasionally be inequivalent. Or, in other words, to explain why these two paradigms can occasionally generate different state variables for the same problem. Thus, I shall identify the essential characteristics of those (pathologic) cases where the two formulations are not equivalent.

I shall also address the following practical question: given a choice between the two schemes, under what conditions is it preferable to use a forward decomposition scheme and under what conditions is a backward scheme preferable?

However, before I can take up these questions the following remarks are in order.

· It is important to realize that the "forward vs backward" classification must be based on an exact, general, formal definition. Because, in general a decomposition scheme as such is neither strictly "forward" nor strictly "backward". Indeed, the popular terms "forward dynamic programming" and "backward dynamic programming" are not sufficiently precise, what is more they are used rather loosely in the literature. This may also explain why questions regarding the relationship between these important concepts are rarely discussed in the literature.

· A proper definition, hence a correct use, of the terms "backward" and "forward" mandates *direction referencing.* In the case of our prototype dynamic programming model, direction is determined by the transition function T. This means that the terms "backward" and "forward" are well-defined even if the sequential decision process under consideration is *cyclic* or *non-truncated.*

· It must also be appreciated that for the same reason, these two properties are not absolute. Because, their definition is contingent on the structure of the postulated transition function, their role can often be reversed simply by a reformulation of the dynamic programming model yielded by a reversal in the direction of the dynamics as specified by T. It is therefore important that these terms be used with care.

· The terms *backward* and *forward* can describe a number of specific generic dynamic programming constructs, the most important ones being the *decomposition scheme*, *composition function* and the *functional equation.* As we shall soon see, it is the decomposition scheme that dictates the forward/backward nature of the other elements of the model. But it should be noted that some of the elements of the dynamic programming models are not affected by the structure of the decomposition scheme.

These remarks make plain why a proper account of "forward dynamic programming" and "backward dynamic programming", calls for a formal treatment of these concepts. In our case this should be straightforward because we are in a position to anchor such a treatment in the framework of the prototype dynamic programming models set out in *Chapter 4* and *Chapter 10.* As we shall see, these models will provide the required setting for a pre-

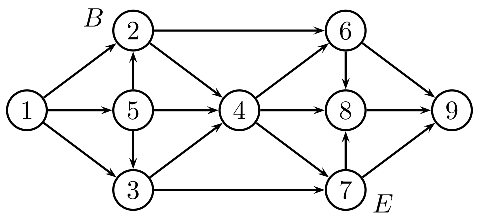


Figure 14.1: Finding the shortest path from B to E

cise analysis and description of the "backward" and "forward" schemes and will thus enable an edifying explanation of the relationship between them.

To set the stage I begin with an example that will bring out the fundamental difference between "forward" and "backward" to thus equip us with an unambiguous definition of these schemes.

14.2.1 Example

Consider the acyclic directed graph shown in Figure 14.1. It consists of $k = 9$ nodes. The arcs' lengths are deliberately omitted. The task is to determine the shortest path from node $B = 2$ to node $E = 7$. Let $\mathcal{K} := \{1, 2, \dots, k\}$.

Formally then our problem of interest — the TARGET PROBLEM — is: find the shortest path from node B to node E.

Clearly, there are two obvious ways to create *modified* problems out of the given target problem:

- BACKWARD DECOMPOSITION: Regarding the destination node (E) as *fixed* and treating the home node (B) as a *parameter*, then a typical modified problem would be the following: what is the shortest path from node $j \in \mathcal{K}$ to node E?
- FORWARD DECOMPOSITION: Regarding the origin node (B) as *fixed*, and treating the destination node (E) as a *parameter*, then a typical modified problem would be the following: what is the shortest path from node B to node $j \in \mathcal{K}$?

Let us now derive the two dynamic programming functional equations that are based on these two types of decomposition. For this purpose let

$$
\begin{aligned}
w(i,j) &:= \text{length of arc } (i,j)\\
suc(j) &:= \text{set of immediate successors of node } (j)\\
pre(j) &:= \text{set of immediate predecessors of node } (j)\\
f(j) &:= \text{length of the shortest path from node } j \text{ to node } E\\
\vec{f}(j) &:= \text{length of the shortest path from node } B \text{ to node } j
\end{aligned}
$$

and for simplicity assume that the length of a path is equal to the *sum* of the arcs' lengths on the path.

By construction, in the case of the backward formulation the objective is to determine the value of $f(B)$ whereas in that of the forward formulation the objective is to determine the value of $\vec{f}(E)$.

Then clearly the functional equation induced by the backward decomposition scheme is as follows:

$$f(j) = \min_{n \in suc(j)} \{w(j,n) + f(n)\} \ , \ j \in \mathcal{K}, suc(j) \neq \varnothing \tag{14.48}$$

observing that if $suc(j) = \varnothing$, then node E cannot be reached from node j. In this case we let $f(j) = \infty$.

Note that if the network is acyclic then clearly by definition $f(E) = 0$ and this is also the case if the network is cyclic but the arcs' lengths are not negative.

And, the dynamic programming functional equation induced by the forward decomposition scheme is as follows:

$$\vec{f}(j) = \min_{n \in pre(j)} \{\vec{f}(n) + w(n,j)\} \ , \ j \in \mathcal{K}, pre(j) \neq \varnothing \tag{14.49}$$

observing that if $pre(j) = \varnothing$, node j cannot be reached from node B. In this case we let $\vec{f}(j) = \infty$.

Note that if the network is *acyclic* then clearly by definition $\vec{f}(B) = 0$ and this is also the case if the network is cyclic but the arcs' lengths are not negative.

Insofar as the problem featured in this example is concerned, the two schemes are clearly similar, so there seems to be no obvious reason to prefer one to the other. The only reason for preferring one to the other would arise in situations where the raw data would be given in a particular format so as to prove more suitable for one rather than the other.

For instance, if the arcs of the network would be specified by a list of sets stipulating the immediate successors of the nodes, then it would be easier to apply a backward scheme. On the other hand, if such a list would specify the immediate predecessors of the nodes, then is would be easier to apply forward scheme.

Otherwise, in this example, the two decomposition schemes are practically equivalent. □

This clear, I now want to demonstrate the precise opposite. Thus, my next example features a case where constructing a backward scheme is no more than a trivial task but the construction of a suitable (non-trivial) forward scheme presents a Herculean task if not a "mission impossible".

14.2.2 Example

Consider the infinite horizon case where the objective function under consideration is of the form

$$g(s_1, x_1, x_2, x_3, x_4, \dots) = x_1 + \sqrt{x_2 + \sqrt{x_3 + \sqrt{x_4 + \sqrt{\cdots}}}} \tag{14.50}$$

Constructing a backward decomposition scheme here is straightforward as we simply let

$$g_n(s_n, x_n, x_{n+1}, \dots) := x_n + \sqrt{x_{n+1} + \sqrt{x_{n+2} + \sqrt{\cdots}}} \tag{14.51}$$

and

$$\rho(n, s, x, a) = x + \sqrt{a} \tag{14.52}$$

for $n = 1, 2, \dots$, observing that by construction

$$g_n(s_n, x_n, x_{n+1}, \dots) = \rho(n, s_n, x_n, g_{n+1}(s_{n+1}, x_{n+1}, x_{n+2}, \dots)) \tag{14.53}$$

as required, and that ρ is strictly increasing with its last argument, hence the scheme is *Markovian.*

The functional equation induced by this scheme is then as follows:

$$f_n(s) = \operatorname*{opt}_{x \in D(n,s)} \rho(n, s, x, f_{n+1}(T(n, s, x))) \ , \ n = 1, 2, \dots \tag{14.54}$$

$$= \operatorname*{opt}_{x \in D(n,s)} \left\{ x + \sqrt{f_{n+1}(T(n, s, x))} \right\} \tag{14.55}$$

But how would g be decomposed in a "forward" manner?

Since the process in non-truncated, it is impossible to "reverse" its direction. So, suppose that we seek to exploit the fact that by definition

$$g(s_1, x_1, x_2, \dots) = \lim_{n \to \infty} \vec{g}_n(s_1, x_1, x_2, \dots, x_n) \tag{14.56}$$

where

$$\vec{g}_n(s_1, x_1, x_2, \dots, x_n) := x_1 + \sqrt{x_2 + \sqrt{x_3 + \sqrt{\cdots + \sqrt{x_n}}}} \tag{14.57}$$

for $n = 1, 2, \dots$.

The idea would be then to express $\vec{g}_{n+1}$ in terms of $\vec{g}_n$. But to see the enormity of this task, consider say $\vec{g}_4$ and $\vec{g}_3$ and try to express $\vec{g}_4$ in terms of $\vec{g}_3$:

$$\vec{g}_3(s_1, x_1, x_2, x_3) = x_1 + \sqrt{x_2 + \sqrt{x_3}} \tag{14.58}$$

$$\vec{g}_4(s_1, x_1, x_2, x_3, x_4) = x_1 + \sqrt{x_2 + \sqrt{x_3 + \sqrt{x_4}}} \tag{14.59}$$

That this is indeed a "mission impossible", becomes clear when one realizes that the values of x_1 and x_2 must not be referred to explicitly.

On first encountering this dilemma *students* tend to suggest the following scheme:

$$\vec{g}_4(s_1, x_1, x_2, x_3, x_4) = \vec{g}_3(s_1, x_1, x_2, x_3) + \sqrt{x_4} \tag{14.60}$$

which, needless to say, is invalid. To see why this is so, consider for instance the specific case where $x_1 = 5, x_2 = 7, x_3 = 3, x_4 = 1$. Here we have

$$\vec{g}_4(s_1, x_1, x_2, x_3, x_4) = \vec{g}_4(s_1, 5, 7, 3, 1) \tag{14.61}$$

$$= 5 + \sqrt{7 + \sqrt{3 + \sqrt{1}}} \tag{14.62}$$

$$= 5 + \sqrt{7 + \sqrt{3 + 1}} \tag{14.63}$$

$$= 5 + \sqrt{7 + 2} \tag{14.64}$$

$$= 5 + 3 \tag{14.65}$$

$$= 8 \tag{14.66}$$

and

$$\vec{g}_3(s_1, x_1, x_2, x_3) = \vec{g}_3(s_1, 5, 7, 3) \tag{14.67}$$

$$= 5 + \sqrt{7 + \sqrt{3}} \tag{14.68}$$

$$= 5 + \sqrt{7 + 1.7320508} \tag{14.69}$$

$$= 5 + 2.955044 \tag{14.70}$$

$$= 7.955044 \tag{14.71}$$

hence,

$$\vec{g}_3(s_1, x_1, x_2, x_3) + \sqrt{x_4} = 8.7955044 \tag{14.72}$$

Clearly, then

$$\vec{g}_4(s_1, x_1, x_2, x_3, x_4) = 8 \neq \vec{g}_3(s_1, x_1, x_2, x_3) + \sqrt{x_4} = 8.955044 \tag{14.73}$$

which means that the proposed scheme fails in this case.

As indicated above, to simply "reverse the direction" is not a proposition here because the process is nontruncated. To explain, consider the case where the process is truncated, namely where N is finite and

$$g(s_1, x_1, x_2, \dots, x_N) = x_1 + \sqrt{x_2 + \sqrt{\cdots + \sqrt{x_N}}} \tag{14.74}$$

In the case of the finite horizon formulation of the problem: We would be able to apply the backward decomposition scheme used for the infinite horizon version of the problem. Indeed, the backward scheme would remain intact, except that now N would be finite.

Likewise, to construct a forward decomposition scheme would be equally straightforward as we would simply reverse the direction of the process. The new stage $n = 1$ would be the old stage N, the new stage $n = 2$ would be the new stage $N - 1$, and in general the new stage n would be the old stage $N - n + 1$. If we call the new decision variables $x'_1, x'_2, \dots, x'_N$ then

x'_n corresponds to x_{N-n+1} and if we call the new state variable s' then s'_n corresponds to s_{N-n+1}.

The "reversed" decomposition scheme for the objective function is then as follows:

$$\vec{g}_n(s'_1, x'_1, x'_2, \dots, x'_n) = \sqrt{\cdots \sqrt{\sqrt{x'_1} + x'_2} + \cdots + x'_{n-1}} + x'_n \tag{14.75}$$

and

$$\vec{\rho}(n, s', x', a) = x' + \sqrt{a} \tag{14.76}$$

In short, in the case of the finite horizon version, converting the backward decomposition scheme for the objective function into a forward scheme amounts to no more than a change in notation resulting from the reversal in the "direction" of the process. This, however, cannot be done if the process is nontrunctated.

And more than this, the above discussion addresses only the difficulties presented by the objective function g. But the point is that to obtain a full-blown forward decomposition scheme through a reversal of the decision variables' indices, it is necessary to also reverse the direction of the transition function. And this, one need hardly point out, is not always possible simply because it may not be allowed.

For instance, suppose that the constraints under consideration are as follows:

$$\left[\cdots\left[\left[[a_1x_1 + a_2x_2]^2 + a_3x_3\right]^3 + a_4x_4\right]^4 + \cdots + x_n\right]^n \leq W \tag{14.77}$$

$$x_j \geq 0\ ,\ j = 1, \dots, n \tag{14.78}$$

where $\{a_j\}$ are some given positive parameters.

Then for the backward formulation we can let

$$s_j := \left[\cdots\left[\left[[a_1x_1 + a_2x_2]^2 + a_3x_3\right]^3 + a_4x_4\right]^4 + \cdots + a_{j-1}x_{j-1}\right]^{j-1} \tag{14.79}$$

for $j = 1, 2, \dots, n$ with $s_1 = 0$.

By construction, then,

$$s_{j+1} = T(j, s_j, x_j) := [s_j + a_jx_j]^j\ ,\ j = 1, \dots, n \tag{14.80}$$

and therefore we can let

$$D(j, s) := \left[0, \frac{W^{1/j} - s}{a_j}\right]\ ,\ s \in S_j := [0, W^{1/j}] \tag{14.81}$$

But there is no apparent formulation for the state variable of the forward formulation — obtained from the reversal of the decision variables' order — other than the trivial one, namely

$$s_j := (x_1, x_2, \ldots, x_{j-1}) \ , \ j = 2, \ldots, n \tag{14.82}$$

with s_1 = empty sequence.

Such incompatibilities between the state variable induced by the objective function and the state variable induced by the constraints are discussed in §*11.6.* □

14.2.3 Example

Consider now the classic shortest path problem assuming that all the arcs' lengths are strictly positive, the network is acyclic, the node of origin — call it node 1 — has no immediate predecessors and the destination node, call it node k — has no immediate successors. Let $\mathcal{K} = \{1, 2, \ldots, k\}$. Such a case is shown in Figure 14.2 with $B = 1$ and $E = k = 9$.

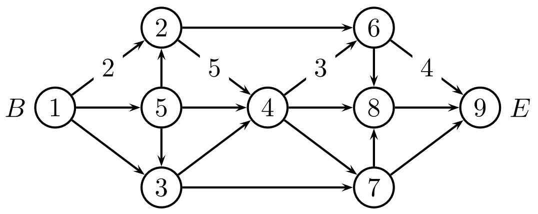


Figure 14.2: Shortest path problem from B to E

In contrast to the standard case where the length of a path is the sum of the lengths of the arcs on the path, assume that here the length of a path is computed as follows:

$$L(y_1, \ldots, y_n) = \begin{cases} w(y_1, y_2) & , \quad n = 2 \\ w(y_1, y_2) + \dfrac{1}{1 + \dfrac{1}{L(y_2, y_3, \ldots, y_n)}} & , \quad n > 2 \end{cases} \tag{14.83}$$

where $L(y_1, \ldots, y_n)$ denotes the length of a path comprising the nodes $(y_1, y_2, \ldots, y_n)$ — in this order — and $w(i,j) > 0$ denotes the length of arc (i,j).

For example, consider the path consisting of the five nodes $(1, 2, 4, 6, 9)$. Then according to the above prescription, the length of this path can be computed *recursively* (on n) as follows.

By definition

$$L(1, 2, 4, 6, 9) = w(1, 2) + \frac{1}{1 + \dfrac{1}{L(2, 4, 6, 9)}} \quad , \ (n = 5) \tag{14.84}$$

$$= 2 + \cfrac{1}{1 + \cfrac{1}{L(2,4,6,9)}} \tag{14.85}$$

So to obtain the value of $L(1,2,4,6,9)$ we have to compute the value of $L(2,4,6,9)$. To do this we invoke the definition of L given in (14.83):

$$L(2,4,6,9) = w(2,4) + \cfrac{1}{1 + \cfrac{1}{L(4,6,9)}} \quad , \ (n = 4) \tag{14.86}$$

$$= 5 + \cfrac{1}{1 + \cfrac{1}{L(4,6,9)}} \tag{14.87}$$

So invoking (14.83) again:

$$L(4,6,9) = w(4,6) + \cfrac{1}{1 + \cfrac{1}{L(6,9)}} \quad , \ (n = 3) \tag{14.88}$$

$$= 3 + \cfrac{1}{1 + \cfrac{1}{L(6,9)}} \tag{14.89}$$

and again

$$L(6,9) = 4 \ , \ (n = 2) \tag{14.90}$$

where, by backward substitution we obtain

$$L(4,6,9) = 3 + \frac{1}{1 + (1/4\ )} = \frac{19}{5} \tag{14.91}$$

$$L(2,4,6,9) = 5 + \cfrac{1}{1 + \cfrac{1}{(19/5)}} = \frac{139}{24} \tag{14.92}$$

$$L(1,2,4,6,9) = 2 + \cfrac{1}{1 + \cfrac{1}{(139/24)}} = \frac{425}{163} \tag{14.93}$$

It goes without saying that the length of a path can be computed iteratively, that is non-recursively, by utilizing the following representation of L which can be derived by inspection from the definition of L in (14.83):

$$L(y_1, \dots, y_n) = w(y_1, y_2) \oplus L(y_2, \dots, y_n) \ , \ n > 2 \tag{14.94}$$

where

$$a \oplus b := a + \cfrac{1}{1 + \cfrac{1}{b}} \quad , \ a, b \geq 0 \tag{14.95}$$

Here then is the complete non-recursive computation procedure:

$$L(6,9) = w(6,9) = 4 \tag{14.96}$$

$$L(4,6,9) = w(4,6) \oplus L(6,9) = 3 \oplus 4 = \frac{19}{5} \tag{14.97}$$

$$L(2,4,6,9) = w(2,4) \oplus L(6,9) = 5 \oplus \frac{19}{5} = \frac{139}{24} \tag{14.98}$$

$$L(1,2,4,6,9) = w(2,4) \oplus L(6,9) = 2 \oplus \frac{139}{24} = \frac{425}{163} \tag{14.99}$$

Since $a \oplus b$ is strictly increasing with b, when b is positive, it follows that $\oplus$ is Markovian for positive arguments. Thus, the following backward functional equation holds:

$$f(j) = w(j,k) \lfloor \min_{\substack{n \in suc(j)\\ n \neq k}} \{w(j,n) \oplus f(n)\} \ , \ j \in \mathcal{K} \backslash \{k\} \tag{14.100}$$

where $f(j)$ denotes the length of the shortest path from node j to the destination node $E = 9$. Our task is to compute the value of $f(B), B = 1$.

Now, writing the functional equation in greater detail, we obtain

$$f(j) = w(j,k) \lfloor \min_{\substack{n \in suc(j)\\ n \neq k}} \left\{ w(j,n) + \frac{1}{1 + \dfrac{1}{f(n)}} \right\} \ , \ j \in \mathcal{K} \backslash \{k\} \tag{14.101}$$

Remark: Note that $\oplus$ violates the *Big M* convention, namely $a \oplus M$ is not always equal to M regardless of the size of M. In fact, for $M = \infty$ we obtain $a \oplus M = a$. So, we appeal to the $\star$ convention and we define

$$a \oplus \star := \star \ , \ \forall a \tag{14.102}$$

This means that we can let $f(j) = \star$ for $j \in \mathcal{K} \backslash \{k\}$ such that $suc(j) = \varnothing$, and more generally, let $f(j) = \star$ for any j such that there is no path from node j to node E.

In the specific example under consideration this is not an issue because $suc(j) \neq \varnothing$ for all $j \neq k$. But such a convention — or a similar one — is required in cases where there are nodes with no successors.

Note that the binary min operation, $\lfloor$, takes care of situations such as those encountered when node j is an immediate predecessor of the destination $E = k$ and it relies on setting $w(j,k) = \infty$ if node j is not an immediate predecessor of node k. □

But, how would we obtain a *forward* decomposition scheme and a *forward* functional equation for this problem?

Since the operator $\oplus$ deployed in (14.94) is neither associative nor commutative, it seems that the only way to obtain a forward decomposition scheme

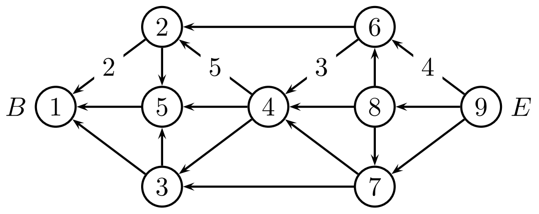


Figure 14.3: Shortest path problem from E to B

for this problem is via a reversal of the arcs' direction and the modification of the objective function L given in (14.83).

So, consider the modified network shown in Figure 14.3 and the modified objective function

$$\vec{L}(y_1,\dots,y_n) = \begin{cases} w(y_1,y_2) & , \quad n=2 \\ w(y_{n-1},y_n) + \dfrac{1}{1+\dfrac{1}{\vec{L}(y_1,y_2,\dots,y_{n-1})}} & , \quad n>2 \end{cases} \tag{14.103}$$

Convince yourself that the shortest path problem represented by this arrangement is equivalent to the original problem we discussed above. In particular, convince yourself that the shortest path from node E to node B in the framework of this problem is equivalent to the shortest path from node B to node E in the original version of the problem.

In the context of this formulation, the forward dynamic programming functional equation — the counterpart of (14.101) — is as follows:

$$\vec{f}(j) = \min_{n\in pre(j)} \left\{ w(n,j) + \frac{1}{1+\dfrac{1}{\vec{f}(n)}} \right\} \ , \ j \in \mathcal{K}, pre(j) \neq \varnothing \tag{14.104}$$

with $\vec{f}(E) = 0$. Our goal is to compute the value of $\vec{f}(B)$.

Needless to say, given this forward formulation, the process of deriving a backward formulation is very similar and it will yield the formulation we obtained at the outset. Also note that $f(j) = \vec{f}(j), j = 1,2,\dots,8$. $\square$

That said, the following question arises: is it always the case that the existence of a formulation implies the existence of a "counterpart"? If so, is there a formula for such a conversion? Does such a conversion always boil down to a simple "reversal in the direction" of the process?

The short answer is that in many cases the conversion is indeed straightforward. In others the conversion can require considerable ingenuity, hence a considerable effort. And in some cases there would be no counterparts at all. To be able to discuss this matter methodically, we need to hark back to the *separability* property. As you will recall, this property enables the formulation of modified problems and consequently the functional equation itself. So let us remind ourself how it all began.

14.3 Initial Problem

In §*4.2* we began with the following definition of a family of initial problems:

Problem $P(\sigma), \sigma \in S_1$:

$$f(s) := \underset{x_1,\dots,x_N}{\text{opt}} \; g(s_1, x_1, x_2, \dots, x_N) \tag{14.105}$$

$$x_n \in D(n, s_n) \ , \ 1 \leq n \leq N \tag{14.106}$$

$$s_{n+1} = T(n, s_n, x_n) \ , \ 1 \leq n \leq N \tag{14.107}$$

$$s_1 = s \tag{14.108}$$

We referred to *Problem* $P(\sigma)$ as THE INITIAL PROBLEM AT σ.

It is extremely important to realize that this formulation is *orientation free:* it is neither "backward" nor "forward". Because, as we have seen, the direction implied by these terms refers to the direction of the decomposition of the *objective function g*. And the direction ascribed to the decomposition is relative to the *transition function* which describes the dynamics of the decision-making process.

But, the "forward" direction implied by the rule $s_{n+1} = T(n, s_n, x_n)$ is only in the sense that this rule indicates that the progression is from one stage (n) to the next (n+1). For this reason we refer to the state observed at stage $n = 1$ as the *initial state* and to the state observed at stage $N + 1$ as the *final state*. As the notion of "backward" and "forward" decomposition is based on the notion of *separable objective functions,* my next task is to examine this foundational concept formally. For convenience, assume that S_1 is a singleton, so let $S_1 = \{\sigma\}$ and $s_1 = \sigma$.

14.4 Separable Objective Functions Revisited

The basic idea is simple, indeed intuitive. The *total* cost or benefit generated by a feasible sequence of decisions $(x_1, \dots, x_N)$ as prescribed by function g, namely $g(s_1, x_1, x_2, \dots, x_N)$, is usually an *accumulation* of values generated at the *stages* of the process.

Recall that the objective function g of a multistage decision model (N, S, D, T, S_1, g) is said to be SEPARABLE if there exists a function ρ and a sequence of functions $(g_n : n \in \mathbb{N}')$ such that the following condition holds: let s_1 be any initial state and let $(x_1, \dots, x_N)$ be any feasible solution to the initial problem at s_1. Then,

$$g(s_1, x_1, x_2, \dots, x_N) = g_1(s_1, x_1, x_2, \dots, x_N) \tag{14.109}$$

$$g_n(s_n, x_n, x_{n+1}, \dots, x_N) = \rho(s_n, x_n, g_{n+1}(s_{n+1}, x_{n+1}, \dots, x_N)) \tag{14.110}$$

for all $n \in \mathbb{N}$, where $s_{n+1} = T(n, s_n, x_n)$.

Observe that by construction $g_{N+1} = g_{N+1}(s_{N+1})$ and recall that by definition $\mathbb{N} := \{1, 2, \dots, N\}$ and $\mathbb{N}' := \{1, 2, \dots, N+1\}$.

Given that our concern here is backward and forward decomposition, it is instructive to refer to such functions as *backward-separable* and to their decomposition schemes $(\rho, (g_n : n \in \mathbb{N}))$ as *backward decomposition schemes.*

Obviously, the forward counterpart will be defined similarly, except that the decomposition will "progress" forward rather than a backward.

Definition 14.4.1 *The objective function g of a dynamic programming model (N, S, D, T, S_1, g) is said to be* BACKWARD-SEPARABLE *if there exits a function ρ and a sequence of functions $(g_n : n \in \mathbb{N}')$ such that the following condition holds: let $(x_1, \dots, x_N)$ be any feasible solution to the initial problem at σ. Then, (14.110) - (14.110) hold for all $n \in \mathbb{N}$. We refer to $(\rho, (g_n : n \in \mathbb{N}'))$ as a* BACKWARD DECOMPOSITION SCHEME.

Similarly, g is said to be FORWARD-SEPARABLE *if there exits a function $\vec{\rho}$ and a sequence of functions $(\vec{g}_n : n \in \mathbb{N}')$ such that $\vec{g}_n$ is a real-valued function on $S_1 \times \mathbb{D}^{n-1} \times S_n$ and the following conditions hold:*

$$g(s_1, x_1, x_2, \dots, x_N) = \vec{g}_{N+1}(s_1, x_1, x_2, \dots, x_N) \tag{14.111}$$

$$\vec{g}_n(s_1, x_1, \dots, x_{n-1}) = \vec{\rho}(n-1, s_{n-1}, x_{n-1}, \vec{g}_{n-1}(s_1, x_1, \dots, x_{n-2})) \tag{14.112}$$

for $1 < n \leq N$, where as usual $s_n = T(n-1, s_{n-1}, x_{n-1})$.

We refer to $(\vec{\rho}, (\vec{g}_n))$ as a FORWARD DECOMPOSITION SCHEME. *Note that this setup implies that g_1 is a real-valued function on S_1.* □

As I noted on several occasions, insofar as dynamic programming is concerned, the most commonly encountered objective function is the *additive* function of the form

$$g(s_1, x_1, x_2, \dots, x_N) = \sum_{n=1}^{N} w(n, s_n, x_n) \tag{14.113}$$

This function is clearly both backward-separable and forward-separable. The same is true for any function of the form

$$g(s_1, x_1, x_2, \dots, x_N) = w(1, s_1, x_n) \oplus \cdots \oplus w(N, s_N, x_N) \tag{14.114}$$

where $\oplus$ is a binary *associative operator*. Note that in such cases the composition functions ρ and $\overline{\rho}$ can be defined as follows:

$$\rho(n, s_n, x_n, r) = w(n, s_n, x_n) \oplus r \ , \ r \in \mathbb{R} \tag{14.115}$$

$$\vec{\rho}(n-1, s_{n-1}, x_{n-1}, r) = r \oplus w(n-1, s_{n-1}, x_{n-1}) \ , \ r \in \mathbb{R} \tag{14.116}$$

Also note that (14.114) implies that

$$g_{N+1}(s) = R_{\oplus} \tag{14.117}$$
$$\vec{g}_1(s) = L_{\oplus} \tag{14.118}$$

where $R_{\oplus}$ denotes the right identity element of $\oplus$ and $L_{\oplus}$ denotes its left identity element.

Recall that the cases $\oplus = \times$, $\oplus = \lceil$, $\oplus = \lfloor$ were discussed in §10.3 and that their identity elements are as follows:

$$R_{+} = L_{+} = 0 \tag{14.119}$$
$$R_{\times} = L_{\times} = 1 \tag{14.120}$$
$$R_{\lceil} = L_{\lceil} = -\infty \tag{14.121}$$
$$R_{\lfloor} = L_{\lfloor} = \infty \tag{14.122}$$

To satisfy the Markovian condition, the multiplicative operator, namely $\times$, is restricted to operate on non-negative arguments.

If $\oplus$ does not possess identity elements it may be necessary to make do without g_{N+1} in the backward scheme and without g_1 in the forward scheme.

Definition 14.4.2 *If the objective function g admits of the representation stipulated in (14.114) and $\oplus$ is associative, then the scheme $(\oplus, w)$ is called an* ASSOCIATIVE BINARY DECOMPOSITION SCHEME. □

With this as background, I can now proceed to outline the derivation of the dynamic programming functional equation induced by a forward decomposition scheme.

14.5 Modified Problems Revisited

In the context of the present discussion:

Problem $P(n, s), n \in \mathbb{N}, s \in S_n$:

$$f_n(s) := \underset{x_n,\dots,x_N}{\text{opt}} \; g_n(s_n, x_n, x_{n+1}, \dots, x_N) \tag{14.123}$$
$$x_j \in D(j, s_j) \; , \; n \leq j \leq N \tag{14.124}$$
$$s_{j+1} = T(j, s_j, x_j) \; , \; n \leq j \leq N \tag{14.125}$$
$$s_n = s \tag{14.126}$$

is referred to as the BACKWARD MODIFIED PROBLEM AT (n, s).

The rationale for this is that *Problem* $P(1, s_1)$ is by construction equivalent to *Problem* $P(s_1)$. It is convenient therefore to let $f_{N+1}(s) := g_{N+1}(s), s \in S_{N+1}$.

Next, the same is done with regard to the modified problems induced by forward decomposition schemes. That is, the modified problems induced by the forward decomposition scheme are defined so that at least one of the modified problems will be equivalent to the initial problem of interest. So in line with our definition of a forward decomposition scheme consider the following class of problems:

Definition 14.5.1 *Problem* $\vec{P}(n,s), n \in \{2,3,\dots,N\}, s \in S_n$:

$$\vec{f_n}(s) := \underset{x_1,\dots,x_{n-1}}{\text{opt}} \ \vec{g}_n(s_1,x_1,x_2,\dots,x_{n-1}) \tag{14.127}$$

$$x_j \in D(j,s_j)\ ,\ 1 \le j < n \tag{14.128}$$

$$s_{j+1} = T(j,s_j,x_j)\ ,\ 1 \le j < n \tag{14.129}$$

$$s_1 = \sigma \tag{14.130}$$

$$s_n = s \tag{14.131}$$

We refer to Problem $\vec{P}(n,s)$ as the FORWARD MODIFIED PROBLEM AT (n,s) GIVEN THE INITIAL STATE σ*. It is convenient to let $f_1(\sigma) := \vec{g}_1(\sigma)$, recalling that in this discussion the initial state of the process, σ, is treated as a given parameter, so formally $S_1 = \{\sigma\}$.* □

In words, the forward modified problem at (n,s) can be read as posing the following question: What is the optimal way to reach state s at stage n from the initial state $s_1 = \sigma$ at stage 1 in accordance with the modified objective function $\vec{g}_n$?

Let $\vec{X}(n,s)$ and $\vec{X}^*(n,s)$ denote the set of feasible/optimal solutions to *Problem* $\vec{P}(n,s)$ respectively.

The idea is then to view the initial problem at $s_1 = \sigma$ as a member of the new family of forward modified problems.

Specifically, in view of (14.111), the idea is to solve *Problem* $\vec{P}(N+1, s_{N+1})$ for some $s_{N+1} \in S_{N+1}$. But the question is: what final state s_{N+1} should be used for this purpose?

The following result addresses this important methodological question whose implications are not only theoretical but also practical.

Theorem 14.5.1 *Assume that N is finite, let $(x_1,\dots,x_N)$ be any optimal solution to Problem $P(\sigma)$ and let $s^* \in S_{N+1}$ be the final state generated by applying $(x_1,\dots,x_N)$ to $s_1 = \sigma$. Then any optimal solution to Problem $\vec{P}(N+1,s^*)$ is also optimal with respect to Problem $P(\sigma)$.*

PROOF. By construction $\vec{g}_{N+1} = g = g_1$, and any feasible solution to *Problem* $\vec{P}(N+1,s^*)$ is also feasible for *Problem* $P(\sigma)$. Hence, if $(x_1,\dots,x_N)$ is an optimal solution to *Problem* $P(\sigma)$ it must also be optimal with respect to *Problem* $\vec{P}(N+1,s^*)$. Thus, any optimal solution to *Problem* $\vec{P}(N+1,s^*)$ must also be optimal with respect to *Problem* $P(\sigma)$. □

Admittedly, this theorem does not provide practical instruction on how to pin down this final state. Still, it is important because of the foundational significance of the assertion that it makes. It gives the assurance that the set of modified problems pertaining to the forward decomposition scheme provides a sound basis for the derivation of the dynamic programming functional equation.

The difficulty arising from the fact that we may not know a priori which final state should be used for the generation of an optimal solution for the initial problem of interest is usually just a minor technical irritant. The fact is that, for a large class of problems, this difficulty does not arise at all because:

Corollary 14.5.1 *If S_{N+1} is a singleton, say $S_{N+1} = \{\vec{s}\}$, then Problem $\vec{P}(N+1, \vec{s})$ is equivalent to Problem $P(\sigma)$.* □

If N is finite, it is possible to forcibly incorporate in the process a unique final state through the use of DUMMY VARIABLES. So, however humble this result may appear at first sight, its implications are in fact far reaching.

14.5.1 Example

Consider the case where the state variable is determined by the following knapsack-type constraints:

$$\sum_{n=1}^{N} b_n x_n \leq v \tag{14.132}$$

$$x_n \in \{0, 1, 2, \dots\} \tag{14.133}$$

where all the coefficients are positive integers.

Suppose that the state variable is defined in the usual fashion, namely let

$$s_n := v - \sum_{j=1}^{n-1} b_j x_j \ , \ n = 2, 3, \dots, N+1 \tag{14.134}$$

with $s_1 = v$.

Then it follows that as a rule S_{N+1} is not a singleton. In view of this, a dummy (slack) variable x_{N+1} would be introduced and the global constraint modified as follows:

$$\sum_{n=1}^{N+1} b_n x_n = v \tag{14.135}$$

with $b_{N+1} = 1$.

In this case the set of final states will be a singleton, namely $S_{N+2} = \{0\}$. One need hardly point out that it is imperative to make sure that the incorporation of this dummy variable in the model does not change the nature of the optimal solutions to the original problem. For instance, if the objective function is linear, all that needs to be done is to set the coefficient of this variable in the objective function to zero. □

Let us now proceed to the second stage in the derivation of the functional equation yielded by the forward decomposition scheme.

14.6 Backward Conditional Problems Revisited

Recall that the conditional problems formulated in §*4.2* were defined by fixing the values of the decision variables associated with the modified problems. In other words, in the case of the backward decomposition scheme this class of problems was defined as follows:

Problem $P(n,s,d), n \in \mathbb{N}, s \in S_n, d \in D(n,s)$:

$$f_n(s,d) := \operatorname*{opt}_{x_{n+1},\dots,x_N} g_n(s_n, x_n, x_{n+1}, \dots, x_N) \tag{14.136}$$

$$x_j \in D(j, s_j)\ ,\ n+1 \leq j \leq N \tag{14.137}$$

$$s_{j+1} = T(j, s_j, x_j)\ ,\ n \leq j \leq N \tag{14.138}$$

$$s_n = s \tag{14.139}$$

$$x_n = d \tag{14.140}$$

We referred to *Problem* $P(n,s,d)$ as the CONDITIONAL PROBLEM AT (n,s,d) and we let $X^*(n,s,d)$ denote the set of optimal solutions to *Problem* $P(n,s,d)$.

In the case of the forward decomposition scheme, the conditional problems take a slightly more complicated form because for a given $s \in S_n$ there can be many pairs (s',x) such that $s' \in S_{n-1}$, $x \in D(n-1,s')$ and $s = T(n-1,s',x)$. In other words, $T(n,s,\cdot)$ might not have a proper inverse. Therefore, for the purposes of our analysis it is conveneient to use the following pseudo-inverse of T:

$$T^{-1}(n,s) := \{(s',x) : s' \in S_{n-1}, x \in D(n-1,s')\ , s = T(n-1,s',x)\} \tag{14.141}$$

for $n = 2, \dots, N+1$ and $s \in S_n$.

By definition then, $T^{-1}(n,s)$ is the set of all (state,decision) pairs at stage $n-1$ that generate state s at stage n. With this in mind consider the following:

Definition 14.6.1 *Problem* $\vec{P}(n,s,s',d), 1 < n \leq N+1, s \in S_n, (s',d) \in$

$T^{-1}(n,s)$:

$$\vec{f}_n(s,s',d) := \underset{x_1,\dots,x_{n-1}}{\text{opt}} \ \vec{g}_n(s_1,x_1,x_2,\dots,x_{n-1}) \tag{14.142}$$

$$x_j \in D(j,s_j) \ ,1 \le j \le n-1 \tag{14.143}$$

$$s_{j+1} = T(j,s_j,x_j) \ , \ 1 \le j \le n-1 \tag{14.144}$$

$$s_{n-1} = s' \tag{14.145}$$

$$x_{n-1} = d \tag{14.146}$$

$$s_n = s \tag{14.147}$$

$$s_1 = \sigma \tag{14.148}$$

We refer to Problem $\vec{P}(n,s,s',d)$ as the FORWARD CONDITIONAL PROBLEM AT (n,s,s',d). *Let $\vec{X}^*(s,s',d)$ denote the set of optimal solutions to Problem $\vec{P}(n,s,s',d)$. For convenience, let $\vec{f}(1,\sigma) = \vec{g}_1(\sigma)$.*

By inspection,

Theorem 14.6.1

$$\vec{f}_n(s) = \underset{(s',x)\in T^{-1}(n,s)}{\text{opt}} \vec{f}_n(s,s',x) \ , \ 1 < n \le N+1, s \in S_n \tag{14.149}$$

In words, the best way to reach state s at stage n is to first reach some state s' in stage $n-1$ such that $s = T(n-1,s',x)$ for some $d \in D(n-1,s')$. Such a (s',x) pair is determined by (14.149).

14.7 Markovian Condition Revisited

Recall that in §*4.2* the Markovian condition

$$X^*(n,s,x) = X^*(n+1,T(n,s,x)) \tag{14.150}$$

for all $1 \le n < N, s \in S_n, x \in D(n,s)$, was introduced to establish the validity of backward dynamic programming functional equations. In the case of the forward decomposition scheme this condition takes the following form:

Definition 14.7.1 *The fowrward decomposition scheme $(\vec{\rho},(\vec{g}_n : n \in \mathbb{N}'))$ is said to be* MARKOVIAN *iff*

$$\vec{X}^*(n,s,s',x) = \vec{X}^*(n-1,s') \tag{14.151}$$

for all (s,s',x) such that $s \in S_n$, and $(s',x) \in T^{-1}(n,s)$.

The scheme is said to be WEAK-MARKOVIAN *iff*

$$\vec{X}^*(n,s,s',x) \cap \vec{X}^*(n-1,s') \neq \varnothing \tag{14.152}$$

for all (s,s',x) such that $s \in S_n$ and $(s',x) \in T^{-1}(n,s)$. □

We are now ready to consider the functional equation induced by the forward decomposition scheme.

14.8 Forward Functional Equation

The generic functional equation derived by means of the backward decomposition scheme is as follows:

$$f_n(s) = \operatorname*{opt}_{x \in D(n,s)} \rho(n, s, x, f_{n+1}(T(n, s, x))) \ , \ 1 \leq n \leq N, s \in S_n \quad (14.153)$$

Its validity is assured by the Weak Markovian condition. For the forward decomposition scheme we have:

Theorem 14.8.1 *If the forward decompostion scheme is Weak-Markovian then*

$$\vec{f}_n(s) = \operatorname*{opt}_{(s',x) \in T^{-1}(n,s)} \vec{\rho}(n-1, s', x, \vec{f}_{n-1}(s')) \quad (14.154)$$

for all $1 < n \leq N+1, s \in S_n$*, with* $\vec{f}(1, s) := \vec{g}_1(s), s \in S_1$.

If the forward Markovian decomposition scheme admits of a binary representation, then this functional equation will take the more familiar form

$$\vec{f}_n(s) = \operatorname*{opt}_{(s',x) \in T^{-1}(n+1,s)} \left\{ \vec{f}_{n-1}(s') \oplus w(n, s', x) \right\} \quad (14.155)$$

for $1 < n \leq N+1, s \in S_n$. And if the model is *stageless,* then

$$\vec{f}(s) = \operatorname*{opt}_{(s',x) \in T^{-1}(s)} \left\{ \vec{f}(s') \oplus w(s', x) \right\} \quad (14.156)$$

for all $s \in S$ such $T^{-1}(s) \neq \varnothing$, observing that in this case

$$T^{-1}(s) := \{(s', x) : s' \in S, s = T(s', x)\}. \quad (14.157)$$

The ability to count on the forward functional equation clearly allows for a greater flexibility in the formulation of problems. The question is then what are the practical implications of this fact?

14.9 Impact on the State Space

The question that immediately springs to mind is whether the availability of forward formulations can be instrumental in reducing the state space of a dynamic programming model. For, as we know by now, securing a small state space is a crucial consideration in the formulation of dynamic programing models.

Consider then the following argument that is often used to suggest that "forward dynamic programming" can sometimes reduce significantly the size of the state space:

In practice, generating the sets of feasible states $\{S_n\}$ is often inexpedient. So, the functional equation is solved instead for sets $\{\widehat{S}_n\}$ such that

$$S_n \subseteq \widehat{S}_n \ , \ \forall n \in \mathbb{N} \tag{14.158}$$

But, the trouble is that in some cases this can be extremely wasteful because $\widehat{S}_n$ can be much larger than S_n.

14.9.1 Example

Consider again the case of the knapsack problem and assume that the state variable in the model is defined by (14.134). Clearly,

$$S_{n+1} = \{s - mb_n : s \in S_n, m = 0, 1, \dots, \lfloor s/b_n \rfloor\} \ , \ n = 2, 3, \dots, N \tag{14.159}$$

with $S_1 = \{v\}$.

If a backward functional equation is used to solve the problem and implemented verbatim, then the solution procedure will commence at stage $n = N$ and will require the set S_N. The problem is that to generate this set all the sets $S_j, j = 1, 2, \dots, N-1$ must be generated first, which is almost as demanding (computationally) as solving the functional equation under consideration.

So in practice we often do the following. Instead of using the exact sets, the functional equation is solved for the "outer approximation" of these sets given by

$$\widehat{S}_n = \{0, 1, 2, \dots, v\} \ , \ n = 2, \dots, N \tag{14.160}$$

with with $\widehat{S}_1 = S_1 = \{v\}$.

But it is immediately clear that this lax approach can be extremely wasteful. For instance, consider the simple case where $N = 4$ and the knapsack constraint is as follows:

$$8765x_1 + 7654x_2 + 9876x_3 + 4897x_4 \leq 100,000 \tag{14.161}$$

In this case the approximating sets will be

$$\widehat{S}_n = \{0, 1, 2, \dots, 100000\} \ , \ n = 2, \dots, N \tag{14.162}$$

with $\widehat{S}_1 = S_1 = \{100,000\}$.

This means that the functional equation will be solved

$$|\widehat{S}_1| + |\widehat{S}_2| + |\widehat{S}_3| + |\widehat{S}_4| = 1 + 3 \times 100,001 = 300,004 \tag{14.163}$$

times, the odd 1 representing the singleton $S_1 = \{v\}$. It is easy to ascertain that

$$|S_1| + |S_2| + |S_3| + |S_4| \leq 34,000 \tag{14.164}$$

Clearly, no more needs to be said about the drawbacks of this lax approach in this case.

On the other hand — so the argument goes — if a forward functional equation is used, then given that in this case the exact sets can be generated in an impromptu fashion as the functional equation is being solved, there is no need to use the outer approximations at all.

Remark: Considerations of this kind can also be taken into account with respect to "backward" functional equations, albeit not always in such a straightforward manner. □

It should also be noted that a similar argument can be invoked to explain the use of the backward formulation in cases where the final state is unique, and the initial state is not fixed a priori.

14.10 Anomaly

A comparison between the forward-formulation and the backward formulation gives rise to an intriguing question.

Given the preceding discussions, it seems rather clear that a dynamic programming functional equation based on a forward formulation should at times be more complicated than the functional equation based on a backward formulation, and vice versa.

What is intriguing, though, is that this occurs in cases where the respective formulations are of a problem that to all intents and purposes seems to be utterly "symmetric". That is, the problem has a unique initial state, a unique final state and an associative binary decomposition scheme.

One would have thought that in such cases the forward and backward formulations would be very similar. In particular, that they would have the same state variable.

And what, on the face of it, is even more intriguing is that in certain cases a forward formulation is readily available, yet no "equivalent" backward formulation seems to be available. The following discussion takes up these questions.

14.10.1 Example

Consider the small network depicted in Figure 14.4 and the following problem associated with it:

· The task is to traverse from node 1 to node 7.
· As an arc is traversed, revenue (displayed on the arc) is being generated.
· The objective is to maximize the total revenue (sum of the arc revenues).

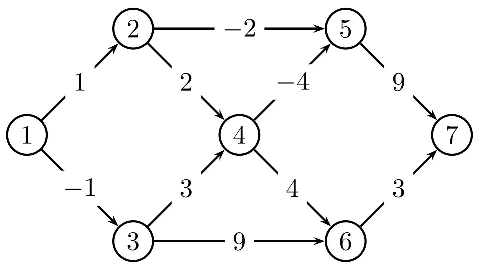


Figure 14.4: Longest path problem

· What is the optimal path?

For example, the revenue generated by the path $(1,2,5,7)$ is equal to $(1+(-2)+9) = \$8$. So, the objective is to identify the path from node 1 to node 7 that generates the maximum revenue.

I consider two versions of this generic problem. In the first version no *cash-flow* restrictions are imposed on the problem. This means that at any stage of the process the accumulated revenue is allowed to be *negative.* In the second version, the problem is cash-flow sensitive meaning that the accumulated revenue is not allowed to be negative at any stage of the process.

Let $r(i,j)$ denote the revenue generated by arc (i,j), let $pre(j)$ denote the set of immediate predecessors of node j and let $suc(j)$ denote the set of immediate successors of node j. Assume that there are n nodes, $\{1,2,\dots,n\}$, and that the process starts at node 1 and terminates at node n. For simplicity assume that the network is acyclic. Let $\mathcal{K} := \{1,2,\dots,k\}$.

14.10.2 Version #1: Cash-Flow-Free

This version of the problem is the classic *longest path problem.* The forward and backward dynamic programming functional equations are thus as follows:

Backward	Forward
$f(j) = \max\limits_{n\in suc(j)} \left\{r(j,n) + f(n)\right\}$	$\vec{f}(j) = \max\limits_{n\in pre(j)} \left\{\vec{f}(n) + r(n,j)\right\}$
$j \in \mathcal{K}\backslash\{k\}$	$j \in \mathcal{K}\backslash\{1\}$
$f(k) = 0$	$\vec{f}(1) = 0$

Unsurprisingly, the two formulations have the same state space S, namely in both cases the states are nodes of the network and $S = \mathcal{K}$.

Solving the functional equation for the problem depicted in Figure 14.4 we obtain the following results:

j	1	2	3	4	5	6	7
$f(j)$	11	9	12	7	9	3	0
$\vec{f}(j)$	0	1	−1	3	−1	8	11

Thus, the maximum revenue is equal to $f(1) = \vec{f}(7) = 11$ and the optimal path is $x = (1, 3, 6, 7)$.

14.10.3 Version #2: Cash-Flow Constraint

This version of the problem prohibits negative intermediate revenues. This means that it is imperative at node j to have the value of the revenue accrued so far in order to determine the arcs to be traverse next.

This suggests the use of a state variable of the form $s = (j, v)$ where j denotes the current node and v denotes the revenue accumulated so far. The cash-flow restriction is expressed by the requirement that v is not allowed to be negative at any point. Thus, if the process is at state $s = (j, v)$ then traversing from node j to node m along arc (j, m) is allowed only if $v + r(j, m) \geq 0$ or equivalently if $v \geq -r(j, m)$.

If it is assumed that the process begins at zero revenue, then the initial state of the process would be $s_1 = (1, 0)$. Let $V(j)$ denote the set of feasible values of v pertaining to node j, observing that $V(1) = \{0\}$ and

$$V(j) = \big\{v + r(m, j) : v \in V(m), m \in pre(j), v \geq -r(m, j)\big\} \qquad (14.165)$$

for $j \in \mathcal{K} \backslash \{1\}$.

Since it is assumed that the network is acyclic, the sets $V(1), \ldots, V(n)$ are finite and can be easily computed. Observe that by construction the transition function is as follows:

$$T((j, v), m) := \big(m, v + r(j, m)\big) \ , \ j \in \mathcal{K}, v \in V(j), m \in suc(j) \qquad (14.166)$$

That said, the backward and forward functional equations for the version under consideration are as follows:

Backward functional equation:

Let

$f(j, v) :=$ maximum total revenue generated at the end of the process given that the process commences at node j and \$$v$ is available in cash.

Then the backward functional equation will be as follows:

$$f(j, v) = \max_{\substack{m \in suc(j) \\ v + r(j,m) \geq 0}} \{r(j, m) + f(m, v + r(j, m))\} \qquad (14.167)$$

for $j \in \mathcal{K} \backslash \{k\}, v \in V(j)$ with $f(n, v) = 0, \forall v \in V(k)$.

Forward functional equation:

Let

$\vec{f}(j, v) :=$ maximum total revenue that can be accumulated while traveling from node 1 to node j given that \$$v$ has been accumulated en route from node 1 to node j.

This leads to the following forward functional equation

$$\vec{f}(j,v) = \max_{\substack{m\in pre(j)\\ v+r(m,j)\geq 0}} \left\{\vec{f}(m, v - r(m,j)) + r(m,j)\right\} \tag{14.168}$$

for $j \in \mathcal{K}\backslash\{1\}, v \in V(j)$, with $\vec{f}(1,0) = 0$.

So far so good except that ... although this functional equation is perfectly valid, the definition of $\vec{f}(j,v)$ is not all that sensible. Because: the maximum revenue accumulated along the route from node 1 to node j is made conditional on the (not necessarily optimal) value v? In other words, what is the point of seeking the non-optimal values of v at node j? The inference is then that it is far more sensible to base the forward functional equation on the following far simpler, indeed straightforward family of modified problems:

$\vec{f}(j) :=$ maximum revenue that can be accumulated while traveling from node 1 to node j.

Then clearly,

$$\vec{f}(j) = \max_{\substack{m\in pre(j)\\ \vec{f}(m)+r(m,j)\geq 0}} \left\{\vec{f}(m) + r(m,j)\right\} \ , \ j \in \mathcal{K}\backslash\{1\} \tag{14.169}$$

with $\vec{f}(1) = 0$.

This functional equation is manifestly more attractive than (14.168) and (14.167) because it does not call for the nuisance parameter v. The state space in this case is simply $S = \mathbb{N} = \{1, \dots, n\}$. Observe that the same state space was used in the derivation of the functional equation for the much simpler Cash-Flow-Free version of the problem!

Applying this functional equation to the problem depicted in Figure 14.4 yields the following results. The two $-\infty$ values are associated with cases where node j cannot be reached from node 1.

j	1	2	3	4	5	6	7
$\vec{f}(j)$	0	1	$-\infty$	3	$-\infty$	7	10

Thus, the maximum revenue that can be generated along a path from node 1 to node $n = 7$ subject to the cash-flow restrictions is equal to $\vec{f}(7) = 10$. The optimal path is $x = (1, 2, 4, 6, 7)$.

Given the ease with which a valid forward functional equation based on $S = \mathbb{N} = \{1, \dots, n\}$ denoting the state space and $s^o = 1$ denoting the initial state is obtained, one would have thought that deriving a valid backward functional equation based on these constructs would be a straightforward matter as well. And yet, this is not so, in spite of the objective function being additive, hence the binary decomposition scheme, $\oplus = +$, being associative and commutative.

The question is then: how is it that a problem with such an objective function, where the binary composite operator is associative and commutative, allows the formulation of a simple forward composition scheme but not an "equivalent" backward decomposition scheme?

Remark:
Before I can take up this matter, I should note that although writing $\vec{f}(m)$ below the max in (14.169) may seem unconventional, this formulation is perfectly legitimate because this $\vec{f}(m)$ is the same $\vec{f}(m)$ displayed in the curly brackets on the right-hand side of the equation. Furthermore, since the network is acyclic, in the framework of the forward paradigm, the values of $\{\vec{f}(m), m \in pre(j)\}$ are regarded as *given* at node j. □

To answer the above question we need to examine the problem in a more *formal* setting. In particular, we need to examine more carefully whether the problem is indeed as "symmetric" as it appears at first.

Consider then this formal statement of the Cash-Flow constrained version of the problem:

Problem P :

$$z^* := \max_{\substack{x_1,\dots,x_k \\ k}} \sum_{j=1}^{k-1} r(x_j, x_{j+1}) \tag{14.170}$$

$$\sum_{i=1}^{j} r(x_i, x_{i+1}) \geq 0 \ , \ j = 1, 2, \dots, k-1 \tag{14.171}$$

$$x_1 = B \tag{14.172}$$

$$x_k = E \tag{14.173}$$

$$x_{j+1} \in suc(j) \ , \ j = 1, 2, \dots, k-1 \tag{14.174}$$

where

- · B = origin node.
- · E = destination node.
- · $suc(j)$ = set of immediate successors of node j.
- · $r(i, j)$ = revenue generated by $arc(i, j)$.
- · x_j = j-th node on the path.

Observe that the k below the max is a gentle reminder that the number of nodes on a feasible path is not fixed in advance, it is a decision variable.

The point of this observation is to indicate that in this case it should be instructive to spell out the family of $(k-1)$ global constraints embedded in (14.171). Note then that for the case where $k = 4$ the picture is this:

$$\begin{array}{ll} r(x_1,x_2) & \geq 0 \\ r(x_1,x_2)+r(x_2,x_3) & \geq 0 \\ r(x_1,x_2)+r(x_2,x_3)+r(x_3,x_4) & \geq 0 \end{array}$$

This done it becomes abundantly clear that the problem in question is not "symmetric" at all, because the global constraints in this case are not "symmetric". Indeed they expand forward. This being the case, it only stands to reason that when handled by dynamic programming these constraints would be treated accordingly. That is, they would give rise to a "forward" decomposition scheme rather than a "backward" scheme.

I might add that as an extra bonus, the state variables induced by these constraints can be aggregated into a single variable, namely

$$\pi_j := \sum_{i=1}^{j-1} r(x_i, x_{i+1}) \ , \ j = 2, 3, \dots, k \tag{14.175}$$

with $\pi_1 = 0$.

The above system of constraints can now be re-stated as follows:

$$\pi_j \geq 0 \ , \ j = 1, 2, 3, 4 \tag{14.176}$$

Note that a crucial point in this analysis is that the *preference is for large values of* π. That is, if node i can be reached from node 1 in two ways, yielding say $\$\pi'$ and $\$\pi''$ such that $\pi'' > \pi'$, then the route that yields $\$\pi''$ is preferable.

So in the framework of the forward formulation, the modified problem at node i is as follows: what path from node 1 to node i, subject to the cash-flow constraint, is the most beneficial? We do not have to consider less beneficial paths from node 1 to node i. □

The conclusion to be drawn at this point is then that the "anomaly" illustrated by the above example has a simple, indeed a self-evident explanation: there are cases where the "forward" decomposition scheme is more suitable than the "backward" scheme, and there are cases where the "backward" decomposition scheme is more suitable than the "forward" scheme.

14.10.4 Example

Suppose that we modify the Cash-Flow requirement in the preceding example and we apply it to future — rather than to past — partial accumulations at the destination node. That is, assume that a path $(x_1, \dots, x_k)$ from the origin, namely from node 1, to the destination, namely node 7, must satisfy the following Cash-Flow requirement:

$$\sum_{i=n}^{k-1} r(x_i, x_{i+1}) \geq 0 \ , \ n = 1, 2, \dots, k-2 \tag{14.177}$$

For instance, if $m = 4$ then the Cash-Flow requirement is as follows:

$$\begin{aligned} r(x_3, x_4) & \geq 0 \\ r(x_2, x_3) + r(x_3, x_4) & \geq 0 \\ r(x_1, x_2) + r(x_2, x_3) + r(x_3, x_4) & \geq 0 \end{aligned}$$

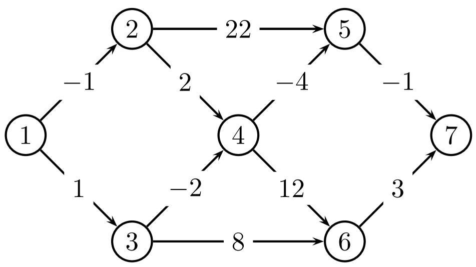


Figure 14.5: Longest path problem

Now, for $j = 1, 2, \ldots, 7$ let

$$f(j) := \text{return generated by an optimal path from node j to node 7}$$

with $f(7) = 0$.

Then the Cash-Flow requirement at node j dictates that the transition from node j to node n is allowed only if $r(j,n) + f(n) \geq 0$, or equivalently $r(j,n) \geq -f(j)$. Thus, the backward functional equation is as follows:

$$f(j) = \max_{\substack{n \in Suc(j) \\ r(j,n) \geq -f(n)}} \{r(j,n) + f(n)\} \ , \ j = 6, 5, 4, 3, 2, 1 \tag{14.178}$$

Our objective is to determine the value of $f(1)$.

Note that writing $f(n)$ below the max operator is not problematic because it occurs elsewhere on the right hand side of the equation anyhow.

For the instance specified by Figure 14.5 this functional equation yields the optimal path $\mathbf{x} = (1, 2, 4, 6, 7)$ whose length is equal to 16. The details are as follows:

j	1	2	3	4	5	6	7
$f(j)$	16	17	13	15	$-\infty$	3	0

Note that the optimal solution to the cash-flow-free problem is the path $\mathbf{y} = (1, 2, 5, 7)$ whose length is equal to 20. This path does not satisfy the Cash-Flow requirement because $r(y_3, y_4) = r(5, 7) = -1 < 0$.

As an exercise in dynamic programming modeling, the reader is encouraged to attempt to formulate a “forward” functional equation for this problem subject to the Cash-Flow requirement on future partial accumulations at the destination (without reversing the directions of the arcs). **Hint**: this is a somewhat tricky task! □

The conclusion to be drawn from this example is that there are cases where decomposing the objective function “backwards” seems most compelling so naturally one would be inclined to decompose it in this fashion. And, as we show in the next section, there are pathologic cases where “forward” decomposition should not even be contemplated.

14.11 Pathologic Cases

The most obvious pathologic case impeding the formulation of a forward decomposition arises in situations where the transition function T is *multi-valued*, namely where $T(n, s, x)$ is a **set** of states rather than a single state. As indicated in *Chapter 9*, such models are called *non-serial* models.

Since this topic is not discussed in this book, I shall not go into the specifics of what types of non-serial models are unsuitable for forward decomposition. I shall only touch on this issue primarily because the most well-known *stochastic* or *probabilistic* applications of dynamic programming are non-serial dynamic programming models.

In such models a decision at stage n of the process generates a number of states, each observed with a given *probability* at the next stage, $n+1$. For the purposes of this discussion it is sufficient to examine the simple case where the number of stages, N, is finite, the state space S is finite, the decision sets $\{D(n, s)\}$ are finite and

$$T(n, s, x) = \{t(n, s, x, q) : q \in Q_n\} \subseteq S_{n+1} \tag{14.179}$$

where Q_n is a given discrete set that depends on stage n, and $t(n, s, x, q)$ denotes the state (in stage $n + 1$) resulting from the application of decision x at stage n given that state s and input q are observed at stage n.

We regard q as the realization of a random variable $\tilde{q}_n$ whose probability mass function is known, call it p_n. So, $p_n(q)$ denotes the probability that $\tilde{q}_n = q, q \in Q$. In this framework,

$$s_{n+1} = t(n, s, x, q) \tag{14.180}$$

denotes the next state of the process, at stage $n + 1$, given that at stage n we observe state s, select decision $x \in D(n, s)$ and the realization of q_n is q. Observe that this means that the state variables $(s_n : n > 1)$ are also realizations of random variables $(\tilde{s}_n : n > 1)$ respectively, whose probability mass functions are determined by t, the decision-making policy used, and the probability mass functions $(p_n : n \geq 1)$. In particular,

$$Prob[\tilde{s}_{n+1} = s' | \tilde{s}_n = s, x_n = x] = \sum_{\substack{q \in Q_n \\ s' = t(n,s,x,q)}} p_n(q) \tag{14.181}$$

where $Prob[\tilde{s}_{n+1} = s' | \tilde{s}_n = s, x_n = x]$ denotes the conditional probability that $\tilde{s}_{n+1} = s'$ given that $\tilde{s}_n = s$ and $x_n = x$.

Now, let

$f_n(s) :=$ optimal expected value of the total return accumulated from stage n onward, given that state s is observed at stage n.

It is immediately obvious that if the objective function is additive, that is if

$$g(s_1, x_1, q_1, \dots, s_N, x_N, q_N) = \sum_{n=1}^{N} w(n, s_n, x_n, q_n) \tag{14.182}$$

then the backward dynamic programming functional equation is as follows:

Theorem 14.11.1

$$f_n(s) = \underset{x \in D(n,s)}{\text{opt}} \left\{ \sum_{q \in Q} p_n(q) \left[w(n, s, x, q) + f_{n+1}(t(n, s, x, q)) \right] \right\} \tag{14.183}$$

for $1 \le n \le N, s \in S_n$, *with* $f_{N+1}(s) := 0, \forall s \in S_N$.

But, what about a "forward" counterpart for this functional equation?

Of course, by reversing the direction of the process we can "automatically" obtain a forward functional equation that is essentially identical to (14.183). But is it possible to derive a "genuine" forward counter-part that does not arise from a simple reversal in the direction of the process.

The answer is yes but ... qualified by caveats and reservations.

The difficulty here is that in order to compute, at some given stage say n, the expected value of the reward accumulated from the initial state to a given state, we need to know the probability distribution function of the state variable (now a random variable) $\tilde{s}_{n-1}$. But this distribution depends not only on the distributions $(p_1, \dots, p_{n-1})$ but also on the decision-making policy that determines the decisions $x_1, \dots, x_{n-2}$. And this is unknown!

The way out of this impasse is to reformulate the multistage decision model, by rephrasing the state variable as a probability mass function on the state space associated with the current state variable s. That is, let the new state space, call it Π be the set of all the probability mass functions on S and similarly let Π_n be the set of all probability mass functions on S_n, $n = 1, 2, \dots, N+1$.

Based on this rephrased state variable, we can now formulate suitable decisions, transitions and returns to complete the modeling task and derive a forward dynamic programming functional equation. The end product is a forward dynamic programming functional equation along these lines:

$$F_{n+1}(\pi) = \underset{(\pi', \delta) \in \Omega(n+1, \pi)}{\text{opt}} \{ W(n, \pi', \delta) + F_n(\pi') \} \tag{14.184}$$

for $1 < n \le N, \pi \in \Pi_{n+1}$ with $F_1(\pi) = 0, \pi \in \Pi_1$.

This equation looks good until you realize what the objects Π_n, W and Ω actually designate. As noted above, Π_n denotes the set of all possible probability mass functions on S_n which, needless to say, is hardly welcome news! But more disconcerting is the fact that:

$$\Omega(n+1,\pi) := \{(\pi',\delta) : \pi' \in \Pi_n, \delta \in \Delta(n), \pi = P(n,\pi',\delta)\} \tag{14.185}$$

where

- Δ_n denotes the set of all policies (decision-making rules) that are feasible at stage n, namely

$$\Delta(n) := \{\delta : \delta(n,s) \in D(n,s), \forall s \in S_n\} \tag{14.186}$$

- $P(n,\pi',\delta)$ is the probability mass function on S_{n+1} induced by π', δ, and p_n, namely $P(n,\pi',\delta)(s)$ is the probability that state s is observed at stage $n+1$ given that the probability mass function on S_n is π' and that the policy δ is used at stage n. More formally,

$$P(n,\pi',\delta)(s) = \sum_{s' \in S_n} \sum_{\substack{q \in Q_n \\ s=t(n,s',\delta(s'),q)}} p_n(q) \ , \ s \in S_{n+1} \tag{14.187}$$

And to top it all off, $W(n,\pi',\delta)$ denotes the expected value of the return at stage n induced by π' and d, namely

$$W(n,\pi',\delta) := \sum_{s \in S_n} \sum_{q \in Q_n} p_n(q)\pi'(s)w(n,s,\delta(s),q) \tag{14.188}$$

In short, other than demonstrating that a forward formulation can be obtained, not much else was accomplished here due to the messy details associated with this model and its complexity. Indeed, the accepted wisdom in dynamic programming circles is: *thou shalt not attempt forward functional equations for stochastic dynamic programming models.*

Similarly, other but not all *diverging* non-serial dynamic programming models are not amenable to forward decomposition. And, some but not all, *converging* non-serial dynamic programming models are not amenable to backward decomposition.

14.12 Summary and Conclusions

Forward decomposition schemes play an important role in dynamic programming. As we saw in this chapter, the distinction between forward and backward formulations of dynamic programming functional equations is not merely a question of style.

While it is true that in many, perhaps most, cases the a choice between the two schemes is essentially a question of *convenience*, there are cases where the two formulations are markedly different from one another in the sense that one is decidedly superior to the other for the case in question.

Hence, the notion that the difference between these two types of decomposition simply amounts to "a reversal in the direction of the process" is not entirely correct. It is based on a limited experience with "simple" shortest path problems in the context of which the two formulations are indeed essentially identical. But not all problems in this Universe are "simple" shortest path problems!

As we saw, from a modeling point of view, the two key factors here are:

· The structure of the composition operator used to decompose the objective function: is it associative? If not, is it evaluated from "right to left" or "left to right"?

· The dynamics of the state of the process: is the transition function a set-valued function? More generally, is the process serial or non-serial? Is it converging or diverging?

And last but not least is the crucial issue of the implications of the manner in which the objective function is decomposed vs the manner in which the constraints are decomposed for the construction of the state transition function.

14.13 Bibliographic Notes

The distinction between forward and backward dynamic programming models is discussed — at least in a cursory fashion — in most introductory textbooks on dynamic programming (e.g. Nemhauser [1966], Dreyfus and Law [1977], Denardo [2003]).

There are numerous articles on the use of forward formulations, e.g. Seinfeld and Lapidus [1968], Waterman and Byers [1985], Alden and Yano [1986], Nandalal and Bogardi [2007], Pinedo [2008].

For implementations of hybrid backward-forward dynamic programming algorithms, see Lasserre [1985], Tanga et al. [2007] and of course the famous Dijkstra's two-tree algorithm (e.g. Evans and Minieka [1992]).

The investigation into the anomaly discussed in §*14.10* was prompted by a query by Prof. Stuart Dreyfus in the middle 1980s and is based on an unpublished joint paper on this topic.

15

Push!

15.1 Introduction

In this chapter I examine a somewhat neglected solution method for dynamic programming functional equations. It is known in the dynamic programming literature as *Reaching*. I call it here the *Push* method because this term, I submit, reflects more faithfully the method's mode of operation.

The term that I use to designate the "conventional" solution method — discussed in *Chapter 5* and elsewhere in the book — is *Pull.*

As we shall see, one of the most celebrated algorithms in *Operations Research* and *Computer Science,* namely *Dijkstra's Algorithm* for the shortest path problem, can be described as a typical implementation of the *Push* method.

But before I can go into the formal description and analysis of the *Push* method, I need to outline in broad terms the context in which this method is discussed.

Consider then the following typical backward and forward dynamic programming functional equations

Backward equation:

$$f(s) = \min_{x\in D(s)} \{r(s,x) \oplus f(T(s,x))\} \ , \ s \in S' \tag{15.1}$$

Forward equation:

$$f(s) = \min_{(s',x)\in T^{-1}(s)} \{f(s') \oplus r(s',x))\} \ , \ s \in S' \tag{15.2}$$

From an algorithmic point of view, these equations can be regarded as "recipes" for determining the left hand side value, $f(s)$, out of the $f(\cdot)$ values on the right hand side of the respective equation.

So, zooming in on these equations, you "can almost hear" the backward equation calling for the values of $\{f(s') : s' = T(s,x), x \in D(s)\}$, and the forward equation for the values of $\{f(s') : (s',x) \in T^{-1}(s), x \in D(s')\}$.

The question, of course, is by what means would these values be determined. To reveal then the inner working of the recipe that seeks to accomplish this task, consider any state, say $\hat{s} \in S$, and the respective backward and forward functional equations for this state:

Backward equation at $\hat{s}$:

$$f(\hat{s}) = \min_{x\in D(\hat{s})} \{r(\hat{s},x) \oplus f(T(\hat{s},x))\} \tag{15.3}$$

Forward equation at $\hat{s}$:

$$f(\hat{s}) = \min_{(s',x)\in T^{-1}(\hat{s})} \{f(s') \oplus r(s',x))\} \tag{15.4}$$

The conventional *Pull* method seems to argue as follows: at the point that the value of $f(\hat{s})$ is ready to be computed, we would have computed already the values of $f(s)$ for $s \in S'$ that are required on the right hand side of the respective equations. Thus, all that remains is to execute the expression on the right hand side of the respective equation by *"pulling"* from storage/memory the required values of $f(s)$.

So, the conventional *Pull* method is a recipe for obtaining the value of $f(\hat{s})$ by computing the expression on the right hand side of the equation. This is done *once* for each $s \in S$. In this respect we can regard the recipe for the right hand side of functional equation as a single (meta) operation.

This means that implementing this method requires a careful *synchronization* of the order in which the $f(s)$ values are computed to ensure that the values needed on the right hand side of the equation have already been computed, or can be computed, before the value of $f(\hat{s})$ is computed.

There are, however, situations where this is either impossible or highly undesirable. For example, if the underlying state transition model is CYCLIC then such a synchronization is impossible.

Be that as it may, the PUSH method offers an alternative approach to the solution of dynamic programming functional equations.

So, in this chapter I examine this method and what I submit is its most celebrated exponent, namely *Dijkstra's Algorithm* for the shortest path problem. And to set the stage for this discussion, it will be instructive to re-write the two generic functional equations as follows:

Backward equation:

$$F[s] = \min_{x \in D(s)} \{r(s,x) \oplus F[T(s,x)]\} \ , \ s \in S \tag{15.5}$$

Forward equation:

$$F[s] = \min_{(s',x) \in T^{-1}(s)} \{F[s'] \oplus r(s',x))\} \ , \ s \in S \tag{15.6}$$

where S is a discrete finite set and F is an array indexed by the elements of S. The basic assumption is that such an array exists but its content is empty. The task is to compute $F(s)$ for all $s \in S$.

In the backward case, $F[s]$ denotes the length of the shortest distance from node s to the destination node, whereas in the forward case $F[s]$ denotes the length of the shortest path from the origin node to node s.

I shall focus then on the case where S denotes the set of nodes on a graph, $r(i,j)$ denotes the length of arc (i,j) and $F[s]$ is a real number associated with node $s \in S$. In this particular framework $D(s)$ denotes the set of immediate successors of node s, $T(s,x) = x$ and the functional equations can be re-written as follows:

Backward equation:

$$F[s] = \min_{x \in Suc(s)} \{r(s,x) \oplus F[x]\} \ , \ s \in S \tag{15.7}$$

Forward equation:

$$F[s] = \min_{x \in Pred(s)} \{F[x] \oplus r(x,s))\} \ , \ s \in S \tag{15.8}$$

where $Suc(s)$ denotes the set of immediate successors of node s and $Pred(s)$ denotes the set of immediate predecessors of node s.

As usual, in most applications $\oplus = +$, but I shall consider other cases as well.

With this in mind let us now go back to the Pull method and examine it more closely.

15.2 The Pull Method

As indicated above, the *Pull* method regards the dynamic programming functional equation as a recipe for computing the left hand side of the equation. So, to execute this recipe the method sets off a sequence of simple *Pull* operations:

$$\textit{Backward Pull at s: } F[s] \leftarrow \min_{D(s)} \{r(s,x) \oplus F[T(s,x)]\} \tag{15.9}$$

$$\textit{Forward Pull at s: } F[s] \leftarrow \min_{(s',x)\in T^{-1}(s)} \{F[T(s',x)] \oplus r(s',x)\} \tag{15.10}$$

where $\leftarrow$ denotes assignment.

In the case of the shortest path problem these operations are as follows:

$$\textit{Backward Pull at s: } F[s] \leftarrow \min_{x\in Suc(s)} \{r(s,x) \oplus F[x]\} \tag{15.11}$$

$$\textit{Forward Pull at s: } F[s] \leftarrow \min_{x\in Pred(s)} \{F[x] \oplus r(x,s)\} \tag{15.12}$$

Obviously, to carry out these operations the values of $F[\cdot]$ on the right hand side of the assignments must be known.

15.2.1 Example

Consider node $s = 5$ on the graph depicted in Figure 15.1. The Pull operations are as follows:

$$\textit{Backward Pull at } s = 5\textit{: } F[5] \leftarrow \min_{Suc(5)} \{r(5,x) \oplus F[x]\} \tag{15.13}$$

$$\textit{Forward Pull at } s = 5\textit{: } F[5] \leftarrow \min_{x\in Pred(5)} \{F[x] \oplus r(x,5)\} \tag{15.14}$$

Thus, the backward Pull operation at $s = 5$ requires the values of $F[7]$, $F[8]$ and $F[9]$, whereas the forward Pull operation at $s = 5$ requires the values of $F[1]$, $F[2]$ and $F[3]$.

Since the nodes are labeled in an increasing order, F can be computed in an orderly manner by the relevant Pull operations. In the case of the backward operation, we begin at $s = 9$ and then conduct the operations for

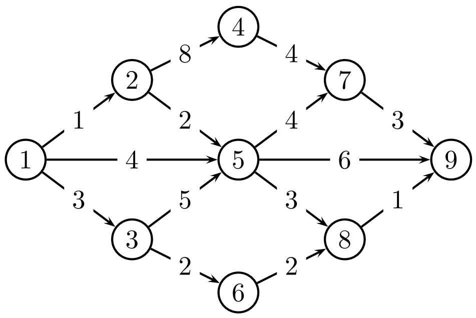


Figure 15.1: A simple acyclic shortest path problem

$s = 8, 7, \ldots, 1$ — in this order. In the case of the forward Pull, we begin at $s = 1$ and then conduct the operations for $s = 2, 3, \ldots, 9$ — in this order.

Consider the case where $\oplus = +$. Since the identity element of $+$ is equal to 0, in the case of the backward Pull operation we set $F[9] = 0$ and in the case of the forward operation we set $F[1] = 0$.

The remaining values of F computed impromptu by the Pull operations are shown in Figure 15.2 next to the respective nodes.

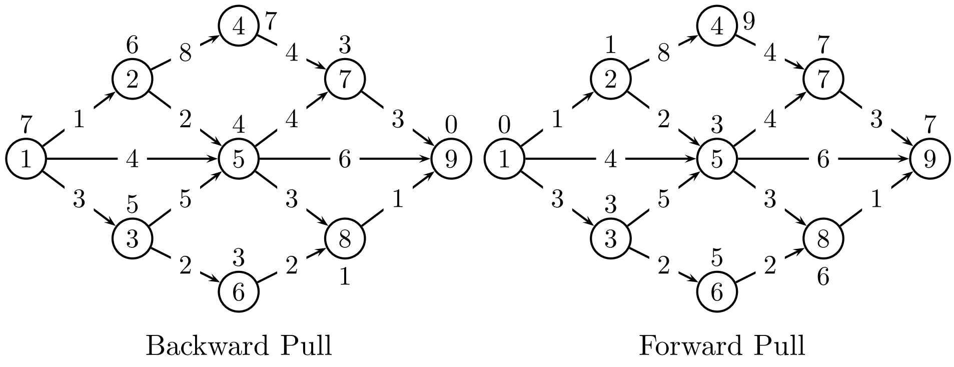


Figure 15.2: Summary of Pull operations

To illustrate, for $s = 5$ the backward Pull operations yields

$$\begin{aligned} F[5] &\leftarrow \min_{x \in Suc(5)} \{r(5,x) + F[x]\} \\ &= \min_{x \in \{7,8,9\}} \{r(5,x) + F[x]\} \\ &= \min \{r(5,7) + F[7]\ ,\ r(5,8) + F[8]\ ,\ r(5,9) + F[9]\} \\ &= \min \{4+3, 3+1, 6+0\} = \min \{7,4,6\} = 4 \end{aligned}$$

The forward Pull operation yields

$$\begin{aligned} F[5] &\leftarrow \min_{x \in Pred(5)} \{r(5,x) + F[x]\} \\ &= \min_{x \in \{1,2,3\}} \{r(5,x) + F[x]\} \end{aligned}$$

$$= \min \{F[1] + r(1,5)\ ,\ F[2] + r(2,5)\ ,\ F[3] + r(3,5)\}$$
$$= \min \{0 + 4, 1 + 2, 3 + 5\} = \min \{4, 3, 8\} = 3$$

Note that in the backward case $F[s]$ is the length of the shortest path from node s to node node 9, whereas in the forward case $F[s]$ is the length of the shortest path from node 1 to node s. So it is hardly surprising that backward $F[1]$ is equal to the forward $F[9]$. □

In cases where the Pull operation is incorporated in a *successive approximation* procedure, it may prove necessary to conduct a number of Pull operations at the same state s. This means that the order in which these operations are conducted can be critical.

15.2.2 Example

Consider the cyclic shortest path problem depicted in Figure 15.3 and assume that the objective is to find the shortest path from node 1 to node 9.

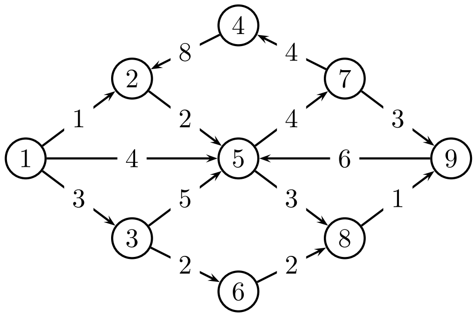


Figure 15.3: A simple cyclic shortest path problem

Since the depicted process is cyclic, the direct method cannot be used in this case. Instead, we need to apply the successive approximation method tailored for a backward functional equation.

Thus, we proceed as follows; given that all the arcs are non-negative, the length of the shortest path from node 9 to itself is equal to zero, we pick the approximation F where $F[9] = 0$ and $F[s] = \infty]$ for all the other nodes.

We then proceed to execute the Pull operations at the nodes in reverse order. That is starting at node 8, we proceed to node 7, then to node 6 and so on, completing the first update at node 1. The results are shown in Figure 15.4 where the updated values of $F[s]$ are shown next to the respective nodes. Note that the second approximation is a significant improvement on the initial approximation.

We then apply this procedure again, conducting the backward Pull operation from node 8 to node 1, to obtain the third approximation. The results are shown in Figure 15.5. Note the improvement in the value of $F[4]$.

We repeat this procedure to obtain the fourth approximation. As there was no change in the approximation we stop.

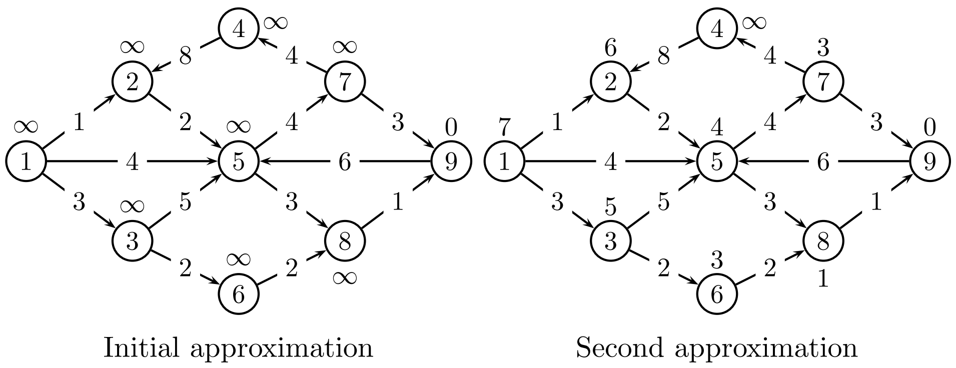


Figure 15.4: First iteration of the successive approximation method

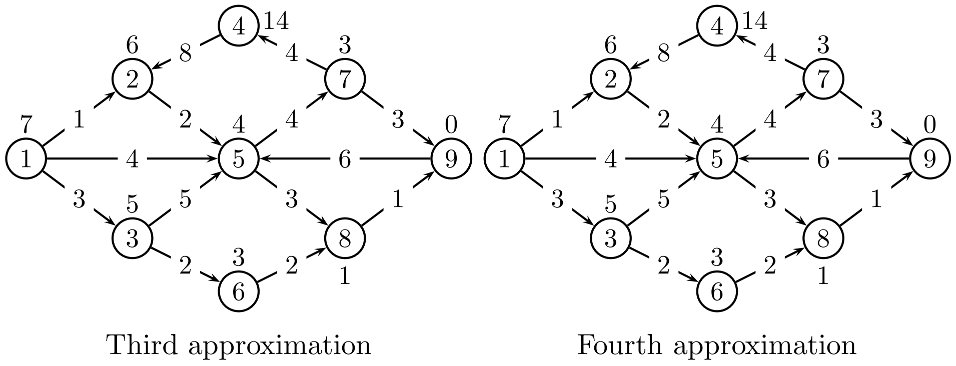


Figure 15.5: Second and third iteration

To establish the method's performance in the reverse order, simply repeat the entire exercise, by conducting the backward Pull operations from node 1 to node 8. The results are shown in Figure 15.6.

> **Remark:** Although in this case the process is cyclic, it can be easily determined, by inspection, that most of the directed arcs have a forward orientation, that is, that arc(i,j) is typically such that $j > i$. So, it is obvious that here it makes sense to conduct the Pull operations from node 8 to node 1. But, in large cyclic problems determining in what order to conduct the Pull operations may prove difficult. Note then that the *Pull* method *per se* does not provide instruction on this matter as it does not deal with the practical question of the order of operations. $\square$

And before turning to the discussion of the *Pull* method's rival, the *Push* method, I want to point out that whereas the *Pull* method is clearly a direct implementation of the functional equation: it so to speak "animates" the equation's recipe for computing the values of $F[\cdot]$, the *Push* method, takes a considerably more elaborate approach to the solution of the equation. For although the *Push* is similarly inspired by the structure of the dynamic pro-

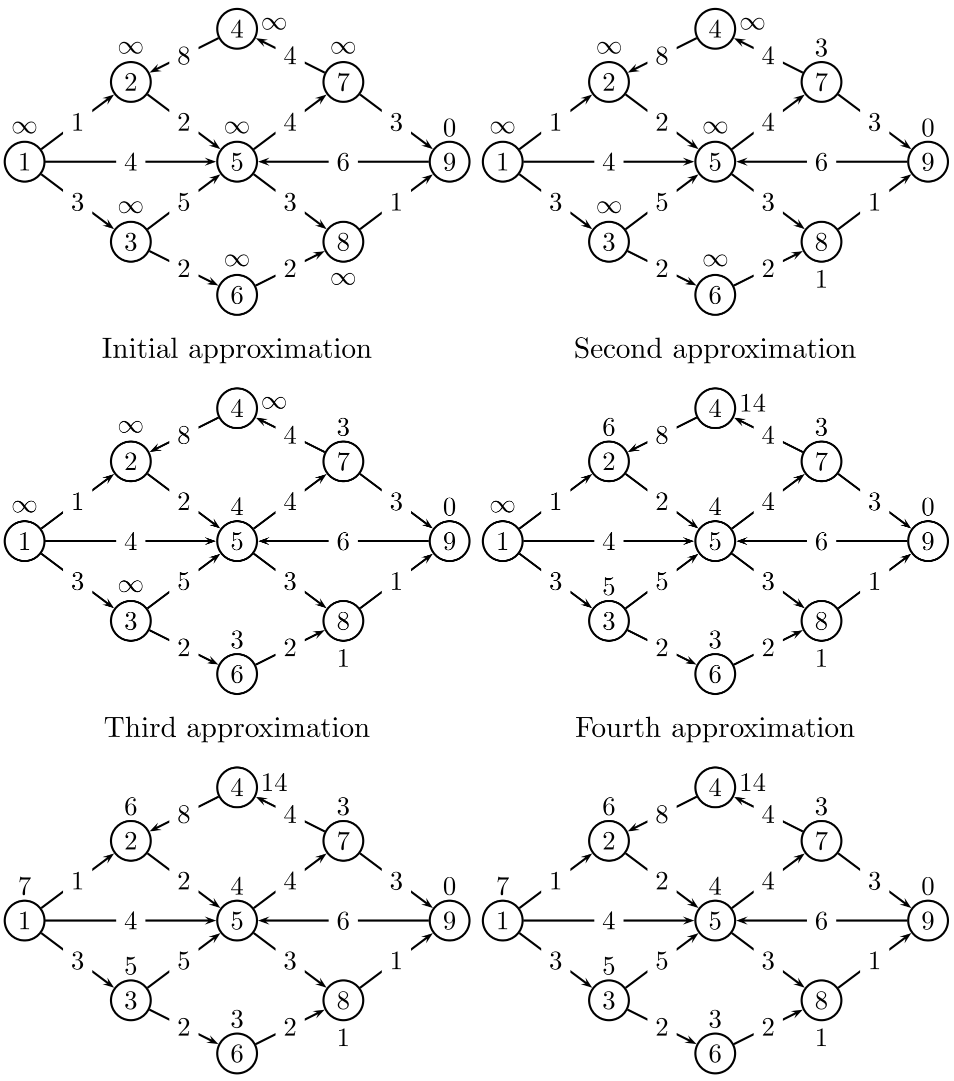


Figure 15.6: First iteration of the successive approximation method

gramming functional equation, the procedure that it lays out for computing the values of $F[\cdot]$ consists of more than a simple “verbatim” implementation of the recipe.

15.3 The Push Method

The thinking underlying this approach can be summed up as follows: suppose that we have an approximation F for the exact value of f and the value of $F[s]$ has just been improved for some state s. Surely, this information can be used to update (improve) the $F[\cdot]$ values of states that are linked to s via the transition function T, hence via the functional equation.

So, the update is carried out as follows:

Backward Push at s:

$$F[s'] \leftarrow \min\ \{F[s'], r(s',x) \oplus F[s]\}\ ,\ \forall (s',x) \in T^{-1}(s) \tag{15.15}$$

Forward Push at s:

$$F[s'] \leftarrow \min\ \{F[s'], F[s] \oplus r(s,x)\}\ ,\ \forall x \in D(s), s' = T(s,x) \tag{15.16}$$

Thus, in the case of the shortest path problem, the Push operations are as follows:

Backward Push at s:

$$F[s'] \leftarrow \min\ \{F[s'], r(s',s) \oplus F[s]\}\ ,\ \forall s' \in Pred(s) \tag{15.17}$$

Forward Push at s:

$$F[s'] \leftarrow \min\ \{F[s'], F[s] \oplus r(s,s')\}\ ,\ \forall s' \in Suc(s) \tag{15.18}$$

Note that whereas a *Pull* operation at s updates/computes the value of $F[s]$, a *Push* operation at s updates the $F[\cdot]$ values of its immediate successors (forward Push) or immediate predecessors (backward Push). For this reason, a *Push* operation is conducted at s only after an improvement in the value of $F[s]$ has been achieved.

The next example illustrates the mechanics of the *Push* method. Subsequent examples will explain its rationale and scope of operation.

15.3.1 Example

Consider again the acyclic shortest path problem depicted in Figure 15.7, assuming that the objective is to find the shortest path from node 1 to node 9.

We solve the problem using a successive approximation procedure based on the forward *Push* operation. The initial approximation is an array F with $F[1] = 0$ and $F[s] = \infty, j = 2, \ldots, 9$.

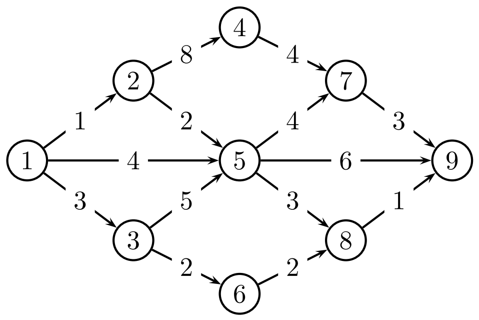


Figure 15.7: A simple acyclic shortest path problem

Figure 15.9 summarizes the results of seven Push operations at $s = 1, 2, \ldots, 7$, respectively — in this order. The final Push operation, at $s = 8$, is shown in Figure 15.8.

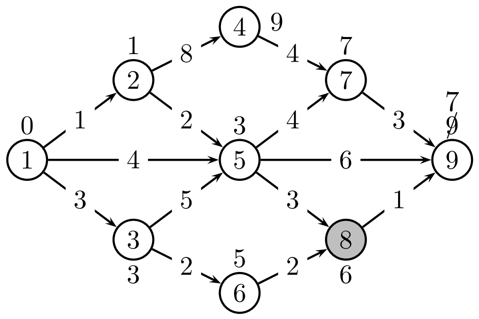


Figure 15.8: Final Push operation at $s = 8$

Since $Suc(8) = \{9\}$, this operation will update only one $F[\cdot]$ value, namely the value of $F[9]$. Indeed, it improves (decreases) the current value of $F[9]$, which is equal to 9, to 7 by considering the path from node 1 to node 8, whose length is equal to 6, and the arc from node 8 to node 9, whose length is equal to $d(8, 9) = 1$.

Formally,

$$F[9] \leftarrow \min \{F[9], F[8] + d(8, 9)\} = \min \{9, 6 + 1\} = 7$$

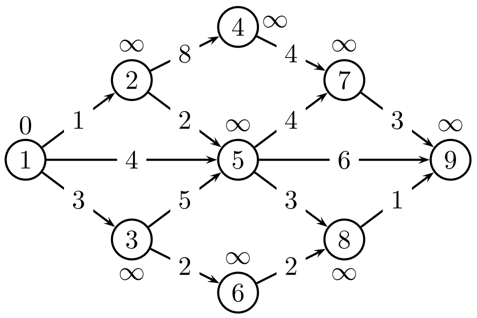
A more exciting update is the Push operation at $s = 5$:

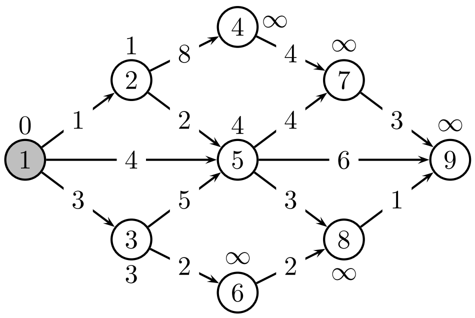
$$F[s'] \leftarrow \min \{F[s'], F[5] + d(5, s)\} \ , \ s' \in Suc(5) = \{7, 8, 9\}$$

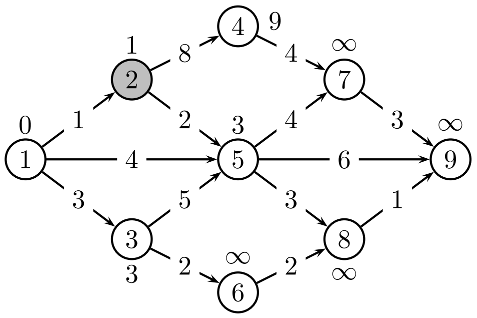
that yields

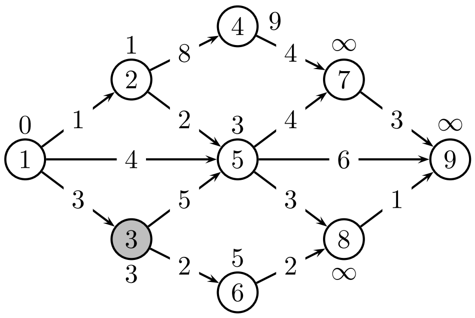
$$\begin{aligned} F[7] &\leftarrow \min \{F[7], F[5] + d(5, 7)\} = \min\{13, 3 + 4\} = 7 \\ F[8] &\leftarrow \min \{F[8], F[5] + d(5, 8)\} = \min\{\infty, 3 + 3\} = 6 \\ F[9] &\leftarrow \min \{F[9], F[5] + d(5, 7)\} = \min\{\infty, 3 + 6\} = 9 \end{aligned}$$

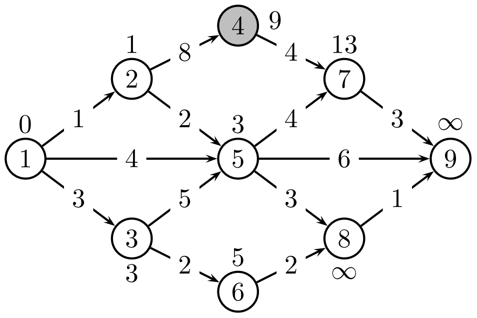
The length of the shortest path from node 1 to node 9 is then equal to $F[9] = 7$ and the shortest path is $(1, 2, 5, 8, 9)$. □

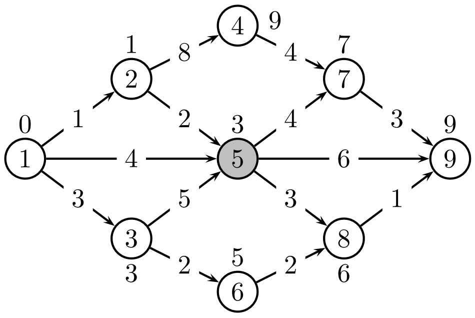

1. Initial approximation

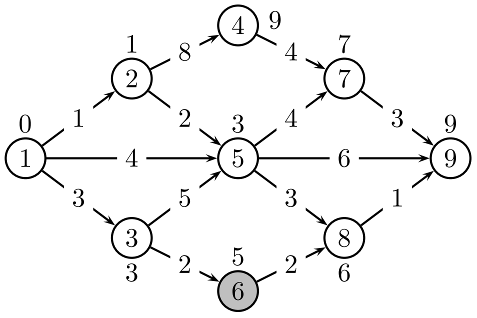

2. A forward Push at $s = 1$


3. A forward Push at $s = 2$


4. A forward Push at $s = 3$


5. A forward Push at $s = 4$


6. A forward Push at $s = 5$


7. A forward Push at $s = 6$

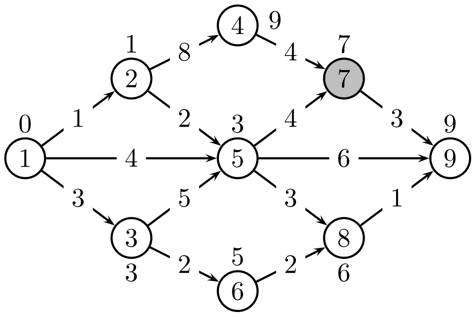

8. A forward Push at $s = 7$

Figure 15.9: Successive approximation via forward Push

The next example illustrates a situation where Push operations are used not only to update the approximation of f, but also to generate the state space S. In this framework the entire procedure can be regarded as an IMPLICIT ENUMERATION procedure.

15.3.2 Example

Consider the following simple unbounded knapsack problem:

$$z^* := \max_{x_1,x_2,x_3,x_4} \{120x_1 + 99x_2 + 83x_3 + 55x_4\} \tag{15.19}$$

$$63x_1 + 56x_2 + 45x_3 + 40x_4 \leq 137 \tag{15.20}$$

$$x_1, x_2, x_3, x_4 \in \{0, 1, 2, \dots\} \tag{15.21}$$

Set

$$\mathbf{v} = (120, 99, 83, 55) \ ; \ \mathbf{w} = (63, 56, 45, 40) \ ; \ W = 137. \tag{15.22}$$

Let $f(s)$ denote the maximum value of the objective function when the right hand side value W is equal to s instead of 137 and the $\leq$ constraint is changed to an $=$ constraint, namely define

$$f(s) := \max_{x_1,x_2,x_3,x_4} \{120x_1 + 99x_2 + 83x_3 + 55x_4\} \tag{15.23}$$

$$63x_1 + 56x_2 + 45x_3 + 40x_4 = s \tag{15.24}$$

$$x_1, x_2, x_3, x_4 \in \{0, 1, 2, \dots\} \tag{15.25}$$

and regard s as a parameter taking values in the set $S := \{0, 1, 2, \dots, 137\}$. For convenience we let $f(s) = -\infty$ if the problem associated with s has no feasible solutions.

Note that $z^* = \max \{f(s) : s \in S\}$ so that once we have the values of $f(s)$ for all $s \in S$, it is easy to determine the value of z^*. The reason that we define f this way is explained below.

In any case, the dynamic programming functional equation for f is as follows:

$$f(s) = \max_{\substack{1 \leq j \leq 4 \\ w_j \leq s}} \{f(s - w_j) + v_j\} \ , \ s \in S \tag{15.26}$$

with $f(s) = 0$.

We shall solve this functional equation by the successive approximation procedure employing a sequence of forward *Push* operations (updates):

$$F[s'] \leftarrow \max \{F[s'], F[s] + v_j\} \ , \ s' = s + w_j \leq W, j = 1, 2, 3, 4 \tag{15.27}$$

where F is an array consisting of $W + 1$ elements, indexed by the state variable $s \in S$.

The mechanics of this generic operation can be spelled out more explicitly as follows:

Forward Push at $s \in S$:

```
For $j = 1, 2, 3, 4$ Do:
    $s' = s + w_j$
    If $s' \leq W$ Do:

        $F[s'] \leftarrow \max\ \{F[s'], F[s] + v_j\}$        (15.28)

    End Do
End Do
```

Since opt = max, we initialize $F[s]$ as follows:

$$F[s] \leftarrow \begin{cases} 0 & , \quad s = 0 \\ -\infty & , \quad s = 1, \ldots, 137 \end{cases} \tag{15.29}$$

All the states for which an improvement in the $F[\cdot]$ value is achieved are recorded. The procedure terminates then when the list of recorded states is empty. The next *Push* operation is conducted at the *smallest* state on the list. The initial list is the singleton $\mathcal{L} = \{0\}$.

Iteration #1.
$\mathcal{L} = \{0\}$ so we conduct a Push operation at $s = 0$. This yields

$$F[0 + w_j] \leftarrow \max\ \{F[0 + w_j], F[0] + v_j\}\ ,\ j = 1, 2, 3, 4$$

namely,

$$\begin{aligned} F[63] &\leftarrow \max\ \{F[63], 0 + 120\} = \max\ \{-\infty, 120\} = 120 \\ F[56] &\leftarrow \max\ \{F[56], 0 + 99\} = \max\ \{-\infty, 99\} = 99 \\ F[45] &\leftarrow \max\ \{F[45], 0 + 83\} = \max\ \{-\infty, 83\} = 83 \\ F[40] &\leftarrow \max\ \{F[40], 0 + 55\} = \max\ \{-\infty, 55\} = 55 \end{aligned}$$

So we adjust $\mathcal{L}$ and set $\mathcal{L} = \{40, 45, 56, 63\}$.

Iteration #2.
$\mathcal{L} = \{40, 45, 56, 63\}$ so we conduct a Push operation at $s = 40$. This yields

$$F[40 + w_j] \leftarrow \max\ \{F[40 + w_j], F[40] + v_j\}\ ,\ j = 1, 2, 3, 4$$

namely

$$\begin{aligned} F[103] &\leftarrow \max\ \{F[103], 55 + 120\} = \max\ \{-\infty, 175\} = 175 \\ F[96] &\leftarrow \max\ \{F[96], 55 + 99\} = \max\ \{-\infty, 154\} = 154 \\ F[85] &\leftarrow \max\ \{F[85], 55 + 83\} = \max\ \{-\infty, 138\} = 138 \\ F[80] &\leftarrow \max\ \{F[80], 55 + 55\} = \max\ \{-\infty, 110\} = 110 \end{aligned}$$

We adjust $\mathcal{L}$ and set $\mathcal{L} = \{45, 56, 63, 80, 85, 96, 103\}$.

Iteration #3.
$\mathcal{L} = \{45, 56, 63, 80, 85, 96, 103\}$ so we conduct a Push operation at $s = 45$. This yields

$$F[45 + w_j] \leftarrow \max\{F[45 + w_j], F[45] + v_j\}\ ,\ j = 1, 2, 3, 4$$

namely

$$\begin{aligned}
&F[108] \leftarrow \max\{F[108], 83 + 120\} = \max\{-\infty, 203\} = 203\\
&F[101] \leftarrow \max\{F[101], 83 + 99\} = \max\{-\infty, 182\} = 182\\
&F[90] \leftarrow \max\{F[90], 83 + 83\} = \max\{-\infty, 166\} = 166\\
&F[85] \leftarrow \max\{F[85], 83 + 55\} = \max\{138, 138\} = 138
\end{aligned}$$

We adjust $\mathcal{L}$ and set $\mathcal{L} = \{56, 63, 80, 85, 90, 96, 101, 103, 108\}$.
Iteration #4.
$\mathcal{L} = \{56, 63, 80, 85, 90, 96, 101, 103, 108\}$ so we conduct a Push operation at $s = 56$. This yields

$$F[56 + w_j] \leftarrow \max\{F[56 + w_j], F[56] + v_j\}\ ,\ j = 1, 2, 3, 4$$

namely

$$\begin{aligned}
&F[119] \leftarrow \max\{F[119], 99 + 120\} = \max\{-\infty, 219\} = 219\\
&F[112] \leftarrow \max\{F[112], 99 + 99\} = \max\{-\infty, 198\} = 198\\
&F[101] \leftarrow \max\{F[101], 99 + 83\} = \max\{182, 182\} = 182\\
&F[96] \leftarrow \max\{F[96], 99 + 55\} = \max\{154, 154\} = 154
\end{aligned}$$

We adjust $\mathcal{L}$ and set $\mathcal{L} = \{63, 80, 85, 90, 96, 101, 103, 108, 112, 119\}$.
Iteration #5.
$\mathcal{L} = \{63, 80, 85, 90, 96, 101, 103, 108, 112, 119\}$ so we conduct a Push operation at $s = 63$. This yields

$$F[63 + w_j] \leftarrow \max\{F[63 + w_j], F[63] + v_j\}\ ,\ j = 1, 2, 3, 4$$

namely

$$\begin{aligned}
&F[126] \leftarrow \max\{F[126], 120 + 120\} = \max\{-\infty, 240\} = 240\\
&F[119] \leftarrow \max\{F[119], 120 + 99\} = \max\{219, 219\} = 219\\
&F[108] \leftarrow \max\{F[108], 120 + 83\} = \max\{203, 203\} = 203\\
&F[103] \leftarrow \max\{F[103], 120 + 55\} = \max\{175, 175\} = 175
\end{aligned}$$

We adjust $\mathcal{L}$ and set $\mathcal{L} = \{80, 85, 90, 96, 101, 103, 108, 112, 119, 126\}$.
Iteration #6.
$\mathcal{L} = \{80, 85, 90, 96, 101, 103, 108, 112, 119, 126\}$ so we conduct a Push operation at $s = 80$. This yields

$$F[80 + w_j] \leftarrow \max\{F[80 + w_j], F[80] + v_j\}\ ,\ j = 2, 3, 4$$

namely

$$F[136] \leftarrow \max\{F[136], 110+99\} = \max\{-\infty, 209\} = 209$$
$$F[125] \leftarrow \max\{F[125], 110+83\} = \max\{-\infty, 193\} = 193$$
$$F[120] \leftarrow \max\{F[120], 110+55\} = \max\{-\infty, 165\} = 165$$

We adjust $\mathcal{L}$ and set $\mathcal{L} = \{85, 90, 96, 101, 103, 108, 112, 119, 120, 125, 126, 136\}$.

Iteration #7.

$\mathcal{L} = \{85, 90, 96, 101, 103, 108, 112, 119, 120, 125, 126\}$ so we conduct a Push operation at $s = 85$. This yields

$$F[85 + w_j] \leftarrow \max\{F[85 + w_j], F[85] + v_j\}\ ,\ j = 3, 4$$

namely

$$F[130] \leftarrow \max\{F[130], 138+83\} = \max\{-\infty, 221\} = 221$$
$$F[125] \leftarrow \max\{F[125], 138+55\} = \max\{193, 193\} = 193$$

We adjust $\mathcal{L}$ and set $\mathcal{L} = \{90, 96, 101, 103, 108, 112, 119, 120, 125, 126, 130, 136\}$.

Iteration #8.

$\mathcal{L} = \{90, 96, 101, 103, 108, 112, 119, 120, 125, 126, 130, 136\}$ so we conduct a Push operation at $s = 90$. This yields

$$F[90 + w_j] \leftarrow \max\{F[90 + w_j], F[90] + v_j\}\ ,\ j = 3, 4$$

namely

$$F[135] \leftarrow \max\{F[135], 166+83\} = \max\{-\infty, 249\} = 249$$
$$F[130] \leftarrow \max\{F[130], 166+55\} = \max\{221, 221\} = 221$$

We adjust $\mathcal{L}$ and set $\mathcal{L} = \{96, 101, 103, 108, 112, 119, 120, 125, 126, 130, 135, 136\}$.

Iteration #9.

$\mathcal{L} = \{96, 101, 103, 108, 112, 119, 120, 125, 126, 130, 135, 136\}$ so we conduct a Push operation at $s = 96$. This yields

$$F[96 + w_j] \leftarrow \max\{F[96 + w_j], F[90] + v_j\}\ ,\ j = 4$$

namely

$$F[136] \leftarrow \max\{F[136], 154+55\} = \max\{209, 209\} = 209$$

We adjust $\mathcal{L}$ and set $\mathcal{L} = \{101, 103, 108, 112, 119, 120, 125, 126, 130, 135, 136\}$.

Iteration #10.

$\mathcal{L} = \{101, 103, 108, 112, 119, 120, 125, 126, 130, 135, 136\}$ so we conduct a Push operation at $s = 101$. Since $s + w_j > W$ for all $j = 1, 2, 3, 4$, no Push operation is performed.

We adjust $\mathcal{L}$ and set $\mathcal{L} = \{103, 108, 112, 119, 120, 125, 126, 130, 135, 136\}$.

s	$F[s]$	s	$F[s]$
0	0	103	175
40	55	108	203
45	83	112	198
56	99	119	219
63	120	120	165
80	110	125	193
85	138	126	240
90	166	130	221
96	154	135	249
101	182	136	209

Table 15.1: Results generated by the Push Method

This will be the case for all the remaining elements of $\mathcal{L}$, so we ... halt the Push operations, thus terminating the successive approximation procedure. The results are summarized in Table 15.1. The values of $F[s]$ for states that are not listed in this table are equal to $-\infty$, indicating that the problem defined by (15.23)-(15.25) is not feasible.

Observe that the state space S consists of 138 states but only 20 values of $f(\cdot)$ were computed. The point is that most of the states in $S = \{0, 1, 2, \dots, 137\}$ are not feasible in the sense that they cannot be reached from the initial state $s = 0$ by a sequence of feasible decisions.

To recover an optimal solution to the problem of interest, note that the maximum value of $f(s)$ over S is attained for $s = 135$ and is equal to 249. This means that $z^* = 249$.

The optimal solution generated by the dynamic programming functional equation for $s = 135$ is $x = (0, 0, 3, 0)$. □

In the next section I examine a class of problems, for which the *Push* Method proves a particularly suitable solution method.

15.4 Monotone Accumulated Return Processes

There are many cases where the *partial returns* generated by a sequential decision process are *monotone*: the accumulated return is either non-increasing or non-decreasing as the decisions are made.

The most famous case is a shortest path problem where the arc lengths are positive. In such a case the length of a sub-path is non-decreasing as arcs are appended to it.

More generally, in the case of multistage decision models we say that the *accumulated* return process is MONOTONE INCREASING if for any $n \in \mathbb{N}$ and

$s_n \in S_n$ we have

$$g_n(s_n, x_n, x_{n+1}, \dots, x_N) > g_{n+1}(s_{n+1}, x_{n+1}, x_{n+2}, \dots, x_N) \tag{15.30}$$

for all $(x_n, \dots, x_N) \in X(n, s_n)$, observing that this can also be written as

$$\rho(n, s_n, x_n, r) > r \ , \ r = g_{n+1}(s_{n+1}, x_{n+1}, x_{n+2}, \dots, x_N) \tag{15.31}$$

Furthermore, if ρ admits of a binary representation, then this boils down to

$$w(n, s_n, x_n) \oplus r > r \ , \ r = g_{n+1}(s_{n+1}, x_{n+1}, x_{n+2}, \dots, x_N) \tag{15.32}$$

For instance, this will be the case if $\oplus = +$ and the stage return function w yields strictly positive values; or if $\oplus = \times$ and the stage return function yields values strictly greater than 1.

Similarly, we say that the *accumulated* return process is MONOTONE NON-DECREASING if for any $n \in \mathbb{N}$ and $s_n \in S_n$ we have

$$g_n(s_n, x_n, x_{n+1}, \dots, x_N) \geq g_{n+1}(s_{n+1}, x_{n+1}, x_{n+2}, \dots, x_N) \tag{15.33}$$

for all $(x_n, \dots, x_N) \in X(n, s_n)$, observing that this can also be written as

$$\rho(n, s_n, x_n, r) \geq r \ , \ r = g_{n+1}(s_{n+1}, x_{n+1}, x_{n+2}, \dots, x_N) \tag{15.34}$$

Furthermore, if ρ admits of a binary representation, then this boils down to

$$w(n, s_n, x_n) \oplus r \geq r \ , \ r = g_{n+1}(s_{n+1}, x_{n+1}, x_{n+2}, \dots, x_N) \tag{15.35}$$

For instance, this will be the case if $\oplus = +$ and the stage return function w yields non-negative values.

Thus, MONOTONE DECREASING and MONOTONE NON-INCREASING accumulation schemes are defined in a similar fashion. For instance, if $\oplus = +$ and the stage return function w yields strictly negatives values then the accumulated return process is monotone decreasing. This will also be the case if $\oplus = \times$ and w yields values in $[0, 1]$.

Remark:
Note the distinction between

· The binary operator $\oplus$ being monotone with respect to its two arguments.

· The *accumulated* return process induced by $\oplus$ being monotone.

For example, the binary operator $+$ is strictly increasing with respect to its two arguments regardless of whether the arguments take positive/negative values. But the nature of the *accumulated* return process induced by $+$ is crucially dependent on whether the arguments are positive or negative.

As another example, consider the binary min operator, $\lfloor$. This operator is *non-decreasing* with respect to its two arguments, namely

$$a \geq a' \implies a\lfloor b \ \geq \ a'\lfloor b \ , \ \forall b \in \mathbb{R} \tag{15.36}$$

$$b \geq b' \implies a\lfloor b \ \geq \ a\lfloor b' \ , \ \forall a \in \mathbb{R} \tag{15.37}$$

On the other hand, the *accumulated* return process induced by $\lfloor$ is monotone *non-increasing,* namely

$$a\lfloor b \ \leq \ a \ , \ \forall a, b \in \mathbb{R} \tag{15.38}$$

$$a\lfloor b \ \leq \ b \ , \ \forall a, b \in \mathbb{R} \tag{15.39}$$

It is important to keep the distinction between the two notions of monotonicity in mind. One has to do with changes in the result of $a \oplus b$ as we vary one of the arguments of $\oplus$, the other has to do with the relationship between the value of the result of $a \oplus b$ and the values of the arguments a and b. $\square$

Now, consider the following forward dynamic programming functional equation

$$f(s) = \min_{(s',x) \in T^{-1}(s)} \{f(s') \oplus w(s',x)\} \ , \ s \in S \tag{15.40}$$

If $\oplus$ induces a non-decreasing accumulated return process, then clearly

$$f(s) < f(s') \implies f(s) < f(s') \oplus w(s',x) \ , \ \forall x \in D(s) \tag{15.41}$$

and therefore the functional equation under consideration can be written as follows:

$$f(s) = \min_{\substack{(s',x) \in T^{-1}(s) \\ f(s') \leq f(s)}} \{f(s') \oplus w(s',x)\} \ , \ s \in S \tag{15.42}$$

This means that if the number of states is finite, namely if $|S| < \infty$, then the *Push* method can be used to solve the dynamic programming functional equation in a *non-decreasing* order relative to the values of $f(\cdot)$. All we have to do to accomplish this task is to apply the *Push* operation according to the order determined by the $f(\cdot)$ values, commencing at the initial state of the process.

Similarly, consider the functional equation

$$f(s) = \max_{(s',x) \in T^{-1}(s)} \{f(s') \oplus w(s',x)\} \ , \ s \in S \tag{15.43}$$

If $\oplus$ induces a non-increasing accumulated return process, then clearly

$$f(s) > f(s') \implies f(s) > f(s') \oplus w(s',x) \ , \ \forall x \in D(s) \tag{15.44}$$

and therefore the functional equation under consideration can be written as follows:

$$f(s) = \max_{\substack{(s',x)\in T^{-1}(s)\\ f(s')\geq f(s)}} \{f(s') \oplus w(s',x)\} \ , \ s \in S \tag{15.45}$$

This means that if the number of states is finite, namely if $|S| < \infty$, then the *Push* method can be used to solve the dynamic programming functional equation in a *non-increasing* order relative to the values of $f(\cdot)$. All we have to do to accomplish this task is to apply the *Push* operation according to the order determined by the $f(\cdot)$ values, commencing at the initial state of the process.

The following definition summarizes the discussion on the desired relationship between the optimization criterion opt and the monotonicity of the accumulated return process generated by the binary composition operator $\oplus$.

Definition 15.4.1 *The accumulated return process induced by $\oplus$ is said to be* MONOTONE *with respect to* opt *and the decomposition scheme under consideration if it satisfies the relevant condition listed below, where a and b represent* RELEVANT *values of the returns.*

- opt = min*:*
  - *Forward decomposition:* $a \oplus b \geq a$
  - *Backward decomposition:* $a \oplus b \geq b$
- opt = max*:*
  - *Forward decomposition:* $a \oplus b \leq a$
  - *Backward decomposition:* $a \oplus b \leq b$ □

Take special note of the clause "RELEVANT values of the returns". The point is that in some cases the required conditions are satisfied only for certain values of the arguments a and b. For instance, as noted above, if $\oplus = +$, and opt = min, then the relevant values of the returns should be non-negative, whereas if $\oplus = +$, and opt = max, then the relevant values of the returns should be non-positive.

The following example features a longest path problem with $\oplus = \lfloor$, recalling that $a \lfloor b = \min\{a, b\}$, thus the accumulated return process induced by $\oplus$ is monotone with respect to opt = max regardless of the values the returns take.

15.4.1 Example

Consider the graph depicted in Figure 15.10 and the following problem associated with it:

> Find the path from node 1 to node 9 whose shortest arc is the longest possible.

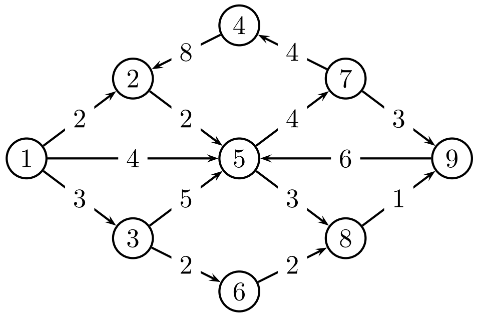


Figure 15.10: Longest shortest arc problem

That is, the length of a path in this case is the length of the worst (shortest) arc on the path. For instance, the length of the path $(1, 3, 5, 8, 9)$ is equal to 1 because the shortest arc on this path is $(8, 9)$ whose length is equal to 1. These problems are called "longest/shortest arc problems". They represent important practical worst-case (bottleneck) problems.

Formally, the problem can be embedded in the following family of problems:

Problem $P(s), s \in S := \{1, 2, \ldots, 9\}$:

$$f(s) := \max_{\substack{x_1,\ldots,x_k \\ k}} \min \{d(x_1, x_2), d(x_2, x_3), \ldots, d(x_{k-1}, x_k)\} \tag{15.46}$$

$$x_1 = s_1 = 1 \tag{15.47}$$

$$x_k = s \tag{15.48}$$

$$x_j \in Suc(s_j) \ , \ j = 2, 3, \ldots, k \tag{15.49}$$

$$s_{j+1} = x_j \ , \ j = 1, 2, \ldots, k \tag{15.50}$$

By definition, $f(s)$ is the length of the longest possible shortest arc on a path from node 1 to node s. The objective is to find an optimal solution to *Problem* $P(9)$.

The forward dynamic programming functional equation is as follows:

$$f(s) = \max_{x \in Pred(s)} \{f(x) \lfloor d(x, s)\} \ , \ s \in S \tag{15.51}$$

Note that because $Pred(1)$ is empty and $\oplus = \lfloor$, it is convenient to set $f(1) = \infty$.

Now, since $\lfloor$ induces a monotone *non-increasing* accumulated return process, the functional equation can be written as follows:

$$f(s) = \max_{\substack{x \in Pred(s) \\ f(s) < f(x)}} \{f(x) \lfloor d(x, s)\} \ , \ s \in S \tag{15.52}$$

The forward Push operation at node s is as follows:

$$F[x] \leftarrow \max \{F[x] \, , \, F[s] \lfloor d(s, x)\} \ , \ x \in Suc(s) \tag{15.53}$$

If at this point an estimate $F[x]$ for $f(x)$ is not available for node x on the right hand side of the assignment, the identity element of max, namely $-\infty$ is invoked, in which case the updated value of $F[x]$ on the left hand side of the assignment will be equal to $d(s,x)$, the length of arc (s,x).

To generate the values of $f(s)$ in a non-increasing order, a sequence of Push operation is set off, starting at the origin node $s_1 = 1$, where $f(1) = \infty$. The expectation is that some of the values of the $f(\cdot)$ values will not be computed because they are smaller than $f(9)$.

In view of this, our book-keeping operation is organized as follows: in each iteration a distinction is drawn between three types of nodes:

· A PROCESSED node is a node where a Push operation has already occurred.
· An ACTIVE node is a node whose $F[\cdot]$ value has already been modified by a Push operation, but has not yet been processed.
· An INACTIVE node is a node that has not yet been affected by a Push operation.

In the framework of this convention, the Push operation at node s can be restated as follows:

$$F[x] \leftarrow \begin{cases} d(s,x) & , x \text{ is inactive} \\ \max\,\{F[x], F[s] \lfloor d(s,x)\} & , x \text{ is active} \end{cases} \quad , x \in Suc(s) \qquad (15.54)$$

Initially there are no processed nodes and there is only one active node: the origin node, $s = 1$. The next Push operation is conducted at an active node whose $F[\cdot]$ value is the largest. After the operation the node's status changes from *active* to *processed.* Then the next node is selected for a Push operation according to the same criterion, and so on. The Push operation terminates when the destination node is selected for processing.

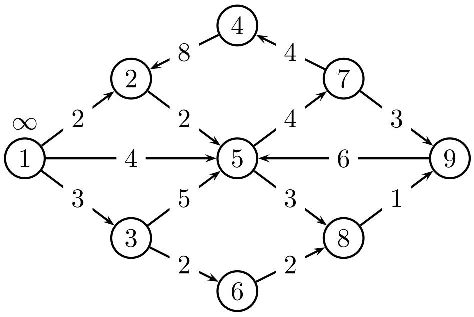
If the objective is to determine the $f(\cdot)$ values for all the nodes, then termination occurs when there are no active nodes. Nodes that are still inactive after termination are not reachable from the origin node.

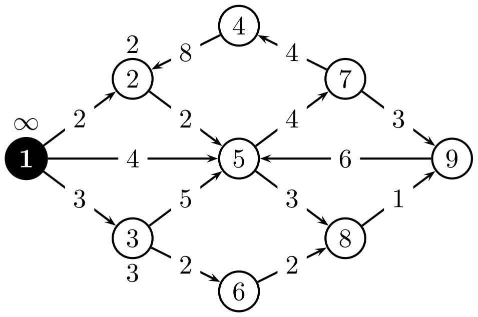
To distinguish between the processed and active nodes on a graph, the former are shaded. The active node at which the Push operation is conducted is shaded black. The $F[\cdot]$ values of processed and active nodes are displayed on the graph next to the respective nodes.

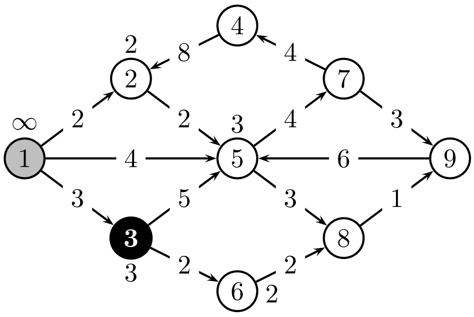
The results generated by the Push operations are displayed in Figure 15.11. Observe that after the first Push operation (at node 1) there is one processed node, namely node 1, and there are three active nodes, namely the three immediate successors of node 1.

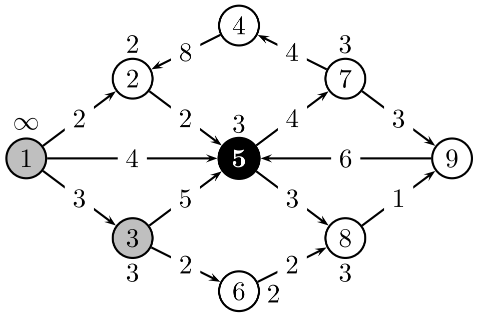
Note that only five Push operations were required to determine the value of $f(9)$. The optimal path is $x = (1,3,5,7,9)$ and its length is equal to $f(9) = 3$. □

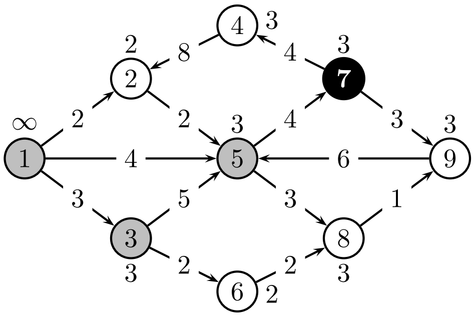
In the next section I give a concise formulation to a method that incorporates Push operations in successive approximation procedures that are


1. Initialization


2. Forward Push at $s = 1$


3. Forward Push at $s = 3$


4. Forward Push at $s = 5$


5. Forward Push at $s = 7$

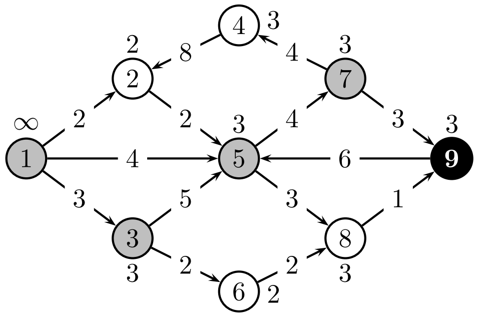

6. Termination

Figure 15.11: Results of the Push procedure

designed for forward dynamic programming functional equations with monotone accumulated return processes.

15.5 Dijkstra's Algorithm

Consider the following generic forward dynamic programming functional equation:

$$f(s) = \underset{(s',x)\in T^{-1}(s)}{\text{opt}} \{f(s') \oplus r(s',x)\} \ , \ s \in S \setminus \{\sigma\} \tag{15.55}$$

with $f(\sigma) = L_{\oplus}$, recalling that σ denotes the initial state of the processes, $L_{\oplus}$ denotes the left identity argument of $\oplus$ and

$$T^{-1}(s) := \{(s',x) : s' \in S, T(s',x) = s\} \ , \ s \in S \tag{15.56}$$

It is convenient to let $Suc(s)$ denote the set of immediate successors of state s, so define

$$Suc(s) := \{T(s,x) : x \in D(s)\} \ , \ s \in S \tag{15.57}$$

In this section I outline the conditions under which the following procedure would be used to solve functional equations of this type.

Algorithm

```
· Initialization:
    · Set $\mathcal{L} = \{\sigma\}$, $\mathcal{P} = \varnothing$ and $F[\sigma] = L_{\oplus}$.
· Iteration:
    · Conduct a FORWARD PUSH OPERATION at the BEST state in $\mathcal{L}$.
    · UPDATE $\mathcal{L}$ and $\mathcal{P}$.
· Termination:
    · Stop if $\mathcal{L}$ is empty, otherwise go to Iteration.
```
□

The *Initialization* and *Termination* steps require no comment. So all I elaborate on is the two steps of the iteration routine, namely the *Push* operation at the best element of $\mathcal{L}$ and the updating of $\mathcal{L}$ and $\mathcal{P}$.

Roughly, $\mathcal{L}$ denotes the set of *active* states and $\mathcal{P}$ denotes the set of *processed* states. Once a state is processed, its $F[\cdot]$ value is not updated again. The next Push operation is conducted at a state in $\mathcal{L}$ whose approximated $f(\cdot)$ value is best over all states in $\mathcal{L}$. The algorithm terminates when the list of active states is empty.

The lists of active and processed states is updated after each Push operation: the state at which the operation was conducted is removed from $\mathcal{L}$ and appended to $\mathcal{P}$. If they are inactive, the hitherto unprocessed immediate successors of the best state become active and are appended to $\mathcal{L}$ (if not already appended).

Since processed states are added to the list $\mathcal{P}$ in each iteration and the total number of states is finite, at some point $\mathcal{P}$ contains all the given states. At this point $\mathcal{L}$ is empty and the process terminates.

The fine details are as follows.

Details:

· PUSH OPERATION.
The BEST state in $\mathcal{L}$ is the element of $\mathcal{L}$ whose $F[\cdot]$ value is best. That is, if opt = min we select the element of $\mathcal{L}$ whose $F[\cdot]$ value is the smallest. if opt = max we select the element of $\mathcal{L}$ whose $F[\cdot]$ value is the largest. Let s^* denote the best state in $\mathcal{L}$, so formally

$$s^* = \arg\operatorname*{opt}_{s\in\mathcal{L}} F[s] \tag{15.58}$$

Note that the choice is not necessarily unique.

Given s^*, the usual forward *Push* operation is conducted at s^*:

$$F[s'] \ \leftarrow \ \begin{cases} F[s^*] \oplus r(s^*,x) & , \ s' \notin \mathcal{L}\cup\mathcal{P} \\ \text{opt } \{F[s'] \ , \ F[s^*]\oplus r(s^*,x)\} & , \ s' \in \mathcal{L} \end{cases} \tag{15.59}$$

for all $x \in D(s^*)$, where $s' = T(s^*,x)$.

· UPDATING $\mathcal{P}$ and $\mathcal{L}$.
This consists of two things, namely
(a) Appending s^* to $\mathcal{P}$:

$$\mathcal{P} \leftarrow \mathcal{P} \cup \{s^*\} \tag{15.60}$$

(b) Appending the hitherto unprocessed successors of s^* to $\mathcal{L}$ and removing s^* from $\mathcal{L}$:

$$\mathcal{L} \leftarrow (\mathcal{L} \cup Suc(s^*)) \setminus \mathcal{P} \qquad \square \tag{15.61}$$

The following observations describe the immediate implications of the manner in which the sets $\mathcal{L}$ and $\mathcal{P}$ are updated and the forward Push operation is conducted.

1. In each iteration a new state is appended to $\mathcal{P}$. Since at the end of each iteration $\mathcal{L}\cap\mathcal{P} = \varnothing$, it follows that the algorithm terminates in no more than $|S|$ iterations.

2. If the accumulated return process induced by $\oplus$ is monotone, then the best states are generated in the order of their $F[\cdot]$ values.

   Specifically, let $s^{(i)}$ denote the best state selected in the i-th iteration. Then,

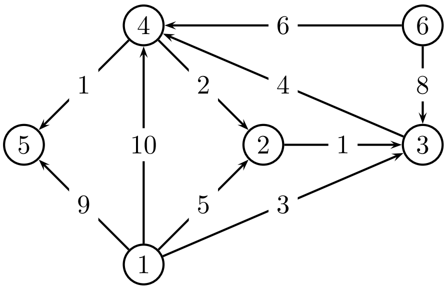
$$\text{opt} = \max : \quad F\left[s^{(1)}\right] \geq F\left[s^{(2)}\right] \geq F\left[s^{(3)}\right] \cdots \tag{15.62}$$

$$\text{opt} = \min : \quad F\left[s^{(1)}\right] \leq F\left[s^{(2)}\right] \leq F\left[s^{(3)}\right] \cdots \tag{15.63}$$

The following example illustrates the algorithm in action in the context of a simple shortest path problem.

15.5.1 Example

Consider the standard shortest path problem depicted in Figure 15.12. Formally we can set $\oplus = +$, $\sigma = 1$ and $S = \{1, 2, 3, 4, 5\}$ and opt = min. Note that $r(i, j)$ denotes the length of arc (i, j).



$r(i,j)$	1	2	3	4	5	6
1	–	5	3	10	9	–
2	–	–	1	–	–	–
3	–	–	–	4	–	–
4	–	2	–	–	1	–
5	–	–	–	–	–	–
6	–	–	8	6	–	–

Figure 15.12: Simple cyclic shortest path problem

Initialization:
We let $\mathcal{L} = \{\sigma\} = \{1\}$, $\mathcal{P} = \varnothing$, and $F[1] = L_+ = 0$.

Iteration #1:
There is only one element in $\mathcal{L}$ so obviously, it is selected as the first best state. And so, we set $s^{(1)} = 1$ and conduct the forward Push operation there. The set of immediate successors of this state is $Suc(1) = \{2, 3, 4, 5\}$, so we update the $F[\cdot]$ values of the states in this set. This yields

$$\begin{aligned}
F[2] &\leftarrow F[1] + r(1,2) = 5 \quad &, (2 \notin \mathcal{L} \cup \mathcal{P})\\
F[3] &\leftarrow F[1] + r(1,3) = 3 \quad &, (3 \notin \mathcal{L} \cup \mathcal{P})\\
F[4] &\leftarrow F[1] + r(1,4) = 10 \quad &, (4 \notin \mathcal{L} \cup \mathcal{P})\\
F[5] &\leftarrow F[1] + r(1,5) = 9 \quad &, (5 \notin \mathcal{L} \cup \mathcal{P})
\end{aligned}$$

So now

$$\begin{aligned}
&\mathcal{P} = \varnothing \cup \{1\} = \{1\}\\
&\mathcal{L} \leftarrow (\{1\} \cup \{2, 3, 4, 5\}) \setminus \{1\} = \{2, 3, 4, 5\}
\end{aligned}$$

Since $\mathcal{L}$ is not empty we conduct the next Push operation.

Iteration #2:
Since opt = min, the best state in $\mathcal{L}$ is 3. So, we set $s^{(2)} = 3$ and conduct the forward Push operation there. The set of immediate successors of this state is $Suc(3) = \{4\}$, so we update the value $F[4]$. This yields

$$F[4] \leftarrow \min\,\{F[4]\ ,\ F[3] + r(3,4)\} = \min\{10, 7\} = 7 \quad , (4 \in \mathcal{L})$$

So now

$$\begin{aligned}&\mathcal{P} \leftarrow \{1\} \cup \{3\} = \{1,3\}\\&\mathcal{L} \leftarrow (\{2,3,4,5\} \cup \{4\}) \setminus \{1,3\} = \{2,4,5\}\end{aligned}$$

Since $\mathcal{L}$ is not empty we conduct the next Push operation.

Iteration #3:
The best state in $\mathcal{L}$ is 2. So, we set $s^{(3)} = 2$ and conduct the forward Push operation there. The set of immediate successors of this state is $Suc(2) = \{3\}$, so, we proceed to update the value $F[3]$. However, since this is a processed state, no update of the value of $F[3]$ is required. Meaning that no Push operation occurs here so that this state is not appended to $\mathcal{L}$.

The updated values of $\mathcal{P}$ and $\mathcal{L}$ are then as follows:

$$\begin{aligned}&\mathcal{P} \leftarrow \{1,3\} \cup \{2\} = \{1,2,3\}\\&\mathcal{L} \leftarrow (\{2,4,5\} \cup \{3\}) \setminus \{1,2,3\} = \{4,5\}\end{aligned}$$

Since $\mathcal{L}$ is not empty we conduct the next Push operation.

Iteration #4:
The best state in $\mathcal{L}$ is 4. So we set $s^{(4)} = 4$ and conduct the forward Push operation there. The set of immediate successors of this state is $Suc(4) = \{2,5\}$, so we proceed to update the values of $F[2]$ and $F[5]$. But as $s = 2$ is a processed state, no update of the value of $F[2]$ is required. So, the Push operation yields

$$F[5] \leftarrow \min\ \{F[5]\ ,\ F[4] + r(4,5)\} = \min\{10, 7+1\} = 8 \quad , \ (5 \in \mathcal{L})$$

The updated values of $\mathcal{P}$ and $\mathcal{L}$ are as follows:

$$\begin{aligned}&\mathcal{P} \leftarrow \{1,2,3\} \cup \{4\} = \{1,2,3,4\}\\&\mathcal{L} \leftarrow (\{4,5\} \cup \{5\}) \setminus \{1,2,3,4\} = \{5\}\end{aligned}$$

Since $\mathcal{L}$ is not empty we conduct the next Push operation.

Iteration #5:
The best state in $\mathcal{L}$ is 5. So we set $s^{(5)} = 5$ and conduct the forward Push operation there. The set of immediate successors of this states is $Suc(5) = \varnothing$, meaning that a Push operation is not required.

The updated values of $\mathcal{P}$ and $\mathcal{L}$ are as follows:

$$\begin{aligned}&\mathcal{P} \leftarrow \{1,2,3,4\} \cup \{5\} = \{1,2,3,4,5\}\\&\mathcal{L} \leftarrow (\{5\} \cup \varnothing) \setminus \{1,2,3,4,5\} = \varnothing\end{aligned}$$

Since $\mathcal{L}$ is empty we stop.

In summary, on termination we have

$\mathcal{P} = \{1,2,3,4,5\}$

s	1	2	3	4	5	6
$F[s]$	0	5	3	7	8	–

Note that $6 \notin \mathcal{P}$, implying that node 6 is not reachable from node 1. Figure 15.13 depicts the sequence of Push operations generated by the algorithm. $\square$

The algorithm outlined above solves the functional equation for all $s \in S$. However, often all we need to know is the value of $f(\hat{s})$ for a given $\hat{s} \in S$. So, to save on computation it may prove worthwhile to alter the termination condition in such cases and to halt the Push operations when the state $\hat{s}$ is deemed best, namely when $\hat{s} = \arg\text{opt}\{F[s] : s \in \mathcal{L}\}$.

And another point to consider. As indicated by the next example, if the accumulated return process induced by $\oplus$ is not monotone with respect to opt then, there is no assurance that the algorithm solves the functional equation for all $s \in S$.

15.5.2 Example

Consider the instance of the standard shortest path problem depicted in Figure 15.14. The object is to find the shortest path from node 1 to node 4, where the length of a path is equal to the sum of the arcs' lengths on the path. Formally we can set opt = min, $\oplus = +$, and $\sigma = 1$. Note that $r(5,4) = -4$, hence the accumulated return generated by $\oplus$ is not monotone with respect to opt.

Next, Dijkstra's Algorithm is applied in a manner where the Push operation terminates when the destination node $s = 4$ is selected for the next Push operation.

Initialization:
We let $\mathcal{L} = \{\sigma\} = \{1\}$, $\mathcal{P} = \varnothing$, and $F[1] = L_+ = 0$.

Iteration #1:
There is only one element in $\mathcal{L}$ hence, it is the first best state. Thus, we set $s^{(1)} = 1$ and conduct the forward Push operation there. The set of immediate successors of this state is $Suc(1) = \{2, 3, 5, 6\}$, so we update the $F(\cdot)$ values of the states in this set. This yields

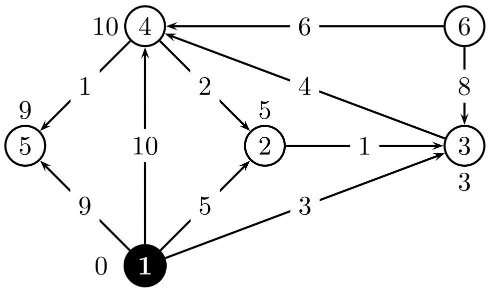
$$\begin{aligned}
F[2] &\leftarrow F[1] + r(1,2) = 5 &&, (2 \notin \mathcal{L} \cup \mathcal{P})\\
F[3] &\leftarrow F[1] + r(1,3) = 3 &&, (3 \notin \mathcal{L} \cup \mathcal{P})\\
F[5] &\leftarrow F[1] + r(1,5) = 10 &&, (5 \notin \mathcal{L} \cup \mathcal{P})\\
F[6] &\leftarrow F[1] + r(1,6) = 9 &&, (6 \notin \mathcal{L} \cup \mathcal{P})
\end{aligned}$$

So now

$$\begin{aligned}
&\mathcal{P} = \varnothing \cup \{1\} = \{1\}\\
&\mathcal{L} \leftarrow (\{1\} \cup \{2,3,5,6\}) \setminus \{1\} = \{2,3,5,6\}
\end{aligned}$$

Since $\mathcal{L}$ is not empty we conduct the next Push operation.

Iteration #2:



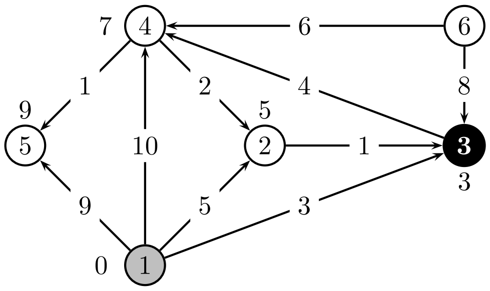
End of the first iteration

s	1	2	3	4	5	6
$F[s]$	0	5	3	10	9	

$\mathcal{P} = \{1\}$

$\mathcal{L} = \{2, 3, 4, 5\}$

$s^{(1)} = 1$



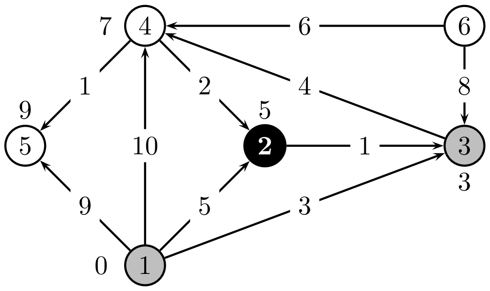
End of the second iteration

s	1	2	3	4	5	6
$F[s]$	0	5	3	7	9	

$\mathcal{P} = \{1, 3\}$

$\mathcal{L} = \{2, 4, 5\}$

$s^{(2)} = 3$



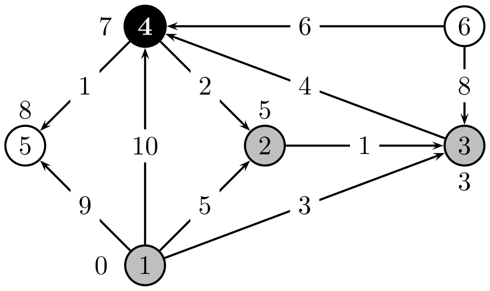
End of the third iteration

s	1	2	3	4	5	6
$F[s]$	0	5	3	7	9	

$\mathcal{P} = \{1, 2, 3\}$

$\mathcal{L} = \{4, 5\}$

$s^{(3)} = 2$



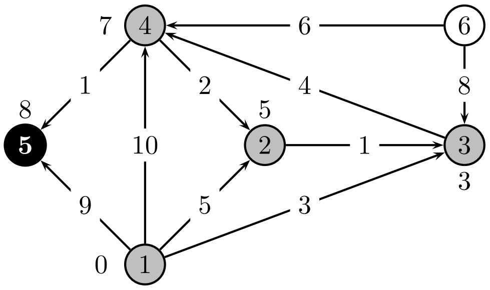
End of the fourth iteration

s	1	2	3	4	5	6
$F[s]$	0	5	3	7	8	

$\mathcal{P} = \{1, 2, 3, 4\}$

$\mathcal{L} = \{5\}$

$s^{(4)} = 4$



End of the fifth iteration

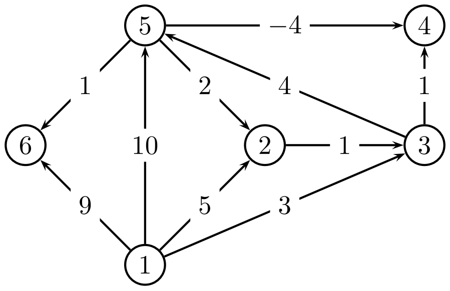
s	1	2	3	4	5	6
$F[s]$	0	5	3	7	8	

$\mathcal{P} = \{1, 2, 3, 4, 5\}$

$\mathcal{L} = \varnothing$

$s^{(5)} = 5$

Figure 15.13: Progression of Push operations



$r(i,j)$	1	2	3	4	5	6
1	–	5	3	–	10	9
2	–	–	1	–	–	–
3	–	–	–	1	4	–
4	–	–	–	–	–	–
5	–	2	–	−4	–	1
6	–	–	–	–	–	–

Figure 15.14: Simple shortest path problem

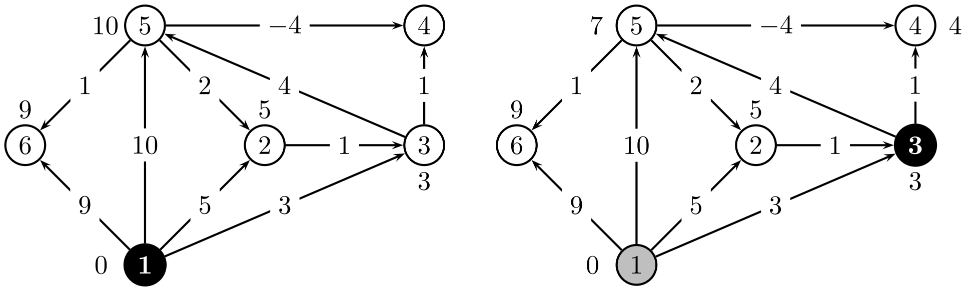


Figure 15.15: Results after the two Push operations (s=1,3)

Since opt = min, the best state in $\mathcal{L}$ is 3. Thus, we set $s^{(2)} = 3$ and conduct the forward Push operation there. The set of immediate successors of this state is $Suc(3) = \{4, 5\}$, so we update the values of $F[4]$ and $F[5]$. This yields

$$\begin{aligned} F[4] &\leftarrow \quad F[3] + r(3,4)\} = 3 + 1 = 4 \quad , \ (4 \notin \mathcal{L} \cup \mathcal{P}) \\ F[5] &\leftarrow \min \ \{F[5] \ , \ F[3] + r(3,5)\} = \min\{10, 7\} = 7 \quad , \ (4 \in \mathcal{L}) \end{aligned}$$

So now

$$\begin{aligned} \mathcal{P} &\leftarrow \{1\} \cup \{3\} = \{1, 3\} \\ \mathcal{L} &\leftarrow (\{2, 4, 5, 6\} \cup \{4, 5\}) \setminus \{1, 3\} = \{2, 4, 5, 6\} \end{aligned}$$

Since $\mathcal{L}$ is not empty we conduct the next Push operation.

Iteration #3:
The best state in $\mathcal{L}$ is the destination node $s = 4$, so we terminate the algorithm.

The results generated by the algorithm are summarized graphically in Figure 15.15.

By inspection, the optimal path from node 1 to node 4 is $(1, 3, 5, 4)$, hence $f(4) = 3 + 4 - 4 = 3$. Clearly then, $f(4) < F[4] = 4$. In short, the algorithm fails to determine the correct value of $f(4)$.

This is hardly surprising because, in this case not all the arcs' lengths are non-negative. Consequently, the accumulated return scheme induced by $\oplus = +$ is not monotone with respect to opt = min. □

Consider then the following theorem. It sets out simple sufficient conditions assuring that Dijkstra's Algorithm will generate a solution to the dynamic programming functional equation under consideration. It is important to note that these conditions do not rule out cyclic state transitions. It therefore applies to cyclic shortest path problems as well.

Theorem 15.5.1 *Consider the functional equation (15.55) and assume that the state space S is finite, namely $|S| < \infty$, and that the accumulated return process induced by $\oplus$ is monotone with* opt. *Then,*

- *The algorithm terminates in a finite number of iterations.*
  Let $\{s^{(i)} : i = 1, 2, \dots, m\}$ denote the sequence of "best" states generated by Dijkstra's Algorithm, namely let $s^{(i)}$ denote the state in $\mathcal{L}$ identified as best in the i-th iteration of the algorithm.
- *The sequence $\{F[s^{(i)}]\}$ is either monotone non-increasing or non-decreasing depending on whether* opt = max *or* opt = min*, respectively.*
- $F\left[s^{(i)}\right] = f\left(s^{(i)}\right) \ , \ \forall i = 1, 2, \dots, m.$
- *If $s \notin \{s^{(i)} : i = 1, 2, \dots, m\}$, then s is not reachable from the initial state $s^{(1)} = \sigma$.* □

Note that in the worst case, a naive implementation of Dijkstra's Algorithm executes $n(n-1)/2$ additions and $n(n-1)$ comparisons, thus its (time) complexity is $O(n^2)$. It should be noted, though, that special data structures can be used to significantly speed up the arg min operation conducted to select the "best" state.

15.6 Summary

The *Push* method offers a procedure that is particularly well-suited for the solution of *forward* dynamic programming functional equations. This is so, because the procedure carrying out the solution of the functional equation is incorporated in the process that generates the states determined by the transition function.

From a dynamic programming perspective this method is a typical *successive approximation* method. Hence, an important aspect of its implementation is the order in which the states are generated.

In this framework, *Dijkstra's Algorithm* is a *Push* type procedure that employs a greedy "best-first" rule for the processing of the states. So when this algorithm is employed to solve large problems, it is essential to use suitable data structures to facilitate an efficient selection of the "next best" state.

15.7 Bibliographic Notes

Details concerning technical and implementation issues pertaining to the *Push* (= *Reaching*) method can be found in Denardo and Fox [1979], Denardo [1982, 2003], Kellerer et al. [2004], Kim et al [2005], Sniedovich [2006], Jensen and Bard [2008].

The *Computer Science* literature on Dijkstra's Algorithm (Dijkstra [1959]) is extensive, and includes standard textbooks (e.g. Cormen et al. [1990]). Interestingly, this literature regards Dijkstra's Algorithm as a Greedy algorithm rather than a dynamic programming algorithm.

The relationship between dynamic programming algorithms and Dijkstra's Algorithm is discussed in Denardo [1982, 2003] and Sniedovich [2006].

Needless to say, a quick search of the WWW will find numerous online animations of Dijkstra's Algorithm (e.g. the online modules at www.ifors.ms.unimelb.edu.au/tutorial/dijkstra/).

Part III

Epilogue

16

What Then Is Dynamic Programming?

In this concluding chapter I round out my discussion of the question "*What is dynamic programming?*" by looking at dynamic programming not from the perspective from which I have investigated it thus far namely as an optimization method but, from a perspective which reveals it to be a basic solution strategy with a far broader scope of operation. To this end I formulate an *abstract dynamic programing model* which enables showing that, in essence, dynamic programming is a profoundly simple, and immensely logical, approach to problem solving. An approach that is applicable not only to optimization problems, but also to problems that are neither phrased nor regarded as optimization problems, or what I call here 'non-optimization' problems.

By concentrating on dynamic programming's treatment of non-optimization problems I hope to bring out more clearly the mechanics of its solution strategy. My thesis is then that examining dynamic programming as a general solution strategy is instructive pedagogically in that it provides deeper insight into its mode of operation; and it is constructive and fruitful methodologically because it furnishes an indication about the directions in which dynamic programming can be extended.

So, my program for this chapter is as follows. I begin with a brief review of dynamic programming's treatment of optimization problems, as delineated in *Chapters 3-4*. This analysis brings out its fundamental strategy, and it serves as a basis for the formulation of an abstract dynamic programming model. I then illustrate how this model is used to handle a number of representative non-optimization problems, and I conclude with some general remarks on dynamic programming.

16.1 Review

Recall that our point of departure in *Chapter 1* was the proposition that dynamic programming's mode of operation is driven by the following Meta-recipe:

· *Embed* your problem in a family of related problems.
· *Derive* a relationship between the solutions to these problems.
· *Solve* this relationship.
· *Recover* a solution to your problem from this relationship.

In *Chapters 3-4* this Meta-recipe was given concrete content when it was put to work in the context a multistage decision model (N, S, D, T, S_1, g) thus bringing to light the relationship between these four objects:

Problem P
Problem P(s)
Problem P(n,s)
Problem P(n,s,x)

So, let us remind ourselves of the formal definitions of these problems, and of the key moves leading to the derivation of the functional equation.

Problem P :

$$p := \operatorname*{opt}_{x\in X} \ g(x) \ , \ x \in X \subseteq X' := X_1 \times \cdots \times X_M \tag{16.1}$$

where g is a real-valued function on X. Let X^* denote the set of optimal solutions to this problem.

Problem P(s),$s \in S_1$:

$$p(s) := \operatorname*{opt}_{(x_1,\dots,x_N)} \ g(s, x_1, x_2, \dots, x_N) \tag{16.2}$$

$$s_1 = s \tag{16.3}$$

$$x_n \in D(n, s_n) \ , \ 1 \leq n \leq N \tag{16.4}$$

$$s_{n+1} = T(n, s_n, x_n) \ , \ 1 \leq n \leq N \tag{16.5}$$

Let $X^*(s)$ denote the set of optimal solutions to *Problem* $P(s)$.

Problem P(n,s),$1 \leq n \leq N, s \in S_n$:

$$f_n(s) := \operatorname*{opt}_{(x_n,\dots,x_N)} \ g_n(s, x_n, x_{n+1}, \dots, x_N) \tag{16.6}$$

$$s_n = s \tag{16.7}$$

$$x_m \in D(m, s_m) \ , \ n \leq m \leq N \tag{16.8}$$

$$s_{m+1} = T(m, s_m, x_m) \ , \ n \leq m \leq N \tag{16.9}$$

Let $X^*(n,s)$ denote the set of optimal solutions to *Problem* $P(n,s)$.

Problem P(n,s,x),$1 \leq n \leq N, s \in S_n, x \in D(n,s)$:

$$f_n(s,x) := \operatorname*{opt}_{(x_{n+1},\dots,x_N)} \ g_n(s, x, x_{n+1}, \dots, x_N) \tag{16.10}$$

$$s_n = s \tag{16.11}$$

$$x_m \in D(m, s_m) \ , \ n+1 \leq m \leq N \tag{16.12}$$

$$s_{m+1} = T(m, s_m, x_m) \ , \ n \leq m \leq N \tag{16.13}$$

Let $X^*(n,s,x)$ denote the set of optimal solutions to *Problem* $P(n,s,x)$.

We refer to *Problem* $P(s)$ as the *initial problem* at s, to *Problem* $P(n,s)$ as the modified problem at (n,s), and to *Problem* $P(n,s,x)$ as the *conditional problem* at (n,s,x).

For simplicity the underlying assumption was that all the initial, modified and conditional problems are well-behaved in that all have optimal solutions.

The derivation of the functional equation was carried out in two steps. First, an appeal to the *Principle of Conditional Optimization*, assured that irrespective of the structure of the decomposition scheme, it is always the case that

$$f_n(s) = \underset{x\in D(n,s)}{\text{opt}} \; f_n(s,x) \ , \ \forall 1 \le n \le N, s \in S_n \tag{16.14}$$

Second, an appeal to the *Markovian Condition* where

$$X^*(n,s,x) = X^*(n+1, T(n,s,x)) \tag{16.15}$$

holds for all $1 \le n < N, s \in S_n, x \in D(n,s)$, assured that

$$f_n(s,x) = \rho(n,s,x,f_{n+1}(T(n,s,x))) \tag{16.16}$$

for all $1 \le n < N, s \in S_n, x \in D(n,s)$.

Thus, substituting the right-hand side of (16.16) for $f_n(s,x)$ in the right-hand side of (16.14), we obtained this functional equation:

$$\boxed{f_n(s) = \underset{x\in D(n,s)}{\text{opt}} \; \rho(n,s,x,f_{n+1}(T(n,s,x))) \ , \ 1 \le n < N, s \in S_n \quad (16.17)}$$

This functional equation, I argued, relates the optimal solutions of *Problem* $P(n,s)$ to the optimal solutions of the modified problems generated by the latter, indicating that:

$$X^*(n,s) = \{(x,z) : x \in D^\circ(n,s), \mathbf{z} \in X^*(n+1,T(n,s,x))\} \tag{16.18}$$

for all $1 \le n < N, s \in S_n$, where

$$D^\circ(n,s) := \{x \in D(n,s) : \rho(n,s,x,f_{n+1}(T(n,s,x))) = f_n(s)\} \tag{16.19}$$

In other words, (16.18) asserts that an optimal solution to *Problem* $P(n,s)$ consists of two components: a decision $x \in D^\circ(n,s)$ and a sequence of decisions $\mathbf{z} = (x_{n+1},\dots,x_N)$, which constitutes an optimal solution to the modified problem at $(n+1,T(n,s,x))$, namely *Problem* $P(n+1,T(n,s,x))$.

In short, what this analysis brought out was that dynamic programming makes the following proposition: embed *Problem* P in a family of parametric problems, namely $\{$*Problem* $P(n,s), 1 \le n < N, s \in S\}$, in such a way that, for each stage-state pair (n,s) the solution to *Problem* $P(n,s)$ would be obtained from the solutions found for the modified problems induced by it, namely from *Problem* $P(n+1,T(n,s,x))$, $x \in D(n,s)$.

Let us now take a closer look at this proposition.

16.2 Non-Optimization Problems

If we abstract the above proposition from the specific context in which it is made, it is immediately clear that it can be read as prescribing a basic approach to problem solving which can be summed up as follows:

> Find a solution to a given problem by creating a family of related problems out of it. Express the relationship between these problems so that a solution to any problem in this family is expressed in terms of the solutions to other members of this family.

Note that absent from this proposition are terms such as *sub-problem* and *decomposition*. This is deliberate to prevent any suggestion of hierarchy between the problems.

The point is then that in this phrasing this proposition applies broadly. Its application is not limited to problems that are expressly classified as "optimization problems". For instance, consider the following elementary problem:

Task A: *Compute the sum of the cubes of the first 100 positive integers.*

Clearly, one should be able to recover a solution to this problem from the solution to the problem:

Task B: *Compute the sum of the cubes of the first 99 positive integers.*

To be able to do this formally, let us define

> **Problem Q**:
> *Compute the sum of the cubes of the first K positive integers.*

Next, define

> **Problem Q**$(k), k = 1, \dots, K$:
> *Compute the sum of the cubes of the first k positive integers.*

To relate these two problems, let z° denote the solution to *Problem Q* and let $z(k)$ denote the solution to *Problem* $P(k), k = 1, 2, 3, \dots, K$. Then, by definition

$$z^\circ := \sum_{m=1}^{K} m^3 \tag{16.20}$$

$$z(k) := \sum_{m=1}^{k} m^3 \ , \ k \in \{1, 2, 3, \dots, K\} \tag{16.21}$$

By inspection then,

$$z^\circ = z(K) \tag{16.22}$$

$$z(k+1) = z(k) + (k+1)^3 \ , \ \forall k = 1, 2, 3, \dots, K \tag{16.23}$$

$$z(1) = 1 \tag{16.24}$$

In brief, one would be able to solve *Problem Q* by means of the following procedure:

```
Program Sum_Cubes(K)
Let z = 0
For k = 1,...,K do:
    z = z + $k^3$
Return z
```

Before I can proceed I ought to make clear my distinction between *optimization* and *non-optimization* problems.

By saying that a problem is a non-optimization problem I clearly do not mean to suggest that it cannot be formulated as an optimization problem.

Obviously, any problem of the form: "*compute the value of* $g(y^\circ)$" where y° is a given element of some set Y and g is a given function of y (not necessarily real-valued), will allow the following formulation as an optimization problem: "$\max\limits_{y\in X} h(g(y))$" where $X := \{y^\circ\}$ and h is any real-valued function on the image of g such that $h(g(y^\circ))$ is well-defined, for instance the case where $h(g(y)) = 0$, for all $y \in X$.

Thus, to phrase *Problem Q* as an optimization problem, we would be able to use the following formulation:

$$z^\circ := \max_{(x_1,\dots,x_K)} \sum_{k=1}^{K} x_k^3 \tag{16.25}$$

$$\text{s.t.} \qquad x_k = k \ , \ k = 1,2,3,\dots,K \tag{16.26}$$

My point is then that by dubbing a given problem a 'non-optimization problem' I mean that the task set by it does not involve an *ostensible* optimization-type operation or relation.

I hasten to add that this is not intended as a formal definition of the term 'non-optimization problem', but as a characterization that seems to be accepted universally. In any event, this characterization is sufficiently clear for the purposes of this discussion so that it need not be pursued any further.

To go back to the procedure described in (16.22)-(16.24).

A cursory examination of this procedure suffices to identify the key tenets of the strategy that, as we have seen, is deployed by dynamic programming. These are:

- The target problem, namely *Problem Q,* is treated as a member of a family of affiliated parametric problems, namely, as a member of the set $\{Problem\ Q(k) : k \in \{1, 2, 3, ..., K\}\}$. Specifically, *Problem Q* is identical to *Problem Q*(K).
- The solution to the target problem is expressed in terms of the solutions of (a) certain parametric problem(s), namely, the solution to *Problem Q* is identical to the solution to *Problem Q*(K).

· The solutions to the parametric problems themselves are related to one another, namely, $z(k+1) = z(k) + (k+1)^3$.
· The relation between the problems operates in the domain of the parameter of the parametric problems, namely in $\{1, 2, 3, \ldots, K\}$.

Having just seen that dynamic programming's central proposition clearly applies outside the realm of optimization, my next task is to identify a mathematical model that makes this possible. That is, the task is to formulate a model that will allow explaining how and why this proposition applies to problems generally, and not only to optimization problems.

16.3 An Abstract Dynamic Programming Model

Whereas the model associated thus far with dynamic programming's mode of operation is based on the premise that it is aimed specifically for the treatment of *optimization problems*, no such assumption is made regarding the model that I formulate below. No component of this model is designed to meet the requirements of properties that are typical of optimization problems. Indeed, every effort is made to frame the model's constituent elements so as to render it capable of handling problems *as such*, that is, problems that are not a priori affiliated with any specific realm of application, for instance optimization.

The point to note then is that the model is considered 'abstract' because it is not *biased* to any particular domain of application.

The model consists of two components:

· A problem transition model.
· A functional equation.

The problem transition model is an analog of the familiar state transition model except that instead of *states* it consists of *problems* and ... it does not involve decisions — only transitions.

In the same vein, the functional equation does not involve the opt operation.

Now to the details.

Definition 16.3.1 *A* PROBLEM TRANSITION MODEL *is a pair* $(\mathcal{P}, \mathcal{T})$ *where* $\mathcal{P}$ *is a set called the* PROBLEM SPACE *whose elements are called* MODIFIED PROBLEMS*, and* $\mathcal{T}$ *is a map from* $\mathcal{P}$ *to the power set of* $\mathcal{P}$*, called the* TRANSITION FUNCTION.

The elements of set $\mathcal{T}(p)$ *are called the* SUPPORTERS *of* p*. Let* $\mathcal{P}''$ *denote the set of all* $p \in \mathcal{P}$ *such that* $\mathcal{T}(p) = \varnothing$*. We shall refer to such problems as* UNSUPPORTED MODIFIED PROBLEMS*. Let* $\mathcal{P}' := \mathcal{P} \backslash \mathcal{P}''$*.*

We assume that for each $p \in \mathcal{P}$ there is a set $Z(p)$ called the SOLUTION *set of p. We shall refer to the elements of $Z(p)$ as* SOLUTIONS *of p.*

So when I say that dynamic programming's solution strategy is applicable generally I mean that this strategy prescribes to solve a problem by formulating for it a suitable problem transition model and a functional equation that will yield a solution to this problem.

Hence:

Definition 16.3.2 *A* DYNAMIC PROGRAMMING MODEL *is a collection $(\mathcal{P}, \mathcal{T}, \pi, \mathcal{L})$ such that $(\mathcal{P}, \mathcal{T})$ is a problem transition model, π is an element of $\mathcal{P}$ and $\mathcal{L}$ is a map such that*

$$Z(p) = \mathfrak{L}\left(\{(p', Z(p')) : p' \in \mathcal{T}(p)\}\right) \ , \ \forall p \in \mathcal{P}' \tag{16.27}$$

In particular,

$$Z(\pi) = \mathfrak{L}\left(\{(p', Z(p')) : p' \in \mathcal{T}(\pi)\}\right) \tag{16.28}$$

We refer to π as the TARGET PROBLEM *and to $\mathcal{L}$ as the* LINKER.

Thus, given a problem, call it *Problem Q*, the idea is to:

· Construct a problem transition model $(\mathcal{P}, \mathcal{T})$.
· Identify a problem $\pi \in \mathcal{P}$ that is equivalent to *Problem Q*.
· Formulate a linker $\mathcal{L}$ so that $(\mathcal{P}, \mathcal{T}, \pi, \mathcal{L})$ is a dynamic programming model.
· Solve the functional equation for $p = \pi$ to obtain the solution set for *Problem Q*.

Of course, not all the model's components always have a manifest role in any given problem situation. Still each object represents a necessary feature of this solution strategy so, it is considered a constituent part of the model.

Let us now take a closer look at the model.

· PROBLEM SPACE, $\mathcal{P}$.
The object $p \in \mathcal{P}$ is a parameter whose role is to serve as a connecting link between the problem sought to be solved — the *target problem* — and the related problems created out of it which are referred to as the modified problems. The target problem itself is equivalent to one of these problems, which is denoted π.
· TRANSITION FUNCTION, $\mathcal{T}$.
This object acts as a relator. It relates the modified problems to one another. For each $p \in \mathcal{P}$ it identifies the problems whose solutions can be used in the construction of the solution to p. The elements of $\mathcal{T}(p)$ are therefore called *supporters of p*.

· Linker, $\mathcal{L}$.
This is a map linking the solution sets of the modified problems via a *functional equation*. That is, the elements of the model are formulated in such a way that the functional equation specified by (16.27) holds. Since this equation applies only to $p \in \mathcal{P}'$, the implication is that solutions to $p \in \mathcal{P}''$ are either readily available or can be constructed — somehow — independently of the the dynamic programming schemes.

I now proceed to elucidate the role that each element of the model has in this solution strategy, and to explain how they are related to one another. I do this in the context of the following simple, indeed trivial, non-optimization problem.

Problem:
A pair of newborn Australian rabbits — male and female — are swept away to a desert island. What will the rabbit population of the island be a year later?

This problem will be referred to henceforth as the *A-Rabbits Problem*, and the landing of the first pair of rabbits on the island as the *Beginning.*

16.3.1 Target Problem

Because our model is intended to have the broadest range of applications possible, *Problem Q* is intentionally unspecified. It must be emphasized, however, that this in no way gives license to a slack problem definition *once the model is applied* to a given problem situation. A precise formulation of the problem considered and a clear definition of what constitutes a solution to this problem are imperative.

In our case, an accurate definition of the problem means taking into account the reproduction details of the "Australian rabbit", which are as follows:

Each month, a doe has a litter of two, male and female. A newborn female rabbit is fertile after two months and in turn has a litter of two, again male and female. The process is assumed to continue in this manner indefinitely.

So formally the target problem would be phrased as follows:

Problem Q:
Determine the Australian rabbit population on the island one year after the Beginning.

The solution to this problem, z°, must then be a positive integer stipulating the total number of rabbits on the island 12 months after the *Beginning.*

16.3.2 Problem Space

The problem variable, $p \in \mathcal{P}$, is a parameter by means of which the target problem is linked to the set of problems derived from it — the modified problems; and by means of which one establishes links among the modified problems themselves and among their respective solution sets.

Recalling the analysis in *Chapter 11,* to identify the *Problem Space* in the context of a given problem, we need to determine the *nature* of the relationship between the target problem and the modified problems.

Because this relationship is governed by the fact that the role of the modified problems is to serve as a means to recover the solution to the target problem, the nature of this relationship is established on grounds of the task set by the target problem.

Translating the above comments to our case entails the following. The objective of our target problem is to determine the rabbit population of the island 12 months after the *Beginning.* To compute this figure we would seek to determine the population profile 11 months after the *Beginning* — population profile meaning: number of adult rabbits (two months old and older), number of young rabbits (one month old), and number of newborn rabbits. But, to compute this figure, we would need the population profile 10 months after the *Beginning*, and so on.

Clearly then, given this relationship between the target problem and its derivative (modified) problems, a modified problem would in this case be a quadruplet: one component will stipulate time (months after the *Beginning*) and each of the remaining three will stipulate the size of an age group in that month. For example, $p = (1, 2, 0, 0)$ indicates a situation where one month after the *Beginning* the population profile consists of two adults, zero young and zero newborn rabbits.

There are, however, other possible formulations for the problem. For instance, it is possible to capitalize on the following two facts regarding the "Australian rabbit" population on the island:

> The number of newborn rabbits n months after the Beginning is equal to the entire rabbit population $n - 2$ months after the Beginning,

and

> The total number of young and adult rabbits n months after the Beginning is equal to the entire rabbit population $n - 1$ months after the Beginning.

It therefore follows that

> The total number of rabbits on the island n months ($n \geq 2$) after the Beginning is equal to the sum of the rabbit populations on the island in the previous two months.

Hence, the solution to the *A-Rabbit Problem* would be computed from the solutions to the above related problems, which are instances of the following generic problem:

> Determine the total rabbit population n months after the Beginning, $n = 0, 1, 2, \ldots, 12$.

The inference is then that we can use the set $\mathcal{P} := \{0, 1, 2, 3 \ldots, 12\}$ as a problem space. In this case, a state $p \in \mathcal{P}$ is a nonnegative integer stipulating time (number of months after the *Beginning*).

16.3.3 Modified Problems

The main features of the modified problems have already been noted in the discussion of the *Problem Space*. So, all that remains to be pointed out, is that the modified problems are formulated in a manner ensuring that one would be able to construct the solution set of the target problem from the solution sets of those modified problems that are its *supporters*, namely $\mathcal{T}(\pi)$.

In the case of the *A-Rabbits Problem* the modified problems would be defined as follows:

> **Problem Q**$(p), p \in \mathcal{P}$:
> *Determine the rabbit population on the island p months after the Beginning.*

Thus, the solution to *Problem Q(p)*, namely $z(p)$, is a positive integer stipulating the total number of rabbits on the island p months after the *Beginning.*

16.3.4 Unsupported Problems

These are modified problems that are characterized by solution sets that, in some sense, are *immediate.* Note, however, that our abstract model allows situations where no unsupported problems exist, namely where $\mathcal{P}''$ is empty.

That is, it is *not* a requirement of the model that a solution for at least one of the modified problems be easily at hand. By analogy, *successive approximation* methods are deployed in situation where $\mathcal{P}''$ is empty.

By providing for unsupported problems, the model is descriptive rather than prescriptive. It simply reflects the fact that, cases involving modified problems with immediate solution sets do exist, for instance, cases where $\mathcal{P}$ is finite and $\mathcal{T}$ is acyclic. Indeed, in such cases the iterative solution procedure is *initiated* at the unsupported problems.

In the case of the *A-Rabbits Problem*, the modified problems pertaining to $p = 0$ and $p = 1$ are trivial, namely it is easily determined, by "inspection", that $z(0) = z(1) = 2$. We can therefore set $\mathcal{T}(0) = \mathcal{T}(1) = \varnothing$ implying that $\mathcal{P}'' = \{0, 1\}$ and $\mathcal{P}' = \mathcal{P} \backslash \mathcal{P}'' = \{2, 3, \ldots, 12\}$.

16.3.5 Transition Function

Consider a typical modified problem, $p \in \mathcal{P}'$. In parallel to its role with regard to optimization problems, the transition function in this model also links p to those problems in $\mathcal{P}$ whose solution sets figure in the construction of the solution set of *Problem* $Q(p)$. More accurately, for each $p \in \mathcal{P}''$ the transition function $\mathcal{T}$ identifies those modified problems whose solution sets will be explicit terms in the expression denoting the solution set $z(p)$. For this reason, the elements of $\mathcal{T}(p)$, are termed *supporters* of p.

Situations exist where the sets $\{\mathcal{T}(p) : p \in \mathcal{P}'\}$ are typically *singletons*. In such cases it is more convenient to treat $\mathcal{T}(p)$ as an *element* of $\mathcal{P}$ rather than a subset thereof. Thus, although formally our model requires $\mathcal{T}(p)$ to be a subset of $\mathcal{P}$, whenever it is a singleton it is treated as an element of $\mathcal{P}$ rather than a subset thereof.

In the case of the *A-Rabbits Problem*, we saw that for $p \in \mathcal{P}' = \{2, 3, \ldots, 12\}$ one can readily obtain $z(p)$ from $z(p-1)$ and $z(p-2)$. We can therefore set in this case $\mathcal{T}(p) := \{(p-1), (p-2)\}, p \in \mathcal{P}'$.

16.3.6 Linker

This object connects the solution sets of the modified problems to one another, the result being a *functional equation*. Formally, $\mathcal{L}$ is a map satisfying (16.27), or equivalently

$$z(p) = \mathcal{L}\left(p, \bigcup_{p' \in \mathcal{T}(p)} \{(p', z(p'))\}\right) \ , \ p \in \mathcal{P}' \tag{16.29}$$

This relation asserts the following: for each supported problem $p \in \mathcal{P}'$, the solution set of *Problem* $Q(p)$, namely $z(p)$, can be expressed in terms of the solution sets pertaining to the supporters of p, namely $\{z(p') \colon p' \in \mathcal{T}(p)\}$.

Note that in (16.27) and (16.29) p' is appended to $z(p')$ so as to enable $\mathcal{L}$ to associate the solution sets with their respective supporters. This is called for because $z(p')$ does not always uniquely identify p'. In cases where this in not called for, the functional equation has a simpler form:

$$z(p) = L\left(p, \bigcup_{p' \in \mathcal{T}(p)} z(p')\right) \ , \ p \in \mathcal{P}' \tag{16.30}$$

For example, in the case of the *A-Rabbits Problem*, we can define $\mathcal{L}$ as follows:

$$\mathcal{L}\left(p, \bigcup_{p' \in \mathcal{T}(p)} z(p')\right) := \sum_{p' \in \mathcal{T}(p)} z(p') \quad , \ p \in \{2, 3, \ldots, 12\} \tag{16.31}$$

Thus, the functional equation takes this form:

$$z(p) = \sum_{p' \in \mathcal{T}(p)} z(p') \ , \ p \in \{2, 3, \dots, 12\} \tag{16.32}$$

$$= z(p-1) + z(p-2) \ , \ p \in \{2, 3, \dots, 12\} \tag{16.33}$$

Solving this equation for $p = 2, 3, \dots, 12$ — in this order — we obtain the following sequence of solutions for the modified problems:

p	0	1	2	3	4	5	6	7	8	9	10	11	12
$z(p)$	2	2	4	6	10	16	26	42	68	110	178	288	466

Thus, $z^\circ = z(12) = 466$.

As the reader may no doubt have noticed, the functional equation (16.33) constitutes a simple application of the famous *Fibonacci sequence*

$$a_n := a_{n-1} + a_{n-2} \ \ , \ n > 1 \tag{16.34}$$

with $a_0 := a_1 := 1$.

I call attention to the fact that, although our abstract model does not expressly apply to decision problems, it can easily be adapted to allows the underlying strategy to handle problems involving DECISION VARIABLES. All one needs to do to this end is to incorporate in the model a set, $\mathbb{D}$, that would be defined the DECISION SPACE, and a collection of sets $\{D(p) : p \in \mathcal{P}\}$ such that $D(p) \subseteq \mathbb{D}$, $\forall p \in \mathcal{P}$, where $D(p)$ would be conceived of as the *set of feasible decisions pertaining to the problem p*.

In this case the transition function will typically accept two arguments, namely $p \in \mathcal{P}$ and $x \in D(p)$, so that the functional equation will take the form

$$z(p) = \mathcal{L}\left(p, \bigcup_{\substack{x \in D(p) \\ p' \in \mathcal{T}(p,x)}} \{(x, p', z(p'))\} \right) \ , \ \forall p \in \mathcal{P}' \tag{16.35}$$

if $\mathcal{T}(p, x)$ is typically a *subset* of $\mathcal{P}$, or

$$z(p) = \mathcal{L}\left(p, \bigcup_{x \in D(p)} \{(x, \mathcal{T}(p,x), z(\mathcal{T}(p,x)))\} \right) \ , \ \forall p \in \mathcal{P}' \tag{16.36}$$

if $\mathcal{T}(p, x)$ is typically an *element* of $\mathcal{P}$.

The immediate implication is then that all the dynamic programming models discussed thus far in this book are in fact special cases of the abstract model outlined above.

As for the solution of the functional equations emanating from this model, the discussion on *solution methods* (*Chapter 5-6*) applies here as well. Roughly, there would be two ways in which to approach the equation.

This would be determined by whether or not the transition function generates *cycles*. In acyclic cases where $\mathcal{P}$ is finite, the functional equation would typically be solved by *direct methods*. Otherwise, it would be solved with *successive approximation methods* or *truncation methods.*

Having formulated the model that, I submit, provides an account of the *essence* of dynamic programming's approach to problem solving, I shall now proceed to examine it in action.

16.4 Examples

In this section I illustrate the working of the solution strategy, encapsulated in the above abstract model, by studying its handling of a number of problems — all non-optimization problems — from vastly different areas. I have purposely chosen extremely simple problems to be able make in, as effective a fashion as possible, the following points.

First, I want to reinforce my main thesis that dynamic programming's strategy is profoundly simple, but at the same time extremely sensible.

Second, by demonstrating that dynamic programming's mode of operation may well be equated with "common sense", I want to highlight its *rudimentary nature.*

Third, by bringing these qualities to the fore, I want to underscore that it is precisely by dint of these qualities that dynamic programming's approach is so widely applicable.

Finally, I want to point out that by virtue of these characteristics, the dynamic programming strategy is in fact being used, *routinely*, by *Homo Sapiens* with no training in dynamic programming ...

And before turning to the examples, I quickly run through the basic ingredients of our abstract dynamic programming model:

· Problem space, $\mathcal{P}$.
· A target problem, $\pi \in \mathcal{P}$.
· Transition function, $\mathcal{T}$.
· Set of unsupported problems, P''.
· Linker, $\mathcal{L}$.

I illustrate, in each example, how these constructs are formulated and how the resulting functional equation is obtained. It is important to keep in mind though, that there are generally several ways of formulating a problem with these constructs.

To highlight the *embedding process* associated with the deployment dynamic programming, I shall refer to the target problem in two different ways: as *Problem* Q and as *Problem* $Q(\pi)$. The representations of these two problems can be different, but both pertain to the same problem. Also, I shall refer to the modified problem $p \in \mathcal{P}$ as *Problem* $Q(p)$.

The first example is taken from the area of *computer programming.* It concerns what is known in this discipline as a *counter.* This concept is of fundamental importance in Von Neumann type computer programming languages such as Algol, Basic, C, Cobol, Fortran, Java, Pascal, PL/1, etc.

In view of the highly *sequential approach* to problem solving of these languages, a problem (computing task) is typically broken down into smaller sub-problems, so that counters are used to construct the solution to the target problem from the solutions to the smaller sub-problems. As we shall soon see, in terms of our model, these counters are the solutions to the modified problems.

16.4.1 Example

Consider the computer programming problem specified by the following input/output requirement:

Input: A finite sequence of real numbers, $(y_1, y_2, \dots, y_k)$.
Output: The scalar

$$z^\circ := \max \{y_n^2 : 1 \leq n \leq k\} - \min \{y_n^2 : 1 \leq n \leq k\} \tag{16.37}$$

A Von Neumann type language would typically decompose this problem into two subproblems, namely:

$$\textit{Problem } Q_{\max} : \text{ Compute the value of } z^\circ_{\max} = \max \{y_n^2 : 1 \leq n \leq k\}$$

and

$$\textit{Problem } Q_{\min} : \text{ Compute the value of } z^\circ_{\min} = \min \{y_n^2 : 1 \leq n \leq k\}$$

It would then obtain the value of z° as follows: $z^\circ = z^\circ_{\max} - z^\circ_{\min}$. The main thrust would be then to solve *Problem* $Q_{\max}$ and *Problem* $Q_{\min}$. So, let us examine how such problems are normally solved by focusing on the former, observing that the treatment of the latter will essentially be identical.

The solution strategy is simple. Capitalizing on the fact that the max operation is *associative,* define:

$$z_n := \max_{1 \leq m \leq n} \{y_m^2\} \tag{16.38}$$

from which it immediately follows that

$$z_{n+1} = \max \{z_n, y_{n+1}^2\}, 1 \leq n < k \tag{16.39}$$

and

$$z_k = z^\circ_{\max} \tag{16.40}$$

Thus, implementing (16.39) results in a procedure along these lines:

```
Program max_squares $(k, y, z)$
Dimension y(k)
Let $z = 0$
For $n = 1$ to $k$ do:
  Begin
    $z = \max(z, y_n^2)$
  End
Return z
```

The central operation of this procedure, referred to in computer science as a "loop", of course, boils down to dynamic programming's solution strategy.

The target problem, namely *Problem* $Q_{\max}$, is solved by means of a family of modified problems defined by (16.38). Formally, we can set $\mathcal{P} := \{1, 2, \dots, k\}$ and define:

> **Problem Q**(p), $p \in \mathcal{P}$:
> Compute the value of $z(p) := \max\left\{y_n^2 : 1 \leq n \leq p\right\}$

Then, clearly the target problem is identical to *Problem* $Q(\pi), \pi := k$. Also, observe that $z(1)$ is clearly equal to y_1^2. Thus, we can set $\mathcal{P}'' = \{1\}$ and $\mathcal{P}' = \{2, 3, \dots, k\}$. Furthermore, by inspection,

$$z(p) = \max\{z(p-1), y_p^2\} \tag{16.41}$$

so that the transition function t would be defined thus

$$\mathcal{T}(p) := p - 1 \ , \ p \in \mathcal{P}' \tag{16.42}$$

and the linker $\mathcal{L}$ as follows:

$$\mathcal{L}(p, z(\mathcal{T}(p))) := z(\mathcal{T}(p)) + y_p^2 \ , \ p \in \mathcal{P}' \tag{16.43}$$

with

$$\mathcal{L}(1, \xi) := y_1^2 \ , \ \text{for any value of } \xi. \ \square \tag{16.44}$$

Our second example is an illustration of the *convolution* operation. The discussion assumes that the reader has some knowledge of *conditional probabilities*.

16.4.2 Example

Let $(\widetilde{v}_n : 1 \leq n \leq k)$ be a sequence of stochastically independent and identically distributed discrete random variables, and let f denote their common probability mass function. Also, let v_n denote the value that $\widetilde{v}_n$ takes. Now, consider the following task:

Problem Q:

Compute the probability function of the random variable

$$\widetilde{V} = \sum_{n=1}^{k} \widetilde{v}_n \tag{16.45}$$

The solution in this case is a real-valued function F on $\mathbb{R}$ such that $F(V)$ is equal to the probability that $\widetilde{V}$ is equal to V.

Now, in line with dynamic programming's strategy, we would argue as follows. If we define,

$$\widetilde{V}_m := \sum_{n=1}^{m} \widetilde{v}_n \ , \ m = 1, 2, 3, \ldots \tag{16.46}$$

then clearly,

$$\widetilde{V}_k = \widetilde{V} \tag{16.47}$$

and

$$\widetilde{V}_{m+1} = \widetilde{V}_m + \widetilde{v}_m \ , \ \forall m = 1, 2, \ldots, k \tag{16.48}$$

So, in view of (16.44)-(16.45), we regard *Problem Q* as an instance of the parametric problem

Problem Q(p), p=1,2,. . . ,k:

Compute the probability mass function of $\widetilde{V}_p$

namely, *Problem Q* is identical to *Problem Q*(k).

And so, let F_p denote the probability mass function of $\widetilde{V}_p$. The fact that *Problem Q* is identical to *Problem Q*(k) implies that F is identical to F_k. Now, capitalizing on the basic properties of *conditional probabilities*, we immediately deduce from (16.48) that

$$\text{Prob}\left[\widetilde{V}_{p+1} = V \mid \widetilde{v}_p = v\right] = \text{Prob}\left[\widetilde{V}_p = V - v\right] \ , \ V, v \in \mathbb{R} \tag{16.49}$$

which in turn implies that

$$\text{Prob}\left[\widetilde{V}_{p+1} = V \mid \widetilde{v}_p = v\right] = \sum_{v \in W} \text{Prob}\left[\widetilde{V}_p = V - v\right] \text{Prob}\left[\widetilde{v}_p = v\right] \tag{16.50}$$

where W denotes the values of v such that $f(v) > 0$. The conclusion is therefore that

$$P_{p+1}(V) = \sum_{v \in W} f(v) \times P_p(V - v) \ , \ \forall 1 \leq p < k, V \in \mathbb{R} \tag{16.51}$$

noting that by definition $F_1 = f$.

This, of course, is a typical dynamic programming functional equation. Observe that the solutions to the modified problems, and to the target problem, are *functions* in this case. Couching the problem under consideration in terms of the model that we outlined above, we have:

$$\mathcal{P} = \{1, 2, \ldots, k\} \tag{16.52}$$

$$\pi = k \tag{16.53}$$

$$\mathcal{T}(p) = p - 1 \ , \ p \in \mathcal{P}' = \{2, 3, \ldots, k\} \tag{16.54}$$

$$\mathcal{P}'' = \{1\} \tag{16.55}$$

$$\mathcal{L}(p, z(\mathcal{T}(p))) = f * z(\mathcal{T}(p)) \ , \ p \in \mathcal{P}' \tag{16.56}$$

where $f * g$ denotes the convolution of f and g, namely it is a real-valued function on $\mathbb{R}$ such that

$$(f * g)(V) = \sum_{v \in W} p(v) g(V - v) \ , \ V \in \mathbb{R} \tag{16.57}$$

The functional equation would thus have this form:

$$z(p) = f * z(p - 1) \ , \ p \in \{2, 3, 4, \ldots, k\} \tag{16.58}$$

observing that by definition $z(1) = f$.

It is interesting to note that the rules of conditional probabilities govern the above analysis in a manner akin to the way the *Principle of Conditional Optimization* governs the derivation of the functional equation of an optimization problem. □

It has long been recognized that dynamic programming's solution strategy is particularly well-suited for optimization problems in the areas of graph theory and networks. What is not as clearly recognized is that this strategy is equally well-suited for non-optimization problems in these areas. Indeed, it is not recognized that many of the strategies routinely used to solve these problems are in actual fact dynamic programing strategies. The following is a case in point.

16.4.3 Example

Recall that the *transitive closure* of a graph is a square boolean matrix whose (i, j)-th entry is equal to 1 if, and only if, there is a path from node i to node j. It will be convenient to use the following terminology: node j is said to be *a successor of node* i if, and only if, there exists a path from node i to node j. If the graph has an arc (i, j), then node j is said to be an *immediate successor*of node i. Suppose then that our object of interest is a square boolean matrix $\mathcal{M}$ such that $\mathcal{M}[i, j]$ is equal to 1 if, and only if, node j is an immediate successor of node i. We refer to $\mathcal{M}$ as the *precedence matrix.* The following is a fundamental problem in graph theory:

Task:
Compute the transitive closure of a graph from its precedence matrix, $\mathcal{M}$.

It goes without saying that, one would be able to restate this problem as an optimization problem and then use the so to speak, conventional dynamic programming approach to solve it. The point I am making, though, is that in its present formulation *the problem is directly amenable to dynamic programming's solution strategy, without having to undergo a formulation involving optimization.* To see this, note that the following observation is trivially true:

Node j is a successor of node i if, and only if,

either

node j is an immediate successor of node i,

and/or

node i has an immediate successor k such that node j is a successor of node k

The solution strategy suggested by this observation is as follows: let m denote the number of nodes comprising the graph, and define

Problem Q(i,j), $i,j = 1,2,\dots,m$:
Is node j a successor of node i ?

Then, the solutions to this family of problems are specified by the matrix stipulating the transitive closure of the graph, say C. The immediate implication of the above observation is that C must have the following property:

$$C[i,j] = \mathcal{M}[i,j] \vee (\mathcal{M}[i,k] \wedge C[k,j] \text{ for some } k = 1,2,3,\dots,m \quad (16.59)$$

where $\wedge$ and $\vee$ denote the boolean *AND* and (inclusive) *OR* operations, respectively.

This is, of course, a typical recursive relation characteristic of the strategy deployed by dynamic programming. In matrix notation it can be restated as follows:

$$C = \mathcal{M} \vee (\mathcal{M} \vee . \wedge C) \qquad (16.60)$$

where the composite operation $\vee.\wedge$ is defined as follows: let A and B be square boolean matrices of the same dimension. Then, $B \vee . \wedge A$ yields a square boolean matrix whose $(i,j)th$ entry is equal to 1 if, and only if, there is at least one k such that both $B[i,k]$ and $A[k,j]$ are equal to 1.

Now, as this strategy is intended to apply to cyclic graphs as well, provisions must be made for situations where it would be impossible to solve (16.60) for C in a direct recursive manner. Suffice it to say then that in such cases one would employ the successive approximation method, which offers the following scheme: Set,

$$C^{(1)} := \mathcal{M} \tag{16.61}$$

and then compute

$$C^{(n+1)} := C^{(n)} \vee (\mathcal{M} \vee . \wedge C^{(n)}), n = 1, 2, 3, \ldots \tag{16.62}$$

The procedure terminates when $C^{(n+1)}$ is equal to $C^{(n)}$. Note that this will occur in no more than m iterations, where m is equal to the dimension of $\mathfrak{M}$. Furthermore, it is clear that the equation has a unique solution. Thus, if the procedure terminates at the n-th iteration it follows that $C = C^{(n)}$. To accelerate the convergence, one can use the following scheme instead. Set

$$C^{(1)} := \mathcal{M} \tag{16.63}$$

and compute

$$C^{(n+1)} := C^{(n)} \vee (C^{(n)} \vee . \wedge C^{(n)}), n = 1, 2, ... \tag{16.64}$$

until $C^{(n+1)}$ is equal to $C^{(n)}$.

The following analysis illustrates how the transitive closure figures in a typical non-optimization graph theory problem. The target problem is as follows:

Target Problem:
Can one fly from city A to city B with a stopover in city C?

The implicit assumption here is that, a direct flights table, call it DFT, is available for the region. This is a boolean matrix whose (i, j)th entry is equal to 1 if, and only if, there is a *direct flight* from city i to city j.

Now, it is immediately clear that the problem under consideration is related to the following problems:

(a) *Can one fly from city A to city C?*
(b) *Can one fly from city C to city B?*

so that its solution would be constructed from the solutions to these two problems. That is, the answer to the target problem would be 'yes' if, and only if, the answer to both these questions is 'yes'.

As for the strategy to obtain this solution, it is obvious that the connecting link between the two (modified) problems as well as between them and the target problem is a pair of cities. That is, that they are instances of the following *generic* problem:

(c) *Can one fly from city i to city j?*

Thus, the *Problem Space*, $\mathcal{P}$, is the collection of all ordered pairs of distinct cities (the matrix DFT is not assumed to be symmetric). So, supposing that there are N cities, and that they are denoted by $1, 2, \ldots, N$, then $\mathcal{P}$ is the collection of all pairs (i, j) where i and j are distinct elements of $\{1, 2, \ldots, N\}$, namely $\mathcal{P} := \{(i, j) : i, j \in \{1, 2, 3, \ldots, N\}, i \neq j\}$.

Hence, the modified problems have this form:

Problem Q(p), $p = (i, j) \in \mathcal{P}$:
Can one fly from city i to city j?

In this framework to solve the target problem we need the solutions to two problems, namely $p = (A, C)$ and $p = (C, B)$ and furthermore the target problem is not an element of $\mathcal{P}$. This technicality will be taken up shortly. For the time being it is convenient to keep $\mathcal{P}$ in its current form.

As for the transition function $\mathcal{T}$, this function can take a number of forms depending on the definition of *Linker*, $\mathcal{L}$. The question is how to link the modified problems in this case. The answer is as follows: Let $p = (i, j)$ be a pair of cities such that $DFT[i, j] = 1$. Then, clearly $z(i, j) = yes$ because there is a direct flight from i to j. This means that all the problems $p = (i, j)$ such that $DFT[i, j] = 1$ are easily solved. Hence, we set

$$\mathcal{P}'' : = \{(i, j) : DFT[i, j] = 1, i, j = 1, 2, \ldots, N\} \tag{16.65}$$

$$\mathcal{T}(i, j) : = \varnothing \ , \ (i, j) \in \mathcal{P}'' \tag{16.66}$$

Given that the decisive factor in the solution of *Problem* $Q(i, j)$ is the solution of *Problem* $Q(k, j)$ for all cities k which can be reached from city i in a direct flight, it follows that the transition function can be defined thus:

$$\mathcal{T}(i, j) := \{(k, j) : DFT[i, k] = 1, k \in \{1, 2, \ldots, N\} \tag{16.67}$$

for $k \notin \{i, j\}\}, (i, j) \notin \mathcal{P}''$, observing that $\mathcal{T}$ is independent of its second argument.

Next, recall that the solution to *Problem* $Q(i, j)$ is equal to "yes" if, and only if, there exists a city k such that the solutions to both *Problem* $P(i, k)$ and *Problem* $Q(k, j)$ are equal to "yes"; otherwise the answer is equal to "no". *Linker* is therefore defined as follows:

$$\mathcal{L}\left(p, \bigcup_{p' \in \mathcal{T}(p)} \{(p', z(p'))\}\right) := \begin{cases} yes & , \ z(k, j) = yes \text{ for some} \\ & \qquad (k, j) \in \mathcal{T}(i, j) \\ no & , \ otherwise \end{cases} \tag{16.68}$$

where $p = (i, j)$.

Thus, if we replace *yes* and *no* by 1 and 0, respectively, the functional equation takes the following concise form:

$$z = DFT \vee (DFT \vee . \wedge \ z) \tag{16.69}$$

Finally, the mechanics of constructing the solution to the target problem from the solutions to *Problem* $Q(A, C)$ and *Problem* $Q(C, B)$ are as follows. We expand the problem set by including in $\mathcal{P}$ the item $\pi := (A, C, B)$ and set $\mathcal{T}(\pi) = \{(A, C), (C, B)\}$. We then extend the definition of *Linker* accordingly, namely

$$\mathcal{L}\left(\pi, \bigcup_{p' \in \mathcal{T}(\pi)} \{(p', z(p'))\}\right) := \begin{cases} yes & , \ z(p') = yes \ , \forall p' \in \mathcal{T}(p) \\ no & , \ \text{otherwise} \end{cases} \quad \square \tag{16.70}$$

16.4.4 Example

A fundamental problem is graph theory is to determine whether a given graph is cyclic, and if so, to identify the cycles. From the definition of transitive closure it follows that node n is on a cycle if and only $C[n, n] = 1$. So to identify the nodes comprising a cycle, we simply identify identical rows of C whose *diagonal* elements are equal to 1. The nodes corresponding to these rows are on the same cycle.

For instance, consider the cyclic directed graph shown in Figure 16.1. By inspection, all the nodes except node 4 are on cycles. There are two cycles: one consisting of the nodes $1, 2$ and 3 and one of the nodes $5, 6$ and 7.

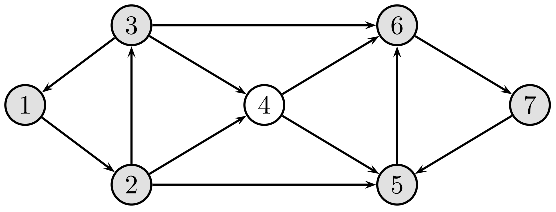


Figure 16.1: Acyclic graph

This is confirmed by the transitive closure matrix C shown in Table 16.1 with the associated precedence matrix $\mathcal{M}$.

Table 16.1: Precedence matrix $\mathcal{M}$ and its transitive closure C

$\mathcal{M}$	1	2	3	4	5	6	7
1		1					
2			1	1	1		
3	1			1		1	
4					1	1	
5						1	
6							1
7					1		

C	1	2	3	4	5	6	7
1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1
3	1	1	1	1	1	1	1
4					1	1	1
5					1	1	1
6					1	1	1
7					1	1	1

By inspection, all the nodes except node 4 are on cycles. Furthermore, there are two cycles: one consisting of the nodes $1, 2$ and 3 and one of the nodes $5, 6$ and 7. Note that although row 4 of C is identical to row 5,6 and 7, node 4 is not on a cycle. This is confirmed by the fact that $C[4, 4] \neq 1$. $\square$

Our next example features an interesting integer-programming problem.

16.4.5 Example

Let c and $\{a_n : 1 \leq n \leq N\}$ be given positive integers such that c is greater than or equal to $\underline{a} := \min\{a_n : 1 \leq n \leq N\}$, and consider the following:

Problem Q:
Find an integer vector $\mathbf{x} = (x_1, \dots, x_N)$ *such that*
$$\sum_{n=1}^{N} a_n x_n = c. \tag{16.71}$$

How would dynamic programming's solution strategy approach this problem?

Proceeding from the observation that *Problem Q* is clearly an instance of the following generic parametric problem

Problem Q(p), $p \in \mathcal{P} := \{0, 1, 2, \dots, c\}$**:**
Find an integer vector $\mathbf{x} = (x_1, \dots, x_N)$ *such that*
$$\sum_{n=1}^{N} a_n x_n = p. \tag{16.72}$$

the idea is to determine whether for a *given* $p \in \mathcal{P}$ (16.72) has an integer solution. Thus, $z(p)$ is an element of the set $\{yes, no\}$ defined as follows:

$$z(p) := \begin{cases} yes & , \text{ if Problem } Q(p) \text{ has an integer solution} \\ no & , \text{ otherwise} \end{cases} \tag{16.73}$$

To derive the dynamic programming functional equation for this problem, observe that since s, $\{a_n\}$ and $\{x_n\}$ are all positive integers, it follows that

· *Problem Q*(0) has the unique solution (0,. . . ,0); and
· *Problem Q*(p), $p \in \mathcal{P}\backslash\{0\}$, has an integer solution if, and only if, for some n, $1 \leq n \leq N$, *Problem Q*$(p - a_n)$ has an integer solution.

Depending on the specific values of c and the coefficients $\{a_n\}$, there can be many modified problems whose solutions are immediate. For example, for any $p \in \mathcal{P}$ such that $p = ka_n$ for some integer k and $n \in \{1, 2, \dots, N\}$, the solution to *Problem Q*(p) is obviously *yes*.

But to keep things simple we set $\mathcal{P}'' = \{0, 1, 2, \dots, \underline{a}\}$ and $\mathcal{P}' = \{\underline{a} + 1, \dots, c\}$, noting that $z(0) = yes$ and $z(p) = no, \forall p \in \mathcal{P}''\backslash\{0\}$.

The relationship between the solutions to the other modified problems would be stated thus:

$$z(p) = \begin{cases} yes & , \ yes \in \{z(p - a_n) : 1 \leq n \leq N\} \\ no & , \ \text{otherwise} \end{cases} \ , \ p \in \mathcal{P}' \tag{16.74}$$

This equation, which is clearly typical of dynamic programming, would be solved iteratively for $s = \underline{a}+1, \underline{a}+2, \dots, c$ — in this order. Formally, we can set

$$\mathcal{P} = \{0, 1, 2, \dots, c\} \tag{16.75}$$

$$\mathcal{P}^{,\prime\prime} = \{0, 1, 2, \dots, \underline{a}\} \tag{16.76}$$

$$\mathcal{P}'' = \{0, \dots, \underline{a}\} \tag{16.77}$$

$$\pi = c \tag{16.78}$$

$$\mathcal{T}(p, x) = p - a_x \ , \ p \in \mathcal{P}', x \in D(p) \tag{16.79}$$

$$D(p) = \{n \in \{1, 2, \dots, N\} : a_n \leq p\} \ , \ p \in \mathcal{P}' \tag{16.80}$$

$$\mathcal{L}\left(p, \bigcup_{x \in D(p)} \{z(\mathcal{T}(p, x))\}\right) = \begin{cases} yes & , \ yes \in \bigcup\limits_{x \in D(p)} z(\mathcal{T}(p, x)) \\ no & , \ otherwise \end{cases} \tag{16.81}$$

Thus, using 1 and 0 instead of *yes* and *no*, we can write the functional equation as follows:

$$z(p) = \vee/\{z(p - a_n) : a_n \leq p, n \in \{1, 2, \dots, N\}\} \tag{16.82}$$

for $p \in \{\underline{a}+1, \underline{a}+2, \dots, c\}$, where for $B \subset \{0, 1\}$, the operation $\vee/B$ yields 1 if, and only if, B contains 1, otherwise it yields 0. Although $\vee/B$ is equivalent to $\max\{b \in B\}$, I prefer to use the former to emphasize that this is inherently a logic operation rather than an arithmetic operation. □

Having demonstrated how versatile and widely applicable dynamic programming's approach to problem solving is, the inevitable question is whether it is possible to single out a specific problem to provide a *paradigm* demonstration of this approach. No doubt, an answer to this question would, in the end be highly subjective.

Still, in my view the following problem offers such an example.

16.5 The Towers of Hanoi Problem

This problem is one of the most famous non-numerical mathematical games. It involves the following situation. Imagine three pegs labelled R, C, L, for **R**ight, **C**enter and **L**eft. N discs of different sizes are placed on peg L in a manner that no large disc is placed on top of a smaller one. The task is to relocate the discs to peg R, ensuring that they will retain the order that they had on peg L. Obviously, peg C can be used as a temporary facility. The rules of the game permit moving one disc at a time, and they dictate that at no time can a larger disc be placed on a smaller one. Figure

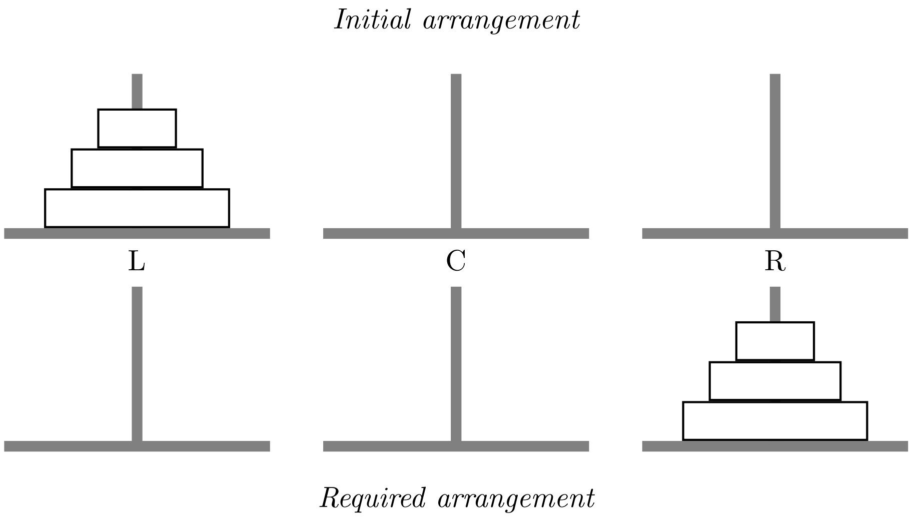


Figure 16.2: Towers of Hanoi problem with 3 pieces

16.2 depicts the original arrangement and the required arrangement of the discs for a problem consisting of $N = 3$ discs.

One can approach this problem from a number of angles. For example, should the problem be viewed as an optimization problem, the objective would be to determine the optimal policy for transferring the discs from peg L to peg R in a manner that would minimize the total number of moves. Or one may settle for determining the minimum number of moves required for this operation — without identifying the optimal policy itself. Be that as it may, the following is a fundamental problem in this analysis:

> **Task:**
> Find a *feasible* — not necessarily optimal — solution for the Towers of Hanoi problem.

Thus, several dynamic programming models can be constructed for this problem. For example, one can reason along these lines:

> As the current configuration determines the range of feasible configurations that can be constructed in *one* move, let the state space S be the collection of all pairs (sr, sl) where sr and sl are possible configurations of discs on peg R, and peg L, respectively. Discs that are not in these sets must by necessity be on peg C. Thus, any such pair gives a complete picture of the situation. The initial state then is specified by $sr = \{1, 2, 3, \dots, N\}$, $sl = \varnothing$ set, whereas the final state is specified by $sr = \varnothing$ and $sl = \{1, 2, 3, \dots, N\}$. The problem therefore amounts to bringing the state from its initial value to the prescribed final value. It thus constitutes a typical *final state* problem.
>
> Next, one would define for each feasible state $s = (sr, sl)$ the associated set of feasible decisions (moves), $D(s)$. Given the game's

strict rules concerning the moves allowed, the formulation of the sets $\{D(s)\}$ is rather simple. Then one would define the transition function t, which is also a rather simple task, and so on.

This line of reasoning is of course valid and if followed through will lead to a valid dynamic programming model. The point is though that the result will be a rather cumbersome dynamic programming model.

How then would one obtain a *simple* dynamic programming model?

The first thing to observe is that the number of discs, as well as the designated target and original pegs, can be viewed as *parameters*. This means that we can consider the following *Problem Space*:

$$\mathcal{P} := \{(n, O, T) : O, T \in \{L, C, R\}, O \neq T, n \in 1, 2, \dots, N\} \tag{16.83}$$

In other words, all we need to know about the current configuration (problem) is:

1. Number of discs to be moved, call it n.
2. The peg on which the discs are located, call it O (for "origin")
3. The peg to which the discs are to be moved, call it T (for "target").

For example, $p = (5, C, L)$ describes a problem that requires moving 5 discs from the Center peg to the Left peg.

Notice that this choice of *Problem Space* will allow viewing the target problem as an instance of the following family of problems:

Problem $\mathbf{Q}(p)$, $p = (n, O, T) \in \mathcal{P}$:
Find a solution for the Towers of Hanoi problem where n discs are to be moved from peg O to peg T.

That is, since the target problem is equivalent to *Problem* $Q(N, L, R)$, we can set $\pi := (N, L, R)$ to represent the target problem. The problem is trivial if $n = 1$, the solution being

$$z(1, O, T) = \text{" move a disc from peg } O \text{ to peg } T \text{ "} \tag{16.84}$$

We may therefore set $\mathcal{P}'' = \{(n, O, T) \in \mathcal{P} : n = 1\}$ to be the set of unsupported problems and $\mathcal{P}' = \{(n, O, T) \in \mathcal{P} : n > 1\}$.

For convenience we adopt the following notation: for a typical state say (n, O, T), let $\neg(O, T)$ denote the third peg, namely let $\neg(O, T)$ denote the *complement* of $\{O, T\}$ with respect to $\{L, C, R\}$. For example, $\neg(L, R) = C$.

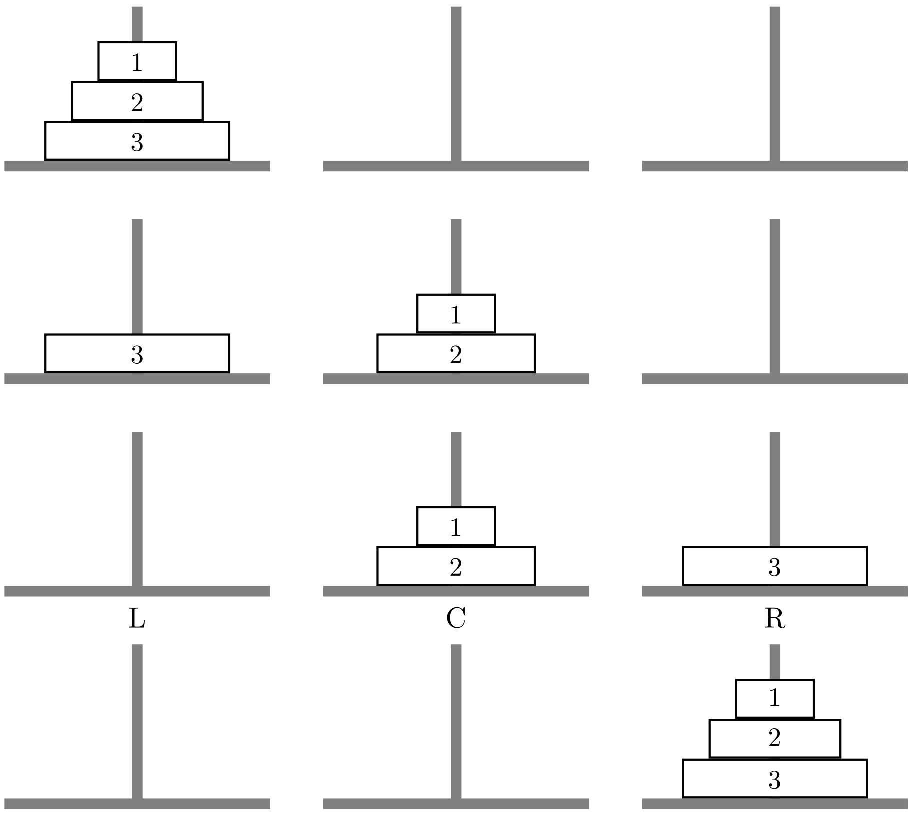
The second point to observe is that, the task of moving n discs from peg O to peg T can be broken down into three related sub-tasks:

Subtask #1: move $n - 1$ discs from peg O to peg $\neg(O, T)$.

Subtask #2: move one disc from peg O to peg T.

Subtask #3: move $n - 1$ discs from peg $\neg(O, T)$ to peg T.

Initial arrangement



Required arrangement

Figure 16.3: Towers of Hanoi problem with 3 pieces

These sub-tasks are illustrated in Figure 16.3 for the case where $n = 3$, $O = L$ and $T = R$. For your convenience the discs are labeled.

The functional equation obtained in this case would therefore be as follows:

$$z(n,O,T) = z(n-1,O,\neg(O,T)) \circ z(1,O,T) \circ z(n-1,\neg(O,T),T) \quad (16.85)$$

where $\circ$ denotes *concatenation.* Note that $(1,O,T)$ is a terminal state for which we have

$$z(1,O,T) := \text{“move a disc from peg } O \text{ to peg } T\text{”} \quad (16.86)$$

Thus, to construct the solution for *Problem* $Q(3,L,R)$, we first apply (16.86) with $s = (n = 3, O = L, T = R)$ and obtain:

$$z(3,L,R) = z(2,L,C) \circ\ z(1,L,R) \circ\ z(2,C,R) \quad (16.87)$$

Next, we apply (16.73) to $s = (n = 2, O = L, T = C)$, which yields

$$z(2,L,C) = z(1,L,R) \circ z(1,L,C) \circ z(1,R,C) \quad (16.88)$$

and then we apply it to $s = (n = 2, O = C, T = R)$ to obtain

$$z(2,C,R) = z(1,C,L) \circ\ z(1,C,R) \circ\ z(1,L,R) \quad (16.89)$$

Substituting these values of $Z(2,L,C)$ and $Z(2,C,R)$ in (16.87) yields

$$\begin{aligned} z(3,L,R) = z(1,L,R) \circ\ z(1,L,C) \circ\ z(1,R,C) \circ\ z(1,L,R) \circ\ z(1,C,L) \\ \circ\ z(1,C,R) \circ\ z(1,L,R) \quad (16.90) \end{aligned}$$

Hence, substituting the seven values of $z(1,O,T)$ in (16.90) with the corresponding expressions given by (16.86) and listing the moves on separate lines, we obtain the following verbally phrased solution:

Move a disc from peg L to peg R
Move a disc from peg L to peg C
Move a disc from peg R to peg C
Move a disc from peg L to peg R
Move a disc from peg C to peg L
Move a disc from peg C to peg R
Move a disc from peg L to peg R

This solution is depicted in Figure 16.4.

To complete the picture I shall now formulate the problem in terms of our abstract dynamic programming model.

Target Problem:
Problem Q:
Find a feasible solution for the Towers of Hanoi problem that requires relocating N discs from peg L to peg R.

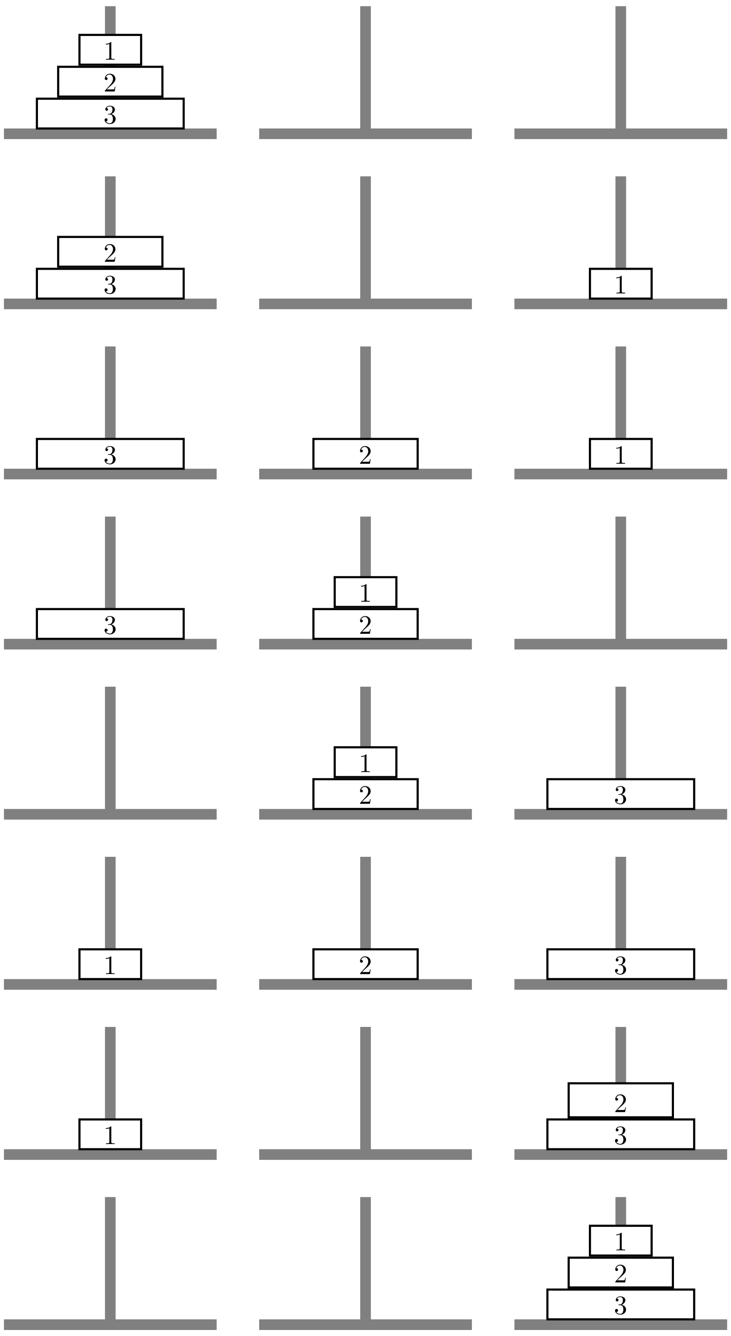


Figure 16.4: Solution to the Towers of Hanoi problem with 3 pieces

Problem Space:

$$\mathcal{P} := \{(n, O, T) : O, T \in \{L, C, R\}, O \neq T, n \in \{1, 2, 3, \dots, N\}\} \tag{16.91}$$

Modified Problems: Problem Q(n,O,T), $(\mathbf{n}, \mathbf{O}, \mathbf{T}) \in \mathcal{P}$**:**
Find a feasible solution for the Towers of Hanoi problem that requires relocating n discs from peg O to peg T.

Unsupported Problems:

$$\mathcal{P}'' := \{(n, O, T) \in \mathcal{P} : n = 1\} \tag{16.92}$$

Transition Function:

$$\mathcal{T}(p) := \{(n-1, O, \neg(O, T)), (1, O, T), (n-1, \neg(O, T), T)\} \tag{16.93}$$

for $p = (n, O, T) \in \mathcal{P}'$.

Linker:

$$\mathcal{L}\left(p, \bigcup_{p' \in \mathcal{T}(p)} \{(p', z(p'))\}\right) := z(n-1, O, \neg(O,T)) \circ z(1, O, T) \circ z(n-1, \neg(O,T), T) \tag{16.94}$$

for $p = (n, O, T) \in \mathcal{P}'$.

Functional Equation:

$$z(n, O, T) = z(n-1, O, \neg(O,T)) \circ z(1, O, T) \circ z(n-1, \neg(O,T), T) \tag{16.95}$$

The point to note about the feasible solution yielded by this functional equation is that it is, in fact, an *optimal solution* to the problem considered in that it minimizes the number of moves required to accomplish the task.

The reader may wish to track down the latent min operation in this functional equation! □

Remark:
The above functional equation has a simple *closed-form* solution meaning that solving it iteratively is unnecessary. More on this fascinating problem can be found in Sniedovich (2002b), including the case where the number of moves is to be *maximized* — without cycling. Note that the number of moves for the min problem is $2^n - 1$, whereas the number of moves for the max problem is $3^n - 1$.

16.6 Optimization-Free Dynamic Programming

My main objective in demonstrating dynamic programming's handling of non-optimization problems was to corroborate my thesis that dynamic programming's approach to problem solving is valid outside the area of optimization and that it is in fact employed widely, if inadvertently, in various disciplines.

I wish to make it clear, though, that this thesis is by no means a revelation. That dynamic programming is applicable outside the realm of optimization has been noted, for instance by Karp [1982], and by Verdo and Poor [1987]. Indeed, Bellman himself was always aware of this, as he tells us in his autobiography (Bellman 1984, p. 203):

> ...In 1955, Chandrasekhar published his book on radiative transfer. I was curious to see what it contained. From my work at Los Alamos, I knew the equations of radiative transfer were the same as the equations of neutron diffusion. Progress in one area meant progress in the other. My attention was immediately caught by chapters on the principles of invariants. I grasped the idea of Ambarzumian immediately, with the extensions of Chandrasekhar. My first thought was "that is dynamic programming without optimization. ...

Furthermore, Bellman pointed out quite clearly that, both philosophically and mathematically, the roots of dynamic programming, to be precise, the roots of the *Principle of Optimiality* and the functional equation, lie in disciplines that are in no way concerned with optimization (Bellman 1957a, p. 115):

> ...As we have recently shown in connection with some joint work (R. Bellman and R. Kalaba, "On the Principle of Invariant Imbedding and Propagation Through Inhomogeneous Media," *Proc. Nat. Acad. Sci.* (1956)), the "principle of optimality" is actually a particular application of what we have called the "principle of invariant imbedding." A special form of the invariance principle was used by Ambarzumian "On the Scattering of Light by a Diffuse Medium," *C.R. Doklady, Sci. U.R.S.S*, 38 (1943), p. 257 and extensively developed by S. Chandrasekhar *Radiative Transfer*, Oxford, 1950. An early use of the method is due to G. Stokes (*Mathematical and Physical Papers,* Vol. IV, "On the intensity of light reflected from or transmitted through a pile of plates," pp 145-156).
>
> The functional equation technique used throughout is intimately related to the "Point of Regeneration" method used in the study of branching processes, cf. R. Bellman and T.E. Harris, "On Age Dependent Binary Branching Processes," *Ann. Math.*, Vol. 55, (1952), pp. 280-295.
>
> Actually, we have made no systematic effort to trace the origin and use of invariance principles, and the above references represent only a few of many that could be cited. One, however, which cannot be ignored is J. Hadamard, "Le principle de Huygens," *Bull. Soc. Math.*

> *France*, 52 (1924), pp. 610-640, where there is an interesting discussion on causality, functional equations and Huygens' principle ...

And yet, dynamic programming was developed by Bellman as an optimization method and to this day it is perceived as a method designed for the solution of optimization problems.

It is my view, though, that there are good reasons to believe that dynamic programming's capabilities as a general problem solving strategy will eventually be recognized more widely, simply because the need for strategies of this kind is very real indeed. For instance, it has been acknowledged for some time now in the disciplines of artificial intelligence and expert systems, that the particular tasks that one is required to accomplish in these areas call for the development of *general problem solving methodologies.* Although dynamic programming had been incorporated into such methodologies in these areas, the role assigned it in them is its conventional role as an optimization method (see for example, Rayward-Smith et al. [1988]). But this is not what I am arguing for. I am not arguing for the incorporation of dynamic programming into larger General Problem Solvers. I am proposing that *dynamic programming itself has the capability to perform as a general problem solver.*

The great attraction of adapting dynamic programming into areas not requiring optimization is that this will allow benefiting from the potent solution strategy made available by dynamic programming, without having to be as concerned about the effects of dimensionality. For indications are that however much the *Curse of Dimensionality* will remain a factor to reckon with in areas not involving optimization, it will nevertheless lose much of its sting.

16.7 Concluding Remarks

I have shown that stripping the specialized mathematical idiom from dynamic programming's approach to optimization problems lays bare its *guiding idea* to problem solving.

This idea, I argued is immensely sensible, simple and effective — and therefore valid — not only in the sphere of optimization but also outside it. I attempted to embody this basic idea in the abstract dynamic programming model that I formulated in *Section 14.3.*

That said, I want to make it clear, that I do not wish to give the impression that the proposed abstract model provides an airtight, rigorous formulation of dynamic programming in its capacity as a general approach to problem solving. Obviously, compared with my description of dynamic programming's treatment of an optimization problem — where I framed a precise mathematical definition of the underlying multistage decision model, set down sufficient conditions that in turn assured the validity of the functional equation, and so on — my description of dynamic programming in terms of the abstract model is marked by a degree of vagueness particularly with regard to the derivation of the functional equation and the grounds for its validity.

But this situation seems to be inevitable, considering that the model's principal aim is to capture the gist of dynamic programming's approach to problem solving. A certain amount of vagueness seems to be inevitable in every attempt to a capture the essence, that is the central idea, of a theory, a method, or an approach.

Perhaps the best way to clarify this point is by way of an analogy with the *Principle of Optimality*.

I have gone to great lengths to point out, in *Chapter 13,* that in the context of an accurately defined multistage decision model the principle has a clearly defined role. It can function either as a sufficient condition for the validity of the functional equation; or as a theorem articulating a property of optimal policies. In either case it has a precise and rigorous mathematical formulation.

At this stage, I may add, that the *Principle* can also be seen as the *Guiding Idea* of the theory. That is, by condensing into a concise statement the main point of dynamic programming as an optimization method, the principle gives voice to the objective that it seeks to accomplished. But in this capacity the principle cannot have the rigor or precision that it has in its capacity as either a sufficient condition or a fundamental property. For here, as Bellman [1984, p. 174] points out, its function — like that of any good principle — is to "...guide the intuition. ..."

This remark connects with my previous remarks concerning the chances of dynamic programming being recognized as a *General Problem Solving Method.* It seem that for this to happen, one would have to emulate Bellman's formulation of dynamic programming as an optimization method. In particular, one would have to formulate a concise and elegantly phrased *principle*, describing dynamic programming's guiding idea as a *General Problem Solver.* This principle will have to play a dual-role similar to the one played so effectively by the *Principle of Optimality* in the context of optimization. But this, it seems, is a formidable challenge!

A first step in this direction would be to workout a non-technical, narrative description of dynamic programming as a general problem solving methodology. I plan to address this task in my next book on dynamic programming.

Stay tuned!

Part IV

Appendices

A

Contraction Mapping

A.1 Overview

The purpose of this appendix is to provide a glossary of the definitions, concepts and basic results of functional analysis that are referred to in *Chapter 6.* The object under consideration is the generic functional equation $u = Au$ where $u \in U$ represents an unknown function and A is a map from U into itself.

In this general class I distinguish between two types of equations. In one the map A causes a contraction in u, while in the other the contraction is in the argument of u. I refer to these two types of contraction as contraction in the *functional space* and contraction in the *domain space*, respectively. In the context of dynamic programming functional equations u represent the (optimal) return (objective) function and its argument is the state variable. Hence, in this context the distinction is between a contraction in the *return space* and contraction in the *state space*.

A.2 Metric Spaces

Definition A.2.1 *A* METRIC SPACE *is a pair* (U, Θ)*, where* U *is a non-empty set and* Θ *is a real-valued function on* $U \times U$ *satisfying the following conditions:*

(a) $\Theta(u, v) \geq 0$, $\forall u, v \in U$.

(b) $\Theta(u, v) = 0$ *if, and only if,* $v = u$.

(c) $\Theta(u, v) = \Theta(v, u)$, $\forall u, v \in U$ *(Axiom of symmetry).*

(d) $\Theta(u,v) \leq \Theta(u,w) + \Theta(w,v)$, $\forall u,v,w \in U$ *(Triangle Axiom).* □

Lemma A.2.1 *Let U denote the set of all bounded real-valued functions on some set S, and define*

$$\Theta(u,v) := \sup_{z \in Z} |u(z) - v(z)| \ , \ u,v \in U \tag{A.1}$$

where $|a|$ denotes the absolute value of a. Then, (U,Θ) is a metric space.

Proof. Since all the elements of U are bounded, Θ is well defined on U^2 and $(a) - (c)$ are trivially true. That the Triangle Axiom holds is assured by the relation $|a+b| \leq |a| + |b|$, which is valid for any pair of real numbers (a,b). □

Definition A.2.2 *A sequence $\{u_n\}$ in a metric space (U,Θ) is said to be a* FUNDAMENTAL, *or* CAUCHY, *sequence if for every $\epsilon > 0$ there is an integer M such that $\Theta(u_k,u_m) < \epsilon$ for all $k,m \geq M$.* □

Definition A.2.3 *A metric space (U,Θ) is said to be* COMPLETE *if every Cauchy sequence in this space converges to some element in U.* □

Lemma A.2.2 *The metric space defined in Lemma A.2.1 is complete.*

Proof. Let (U,Θ) be the metric space defined in *Lemma A.2.1,* and let $\{u_m\}$ be any Cauchy sequence in this space. By definition, for any $\epsilon > 0$ there is a positive integer M such that

$$\sup_{z \in Z} |u_k(z) - u_m(z)| < \epsilon \ , \ \forall k,m \geq M \tag{A.2}$$

so that,

$$|u_k(z) - u_m(z)| < \epsilon \ , \ \forall z \in Z, k,m \geq M \tag{A.3}$$

Hence, the sequence $\{u_n\}$ meets the Cauchy criterion for uniform convergence of real-valued functions (See Rudin [1964, Theorem 7.8, *p.* 134]). This means that this sequence must converge to some real-valued function $u \in U$. It follows therefore that (U,Θ) is a complete metric space. □

Definition A.2.4 *Let (U,Θ) be a metric space and let A be a map on U into itself. That is, assume that $Au \in U$, $\forall u \in U$, where Au denotes the value assigned to u by A. The* MODULUS *of A is the smallest number α satisfying the condition*

$$\Theta(Au,Av) \leq \alpha\Theta(u,v) \ , \ \forall u,v \in U \qquad \square \tag{A.4}$$

A.2.1 Example

Let Z be an interval $[a, b]$, $0 \leq a \leq b < \infty$ and let U be the set of all bounded real-valued functions on Z. Consider now the metric Θ defined by (A.1) and the map A defined as follows:

$$(Au)(z) := \sup_{0\leq x\leq z} \{x^2 + \beta u(z-x)\} \ , \ z \in Z, u \in U \tag{A.5}$$

where β is some constant. Then,

$$\Theta(Au, Av) = \sup_{z\in Z} \left| \sup_{0\leq x\leq z} \{x^2 + \beta u(z-x)\} - \sup_{0\leq x\leq z} \{x^2 + \beta v(z-x)\} \right| \tag{A.6}$$

so that an appeal to Lemma A.2.5 yields

$$\begin{aligned}
\Theta(Au, Av) &\leq \sup_{z\in Z}\{ \sup_{0\leq x\leq z} \big|\{x^2 + \beta u(z-x)\} - \{x^2 + \beta v(z-x)\}\big|\} && \text{(A.7)}\\
&= \sup_{z\in Z} \sup_{0\leq x\leq z} |\beta[u(z-x) - v(z-x)]| && \text{(A.8)}\\
&= \sup_{a\leq z\leq b} |\beta[u(z) - v(z)]| && \text{(A.9)}\\
&= \sup_{z\in Z} |\beta[u(z) - v(z)]| && \text{(A.10)}\\
&= \sup_{z\in Z} |\beta|[u(z) - v(z)]| && \text{(A.11)}\\
&= |\beta| \sup_{z\in Z}[u(z) - v(z)]| && \text{(A.12)}\\
&= |\beta|\Theta(u, v) && \text{(A.13)}
\end{aligned}$$

Thus, the modulus of A is not greater than $\alpha = |\beta|$. □

Lemma A.2.3 *Let (U, Θ) be a metric space and let A be a map on U into itself whose modulus is in the range $0 \leq \alpha \leq 1$. Also, let $\{u_m\}$ be any sequence in U whose limit u' is an element of U. Then, the sequence $\{Au_m\}$ converges to Au', and if $u_m = Au_{m-1}, \forall m > 1$ then $u' = Au'$.*

Proof. Let (U, Θ) be a metric space, A a map on U into itself and, $\{u_m\}$ a sequence in U satisfying all the above conditions. We need to show that the sequence $\{Au_m\}$ converges to Au', with u' denoting the limit of $\{u_m\}$. Observe then that (A.4) entails that

$$\Theta(Au_m, Au') \leq \alpha\Theta(u_m, u') \ , \ \forall m = 1, 2, 3, \ldots \tag{A.14}$$

so that $0 \leq \alpha \leq 1$ ensures that

$$\Theta(Au_m, Au') \leq \Theta(u_m, u') \ , \ \forall m = 1, 2, 3, \ldots \tag{A.15}$$

Since $\{u_m\}$ converges to u', it follows that the right-hand side of (A.15) converges to zero as m increases to infinity, hence the sequence $\{Au_m\}$ must converge to Au'.

Next, if $u_{m+1} = Au_m, \forall m > 1$, it follows that the two sequences $\{u_m\}$ and $\{Au_m\}$ converge to the same point in U, hence $u' = Au'$. □

A.3 Contraction in the Functional Space

Definition A.3.1 *Let (U, Θ) be a metric space and A a map on U into itself. Then, A is said to be a* CONTRACTION MAPPING *if there exists an α, $0 \leq \alpha < 1$, satisfying (A.4).* □

Lemma A.3.1 *Let (U, Θ) be a metric space, A a contraction mapping on it, and u_0 any element of U. Consider the sequence $\{u_m : m = 0, 1, 2, 3, \ldots\}$, where $u_{m+1} = Au_m$, $m = 0, 1, 2, 3, \ldots$. Then, this sequence is a Cauchy sequence.*

Proof. Assume that the above conditions are met. Let $\{u_m\}$ be any sequence such that $u_0 \in U$ and $u_{m+1} = Au_m$, $m = 1, 2, 3, \ldots$. Since A is a contraction mapping, it follows that

$$\Theta(Au_{m+1}, Au_m) \leq \alpha\Theta(u_{m+1}, u_m) \ , \ \forall m = 0, 1, 2, 3 \ldots \tag{A.16}$$

for some $0 \leq \alpha < 1$.

Next, because by construction $u_{m+1} = Au_m$, then

$$\Theta(u_2, u_1) = \Theta(Au_1, Au_0) \leq \alpha\Theta(u_1, u_0) \tag{A.17}$$

$$\Theta(u_3, u_2) = \Theta(Au_2, Au_1) \leq \alpha\Theta(u_2, u_1) \leq \alpha^2\Theta(u_1, u_0) \tag{A.18}$$

By induction therefore,

$$\Theta(u_{m+1}, u_m) \leq \alpha^m\Theta(u_1, u_0) \ , \ m = 1, 2, 3, \ldots \tag{A.19}$$

Now, successive appeals to the Triangle Axiom yields

$$\Theta(u_k, u_m) \leq \sum_{n=m}^{k-1} \Theta(u_n, u_{n+1}) \ , \ \forall k > m \tag{A.20}$$

so that (A.19), in conjunction with (A.20), imply that

$$\Theta(u_k, u_m) \leq \sum_{n=m}^{k-1} \alpha^n\Theta(u_1, u_0) \ , \ \forall k > m \tag{A.21}$$

$$\leq \sum_{n=m}^{k-1} \alpha^n\Theta(u_1, u_0) \tag{A.22}$$

Finally, as $0 \leq \alpha < 1$, it follows that

$$\Theta(u_k, u_m) \leq \left[\frac{\alpha^m}{1-\alpha}\right]\Theta(u_1, u_0) \ , \ \forall k \geq m \tag{A.23}$$

The fact that $0 \leq \alpha < 1$ implies that, for any $\epsilon > 0$ there is an M such that the right-hand side of (A.23) is smaller than ϵ for any pair (k, m) such that $k, m \geq M$. Hence, $\{u_m\}$ is a Cauchy sequence. □

Theorem A.3.1 Fixed Point.
Let (U, Θ) be a complete metric space, A a contraction mapping on it and u_0 any element of U. Consider the sequence $\{u_m\}$, where $u_{m+1} = Au_m$, $m = 0, 1, 2, 3, \ldots$. Then,

1. *The sequence $\{u_m\}$ converges to some $u' \in U$.*

2. *u' is the unique solution in U to the equation $u = Au$.*

Proof. In view of *Lemma A.3.1,* any sequence satisfying the above conditions is a Cauchy sequence. Considering that the metric space here is complete, any sequence satisfying these conditions converges to some $u' \in U$. Hence, part (1) is true.

Next, as the modulus of A is in the range $0 \leq \alpha \leq 1$, and by construction $u_{m+1} = Au_m$, it follows from *Lemma A.2.3* that $u' = Au'$. To show that the equation $u = Au$ has a unique solution in U, assume that there is a pair $(u, v) \in U^2$ such that $u = Au$ and $v = Av$. We have to show that $u = v$.

Since A is a contraction mapping, then

$$\Theta(u, v) = \Theta(Au, Av) \leq \alpha\Theta(u, v) \ , \ 0 \leq \alpha < 1 \tag{A.24}$$

Next, because $0 \leq \alpha < 1$, it follows that $\Theta(u, v) = 0$, hence $u = v$. □

In summary then, the consequence of a contraction in the functional space is as follows: Let $\{u_m\}$ and $\{v_m\}$ be any two sequences such that $u_m = Au_{m-1}$ and $v_m = Av_{m-1}, m > 1$. In cases where the contraction is in the functional space we have

$$\begin{aligned}
\Theta\left(u_m, v_m\right) &= \Theta\left(Au_{m-1}, Av_{m-1}\right) && \text{(A.25)}\\
&\leq \alpha\Theta\left(u_{m-1}, v_{m-1}\right) && \text{(A.26)}\\
&= \alpha\Theta\left(Au_{m-2}, Av_{m-2}\right) && \text{(A.27)}\\
&\leq \alpha^2\Theta\left(u_{m-2}, v_{m-2}\right) && \text{(A.28)}
\end{aligned}$$

and by induction

$$\Theta\left(u_m, v_m\right) \leq \alpha^m\Theta\left(u_0, v_0\right) \ , \ m = 1, 2, 3, \ldots \tag{A.29}$$

If follows then that if $0 \leq \alpha < 1$, then

$$\lim_{m\to\infty} \Theta\left(u_m, v_m\right) = 0 \tag{A.30}$$

Thus, if $\{u_m\}$ converges to some $u'' \in U$, then clearly $\{v_m\}$ must also converge to u'', regardless of what the initial approximations u_0 and v_0 are. Furthermore, $u'' = Au''$.

Definition A.3.2 *A functional equation $u = Au, u \in U$ is said to be a* type-one equation *if there exits a Θ such that (U, Θ) is a complete metric space and A is contraction mapping on U.* □

Theorem A.3.1 outlines a scheme for solving such equations. The relevance of this fact to dynamic programming functional equations is discussed in *Chapter 6.*

A.4 Contraction in the Domain Space

The distinctive feature of equations falling in this group is that the contraction in their case occurs in the *domain space* rather than in the functional space. To make clear the distinction between the two types of contraction consider the following.

A.4.1 Example

Consider the functional equation

$$u(z) = z^2 + u(\beta z) \ , \ 0 \leq z \leq r \tag{A.31}$$

where β is some numeric constant, and let

$$(Au)(z) := z^2 + u(\beta z) \ , \ 0 \leq z \leq r \tag{A.32}$$

Then

$$\begin{aligned} \Theta(Au, Av) &= \sup_{0 \leq z \leq r} |(Au)(z) - (Av)(z)| && \text{(A.33)} \\ &= \sup_{0 \leq z \leq r} \left| \{z^2 + u(\beta z)\} - \{z^2 + v(\beta z)\} \right| && \text{(A.34)} \\ &= \sup_{0 \leq z \leq \beta r} |u(z) - v(z)| && \text{(A.35)} \end{aligned}$$

Now, suppose that $0 \leq \beta \leq 1$. Then, clearly

$$\sup_{0 \leq z \leq \beta r} |u(z) - v(z)| \leq \sup_{0 \leq z \leq r} |u(z) - v(z)| \tag{A.36}$$

and therefore

$$\begin{aligned} \Theta(Au, Av) &\leq \sup_{0 \leq z \leq r} |u(z) - v(z)| && \text{(A.37)} \\ &= \Theta(u, v) && \text{(A.38)} \end{aligned}$$

Hence,

$$\Theta(Au, Av) \leq \alpha \Theta(u, v) \ , \ \alpha = 1 \tag{A.39}$$

However, since $\alpha = 1$, there is no contraction in the functional space. On the other hand, define

$$\begin{aligned} Z &: = [0, r] && \text{(A.40)} \\ Z(m) &: = [0, \beta^m r] \ , \ m = 0, 1, 2, 3, \ldots && \text{(A.41)} \\ \Theta_Y(u, v) &: = \sup_{z \in Y} |u(z) - v(z)| \ , \ Y \subseteq Z && \text{(A.42)} \end{aligned}$$

Then, (A.42) yields

$$\Theta(Au, Av) \leq \Theta_{Z(1)}(u(z) - v(z)) \tag{A.43}$$

Now, let $\{u_m\}$ and $\{v_m\}$ be any two sequences such that $u_m = Au_{m-1}$ and $v_m = Av_{m-1}, m > 0$, successive appeals to (A.43) yield

$$\begin{aligned}\Theta\left(u_m, v_m\right) &= \sup_{0\leq z\leq r} |u_m(z) - v_m(z)| && \text{(A.44)}\\ &= \sup_{0\leq z\leq r} |(Au_{m-1})(z) - (Av_{m-1})(z)| && \text{(A.45)}\\ &= \Theta_{Z(0)}\left(Au_{m-1}, Av_{m-1}\right) && \text{(A.46)}\\ &\leq \Theta_{Z(1)}\left(u_{m-1}, v_{m-1}\right) && \text{(A.47)}\\ &\leq \Theta_{Z(2)}\left(u_{m-2}, v_{m-2}\right) && \text{(A.48)}\end{aligned}$$

and by induction

$$\Theta\left(u_m, v_m\right) \leq \Theta_{Z(m)}\left(u_0, v_0\right) \;,\; m = 1, 2, 3, \ldots \tag{A.49}$$

It follows then that if $0 \leq \beta < 1$, then the sequence $\{Z(m)\}$ has the property that $Z(m+1) \subset Z(m), \forall m > 0$ and as $m \to \infty$ it converges to the singleton $Z^* = \{z^*\}$, $z^* = 0$. Since $z^* \in Z(m)$ for all $m \geq 0$, we say that there is a *contraction in the domain space Z*.

The point to note here is that (A.49) indicates that the contraction, thereby the convergence of the sequence concerned, is crucially affected by the nature of the initial approximation u_0 at $z^* = 0$. For example, suppose that u_0 and v_0 are real-valued functions on $Z = [0, r]$ such that $u_{(0)}(0) = v_0(0)$, and that both are continuous at $z = 0$. Since the sequence $\{Z(m)\}$ converges to the singleton $\{0\}$, it follows that

$$\lim_{m\to\infty} \Theta_{Z(m)}\left(u_0, v_0\right) = 0 \tag{A.50}$$

Hence, (A.49) implies that if the sequence $\{u_m\}$ converges to some $u'' \in U$ such that $u''(0) = u_0(0)$ and u'' is continuous at $z = 0$, the sequence $\{v_m\}$ must also converge to u''. □

That said, observe however that these conditions do not provide for an effective contraction in the functional space. Therefore, to render the contraction in the domain space effective in the functional space as well, additional measures need to be taken. Consider then the following.

Assumption A.4.1
1. There exist a metric Θ' on Z, an element $z^* \in Z$ and a real number $K \in [0, \infty)$ such that

$$\Theta'(z, z^*) \leq K \;,\; \forall z \in Z \tag{A.51}$$

2. There exists a scalar $\alpha \in [0,1)$, such that

$$\Theta_{Z(m)}(Au, Av) \leq \Theta_{Z(m+1)}(u,v) \ , \ \forall u, v \in U, m = 0,1,2,3,\dots \tag{A.52}$$

where

$$Z(m) := \left\{ z \in Z : \Theta'(z, z^*) \leq \alpha^m K \right\} \ , \ m = 0,1,2,3,\dots \tag{A.53}$$

$$\Theta_Y(u,v) := \sup_{z \in Y} |u(z) - v(z)| \ , \ Y \subseteq Z, u, v \in U \tag{A.54}$$

Note that the above definitions entail that $Z(0) = Z$ and that $\Theta_Z(u,v) = \Theta(u,v)$. □

Continuing with *Example A.4.1,* suppose that we set $z^* = 0$ and let

$$\Theta'(z,z') := |z - z'| \ , \ z, z' \in Z := [0,r] \tag{A.55}$$

Since $Z = [0,r]$, we can now set $K = r$ so that $\Theta'(z,z^*) \leq K$, $\forall z \in Z$. Also, if we set $\alpha = \beta$, then

$$Z(m) = [0, \beta^m r] \ , \ m = 0,1,2,3,\dots \tag{A.56}$$

Therefore, (A.52) is true. □

As for provisions to ensure uniqueness, consider the following.

Assumption A.4.2 *There exists a real number c such that*

$$\sum_{m=0}^{\infty} \Theta_{Z(m)}(Aw, w) < \infty \ , \ w(z) := c, \forall z \in Z \qquad \square \tag{A.57}$$

These two assumptions define another type of functional equation. This type of equation is discussed in *Chapter 6.*

Definition A.4.1 *We say that $u = Au$ is a* FUNCTIONAL EQUATION OF TYPE TWO *if there exists a Θ such that (U, Θ) is a metric space satisfying Assumptions A.4.1 and Assumption A.4.2.* □

Continuing with *Example A.4.1,* set $c = 0$. Since $\alpha = \beta$ and $Z(m) = [0, \beta^m r]$, it follows that

$$\Theta_{Z(m)}(Aw, w) = \sup_{0 \leq z \leq r\beta^m} |(Aw)(z) - w(z)| \tag{A.58}$$

$$= \sup_{0 \leq z \leq r\beta^m} |\{z^2 + w(\beta z)\} - w(z)| \tag{A.59}$$

$$= \sup_{0 \leq z \leq r\beta^m} |\{z^2 + c\} - c| \tag{A.60}$$

$$= \sup_{0 \leq z \leq r\beta^m} |z^2| \tag{A.61}$$

$$= (r\beta^m)^2 \tag{A.62}$$

Hence,

$$\sum_{m=0}^{\infty} \Theta_{Z(m)}(Aw, w) = \sum_{m=0}^{\infty} (r\beta^m)^2 \tag{A.63}$$

$$= r^2 \sum_{m=0}^{\infty} \beta^{2m} \tag{A.64}$$

$$= \frac{r^2}{1-\beta^2} < \infty \qquad \square \tag{A.65}$$

Now, let $U(c)$, $c \in \mathbb{R}$, denote the set of all the bounded real-valued functions on Z whose members are continuous at z^* and equal to c there. Also, let $\{w_m\}$ denote the sequence defined by $w_m = Aw_{m-1}, \forall m > 0$ with $w_0(z) := w(z) = c$, $\forall z \in Z$. Supposing that *Assumptions A.4.1* and *Assumption A.4.2* hold, then the following observations regarding the sequence $\{w_m\}$ are of interest to us.

Observation 1. Since $\Theta_{Z(m)}(u, v)$ is non-negative, it follows from (A.57) that

$$\lim_{m\to\infty} \Theta_{Z(m)}(Aw, w) = 0 \tag{A.66}$$

Also, because the sequence $\{Z(m)\}$ converges to $\{z^*\}$ and $w_1 = Aw$, it follows that w_1 is a member of $U(c)$. Repeating this argument and invoking (A.52), it follows by induction that

$$w_m \in U(c) \ , \ \forall m \geq 0 \tag{A.67}$$

Observation 2. Since by definition $w_{m+1} = Aw_m$, it follows that

$$\Theta(w_2, w_1) \leq \Theta_{Z(1)}(w_1, w_0) \tag{A.68}$$

recalling that by definition $Z(0) = Z$ and $\Theta = \Theta_Z$. Therefore, because

$$\Theta(w_3, w_2) = \Theta(Aw_2, Aw_1) \tag{A.69}$$

it follows from (A.52) that

$$\Theta\left(w_3, w_2\right) = \Theta\left(Aw_2, Aw_1\right) \tag{A.70}$$

$$\leq \Theta_{Z(1)}\left(w_2, w_1\right) \tag{A.71}$$

$$\leq \Theta_{Z(2)}\left(w_1, w_0\right) \tag{A.72}$$

so that by induction,

$$\Theta\left(w_{m+1}, w_m\right) \leq \Theta_{Z(m)}\left(w_1, w_0\right) \ , \ m = 1, 2, 3, \ldots \tag{A.73}$$

Since w_0 and w_1 are continuous at $z = z^*$ and equal there, and the sequence $\{Z(m)\}$ converges to $\{z^*\}$, it follows that the right-hand side of (A.73) converges to zero. Thus,

$$\lim_{m\to\infty} \Theta(w_{m+1}, w_m) = 0 \tag{A.74}$$

Observation 3. For any $m \geq k$ we have

$$\Theta\left(w_m, w_k\right) = \sup_{z\in Z} |w_m(z) - w_k(z)| \tag{A.75}$$

$$= \sup_{z\in Z} \left| \sum_{i=k}^{m-1} [w_{i+1}(z) - w_i(z)] \right| \tag{A.76}$$

so that invoking the Triangle Axiom we obtain

$$\Theta\left(w_m, w_k\right) \leq \sum_{i=k}^{m-1} \Theta\left(w_{i+1}, w_i\right). \tag{A.77}$$

It therefore follows from (A.73) that

$$\Theta\left(w_m, w_k\right) \leq \sum_{i=k}^{m-1} \Theta_{Z(i)}(w_1, w_0) \tag{A.78}$$

$$= \sum_{i=k}^{m-1} \Theta_{Z(i)}(Aw, w) \tag{A.79}$$

recalling that by definition $w_0 = w$ and $w_1 = Aw$. □

And with this in mind, consider now the following.

Theorem A.4.1 *Let $u = Au$ be any functional equation satisfying Assumption A.4.1 and Assumption A.4.2. Then, the sequence $\{w_m\}$ converges to some $w'' \in U(c)$ such that $w'' = Aw''$.*

Proof. First, note that (A.57), in conjunction with (A.79), imply that for any $\epsilon > 0$ there is an M such that $\Theta\left(w_m, w_k\right) \leq \epsilon$ for all $m, k \geq M$. Hence, $\{w_m\}$ is a Cauchy sequence. Since (U, Θ) is a complete metric space it follows that the sequence $\{w_m\}$ converges to some $w'' \in U$. Furthermore, because the convergence is uniform, it follows from (A.67) that $w'' \in U(c)$. To show that $w'' = Aw''$, note that by construction $Z(1) \subseteq Z(0) = Z$ so that (A.52) entails that

$$\Theta(Au, Av) \leq \Theta(u, v) \ , \ \forall u, v \in U \tag{A.80}$$

The inference then is that *Assumption A.4.1* holds whereupon in view of Lemma A.2.3 we may conclude that $w'' = Aw''$. □

Having obtained satisfactory results for the sequence induced by the initial approximation $w^{(0)} = w$ let us now consider the more general case.

Theorem A.4.2 *If $u = Au$ is a functional equation of Type Two, then it has at most one solution in $U(c)$.*

Proof. Let $u = Au$ be any Type Two functional equation and assume that there exists a pair $(u, v) \in U(c) \times U(c)$ such that $u = Au$ and $v = Av$. To prove the theorem's validity, we need to show that $u = v$. Note then that, because $u = Au$ and $v = Av$, we have

$$\Theta(u, v) = \Theta(Au, Av) \tag{A.81}$$

so that this, in conjunction with (A.52), entail that

$$\Theta(u, v) \leq \Theta_{Z(1)}(u, v) \tag{A.82}$$

because, by construction, $Z(0) = Z$ and $\Theta = \Theta_Z$. Also in view of $u = Au$ and $v = Av$, it follows from (A.82) that

$$\Theta(u, v) \leq \Theta_{Z(1)}(Au, Av) \tag{A.83}$$

Thus, again invoking (A.52) we deduce from (A.83) the following

$$\Theta(u, v) \leq \Theta_{Z(2)}(Au, Av), \tag{A.84}$$

so that by induction,

$$\Theta(u, v) \leq \Theta_{Z(m)}(u, v), \forall m = 0, 1, 2, 3, \ldots \tag{A.85}$$

Given that $z^* \in Z(m)$ for all $m \geq o$, the sequence $\{Z(m)\}$ converges to $\{z^*\}$ and u and v are continuous at z^* and equal there, (A.85) yields $\Theta(u, v) = 0$. Hence, $u = v$. □

The question is then whether the initial approximation $u_0 \in U(c)$ is a factor in the convergence of a sequence $\{u_m\}$ generated by $u_{m+1} = Au_m$ to the unique solution of $u = Au$ in $U(c)$. Consider then the following.

Theorem A.4.3 *Let $u = Au$ be any Type Two functional equation and let u_0 be any element of $U(c)$. Then the sequence $\{u_m\}$ generated by $u_{m+1} = Au_m$ converges to the unique solution of this equation in $U(c)$.*

Proof. Given the foregoing results, it is sufficient to show that for any $u_0 \in U(c)$ the sequences $\{u_m\}$ and $\{w_m\}$ converge to the same point. Now, by construction $u_{m+1} = Au_m$ and $w_{m+1} = Aw_m$ so that

$$\Theta\left(u_{m+1}, w_{m+1}\right) = \Theta\left(Au_m, Aw_m\right) \ , \ \forall m = 0, 1, 2, 3, \ldots \tag{A.86}$$

Thus, successive appeals of (A.86) to (A.52) yields

$$\Theta\left(u_m, w_m\right) \leq \Theta_{Z(m)}(u, w) \ , \ \forall m = 0, 1, 2, 3, \ldots \tag{A.87}$$

Since by construction $z^* \in Z(m)$ for all $m \geq 0$ and the sequence $\{Z(m)\}$

converges to $\{z^*\}$, it follows from (A.87) that the sequence $\{\Theta(u_m, w_m)\}$ converges to zero, given that u and w are continuous at z^* and equal there. Therefore, considering that $\{w_m\}$ converges to w'' it follows that $\{u_m\}$ also converges to w''. □

Continuing with *Example A.4.1,* since $c = 0$, set $w(z) = 0$, $\forall z \in Z$, by construction

$$\begin{aligned}
w^{(1)}(z) :&= (Aw^{(0)})(z)\ ,\ 0 \leq z \leq r && \text{(A.88)}\\
&= z^2 + w^{(0)}(\beta z) && \text{(A.89)}\\
&= z^2 + 0 && \text{(A.90)}\\
&= z^2 && \text{(A.91)}\\
w^{(2)}(z) :&= (Aw^{(1)})(z), 0 \leq z \leq r && \text{(A.92)}\\
&= z^2 + w^{(1)}(\beta z) && \text{(A.93)}\\
&= z^2 + (\beta z)^2 && \text{(A.94)}\\
&= z^2(1 + \beta^2) && \text{(A.95)}\\
w^{(3)}(z) :&= (Aw^{(2)})(z), 0 \leq z \leq r && \text{(A.96)}\\
&= z^2 + w^{(2)}(\beta z) && \text{(A.97)}\\
&= z^2 + (\beta z)^2(1 + \beta^2) && \text{(A.98)}\\
&= z^2(1 + \beta^2 + \beta^4) && \text{(A.99)}
\end{aligned}$$

By induction,

$$\begin{aligned}
w^{(m)}(z) &= z^2 \sum_{k=0}^{m-1} \beta^{2k}, m \geq 1, 0 \leq s \leq r && \text{(A.100)}\\
&= z^2 \frac{1 - \beta^{2m}}{1 - \beta^2} && \text{(A.101)}
\end{aligned}$$

Hence,

$$\begin{aligned}
w''(z) :&= \lim_{m \to \infty} w^{(m)}(z), 0 \leq s \leq r && \text{(A.102)}\\
&= \lim_{m \to \infty} z^2 \frac{1 - \beta^{2m}}{1 - \beta^2} && \text{(A.103)}\\
&= \frac{z^2}{1 - \beta^2} && \text{(A.104)}
\end{aligned}$$

recalling that $0 < \beta < 1$.

To ascertain that w'' is indeed a solution to the equation $u = Au$, define

$$w^\circ(z) := (Aw)(z), 0 \leq z \leq r \qquad \text{(A.105)}$$

We need to show then that $w^\circ = w''$. Now, by definition

$$\begin{aligned} w^\circ(z) &= z^2 + w''(\beta z), 0 \leq z \leq r && \text{(A.106)}\\ &= z^2 + \frac{(\beta z)^2}{1-\beta^2} && \text{(A.107)}\\ &= z^2\left[1 + \frac{\beta^2}{1-\beta^2}\right] && \text{(A.108)}\\ &= w''(z) && \text{(A.109)}\end{aligned}$$

Hence, $w^\circ = w''$. $\square$

The following result is often used in conjunction with the metric space specified by (A.1), eg. *Example A.2.1.*

Lemma A.4.1 *Let Z be a non-empty set and let (q_1, q_2) be any pair of bounded real-valued functions on Z. Then,*

$$\left|\sup_{z\in Z} q_1(z) - \sup_{z\in Z} q_2(z)\right| \leq \sup_{z\in Z}|q_1(z) - q_2(z)| \tag{A.110}$$

PROOF. Define

$$q'_n := \sup_{z\in Z} q_n(z)\ ,\ n = 1, 2 \tag{A.111}$$

Since (A.110) is trivially true if $q'_1 = q'_2$, assume that $q'_1 \neq q'_2$. Indeed, with no loss of generality, assume that $q'_1 > q'_2$. Hence, there exists some $\epsilon > 0$ such that

$$q'_1 - q'_2 > \epsilon \tag{A.112}$$

observing that we can make ϵ arbitrarily small.

This implies the existence of some $z^\circ \in Z$ such that

$$q_2(z^\circ) \leq q'_2 < q'_1 - \epsilon \leq q_1(z^\circ) \tag{A.113}$$

which in turn implies that

$$q'_1 - q'_2 - \epsilon < q_1(z^\circ) - q_2(z^\circ) \leq \sup_{z\in Z}|q_1(z) - q_2(z)|. \tag{A.114}$$

Now, as it is assumed that $q'_1 > q'_2$, it follows that

$$q'_1 - q'_2 = |q'_1 - q'_2|, \tag{A.115}$$

so that (A.114)-(A.115) yield

$$|q'_1 - q'_2| < \epsilon + \sup_{z\in Z}|q_1(z) - q_2(z)|. \tag{A.116}$$

Since ϵ can be arbitrarily small, this entails that

$$|q'_1 - q'_2| \leq \sup_{z\in Z}|q_1(z) - q_2(z)| \tag{A.117}$$

Consequently, (A.110) is true. $\square$

A.5 Bibliographic Notes

An in-depth treatment of the topic of contraction mapping can be found in Kolmogorov and Fomin [1957], Rudin [1964], and Moore [1985]. A discussion on the use of contraction mapping in the solution of dynamic programming functional equations can be found in Denardo [1968].

B

Fractional Programming

B.1 Introduction

Fractional programming, (referred to in *Chapter 12*), is a branch of nonlinear optimization which concerns itself with optimization problems involving ratio functions. Of interest to us here is the following family of problems:

$$\textbf{Problem F}: \qquad c := \max_{z \in Z} r(z) := \frac{v(z)}{w(z)} \tag{B.1}$$

where v and w are real-valued functions on some set Z, and $w(z) > 0$, $\forall z \in Z$. Let Z^* denote the set of optimal solutions to this problem. We assume that the set Z^* is not empty. □

Now, fractional programming proposes several methods to tackle problems of this type. Of these, the most common is the *parametric method.* Its thesis is that an optimal solution to *Problem F* can be obtained via the solution of a parametric problem of the following type:

$$\textbf{Problem F}(\lambda): \quad a(\lambda) := \max_{z \in Z} r_\lambda(z) := v(z) - \lambda w(z) \ , \ \lambda \in \mathbb{R} \tag{B.2}$$

Let $Z^*(\lambda)$ denote the set of optimal solutions to *Problem $F(\lambda)$*. It is assumed that *Problem $F(\lambda)$* has at least one optimal solution for each $\lambda \in \mathbb{R}$. □

The justification for seeking the solution of *Problem F* via *Problem $F(\lambda)$* is in the fact that a $\lambda \in \mathbb{R}$ exists such that every optimal solution to the latter is also optimal for the former. That this is indeed the case is shown by:

B.1.1 Theorem

$$z \in Z^* \iff z \in Z^*(r(z)) \tag{B.3}$$

Proof. Part 1. $y \in Z^* \Longrightarrow y \in Z^*(r(y))$.

Let y be any element of Z^*. Then, by definition $r(y) = c$, and

$$c = \frac{v(y)}{w(y)} \geq \frac{v(z)}{w(z)}, \forall z \in Z \tag{B.4}$$

Since w is positive on Z, multiplying (B.4) by $w(z)$ yields

$$v(z) - cw(z) \leq 0 \ , \ \forall z \in Z \tag{B.5}$$

On the other hand, $r(y) = c$ entails that

$$v(y) - cw(y) = 0 \tag{B.6}$$

Therefore,

$$v(y) - r(y)w(y) \geq v(z) - r(y)w(z), \forall z \in Z \tag{B.7}$$

which in turn implies that $y \in Z^*(r(y))$.

Part 2. $y \in Z^*(r(y)) \Rightarrow y \in Z^*$.
Let y be any element of Z such that $y \in Z^*(r(y))$. Then, by definition (B.7) holds. Also, since $r(y) = v(y)/w(y)$, the left-hand side of (B.7) is equal to zero. Thus, (B.7) entails that

$$0 \geq v(z) - r(y)w(z), \forall z \in Z \tag{B.8}$$

Now, since $w(z) > 0$, for all $z \in Z$, dividing (B.8) by $w(z)$ yields

$$r(y) \geq \frac{v(z)}{w(z)}, \forall z \in Z \tag{B.9}$$

This implies in turn that $y \in Z^*$. □

The following is an immediate implication of *Theorem B.1.1*:

B.1.2 Corollary

The equation $a(\lambda) = 0$ has a unique solution $\lambda \in \mathbb{R}$; specifically, $a(\lambda) = 0$ if, and only if, $\lambda = c$. □

To solve the equation $a(\lambda) = 0$, one would normally use Dinkelbach's Algorithm or one of its many offspring.

B.2 Dinkelbach's Algorithm

The following simple procedure is the major technique used to solve fractional programming problems.

```
Step 1: Select some $z \in Z$ and set $k = 1$, $z^{(1)} = z$, and $\lambda^{(1)} = r(z)$.
Step 2: Solve Problem $F(\lambda^{(k)})$ and select some $z \in Z^*(\lambda^{(k)})$.
Step 3: If $a(\lambda^{(k)}) = 0$, set $z' = z$ and $\lambda' = r(z)$, and stop.
        Otherwise, set $z^{(k+1)} = z$ and $\lambda^{(k+1)} = r(z)$.
Step 4: Set $k = k + 1$ and go to Step 2.                              □
```

The algorithm's central feature can be summed up as follows:

B.2.1 Theorem

Either Dinkelbach's Algorithm terminates, in which case $\lambda' = c$ and $z' \in Z$, or else the sequence $\{\lambda^{(k)}\}$ that it generates converges superlinearly to c. Termination is assured if Z is finite. □

We shall prove this in stages. As a prelude consider the following observation which derives by inspection from the definition of *Problem* $F(\lambda)$, that is, from (B.2).

B.2.2 Lemma

1. *$a(\lambda) > 0$ if, and only if, $\lambda < c$.*
2. *$a(\lambda) < 0$ if, and only if, $\lambda > c$.*
3. *$a(\lambda)$ is strictly decreasing with λ (note that $w(z) > 0$, $\forall z \in Z$).*
4. *$a(\lambda)$ is continuous and convex on $\mathbb{R}$.* □

Turning now to the proof. By construction, the algorithm terminates if, and only if, $a(\lambda^{(k)}) = 0$. Consequently, *Corollary B.1.2* implies the following:

B.2.3 Corollary

If Dinkelbach's Algorithm terminates then $\lambda' = c$ and $z' \in Z^*$. □

Having established this, I shall now show that the sequence $\{\lambda^{(k)}\}$ generated by the algorithm is strictly increasing. For this purpose let (z, y) be any pair such that $z \in Z$, $y \in Z^*(r(z))$ and $r(z) \neq r(y)$. I need to show that $r(y) > r(z)$. Observe then that by definition $y \in Z^*(r(z))$ implies that

$$v(y) - r(z)w(y) \geq v(z) - r(z)w(z) \tag{B.10}$$

Since $r(z) = v(z)/w(z)$, the right-hand side of (B.10) is equal to zero, hence

$$v(y) - r(z)w(y) \geq 0 \tag{B.11}$$

This implies in turn that $r(y) \geq r(z)$. Therefore, since $r(y) \neq r(z)$, if follows from (B.11) that $r(y) > r(z)$. Hence,

B.2.4 Lemma

The sequence $\{\lambda^{(k)}\}$ generated by Dinkelbach's Algorithm is strictly increasing, namely $\lambda^{(k+1)} > \lambda^{(k)}$ for all k. □

Consider then the case where Z is finite. Granted *Lemma B.2.4,* there must exist a finite k for which Dinkelbach's Algorithm generates a $\lambda^{(k)}$ such that $r(y) = r(z^{(k)})$ for all $y \in Z^*(r(z^{(k)}))$. This entails that $a(\lambda^{(k)}) = 0$, from which it follows that the algorithm terminates. Hence,

B.2.5 Lemma

If Z is finite, then Dinkelbach's Algorithm terminates, $\lambda' = c$ and $z' \in Z^*$. □

To complete the picture it is necessary to account for the case where the algorithm fails to terminate. Here, it must be shown that the sequence $\{\lambda^{(k)}\}$ converges to c. To this end it is sufficient to show that this sequence converges to a λ'' such that $a(\lambda\text{"}) = 0$, because *Corollary B.1.2* would then imply that $\lambda'' = c$. But to do this we need support from the following result.

B.2.6 Lemma

The sequence $\{w(z^{(k)}) : k > 1\}$ generated by Dinkelbach's Algorithm is nonincreasing, namely $w(z^{(k+1)}) \le w(z^{(k)})$ for all $k > 1$, and it converges to some $w' > 0$.

Proof. Let (λ, μ, z, y) be any quadruplet such that $(\lambda, \mu) \in \mathbb{R}^2$, $\lambda > \mu$, $z \in Z^*(\lambda)$ and $y \in Z^*(\mu)$. I shall first show that $w(z) \le w(y)$. This will enable to demonstrate that the sequence $\{w(z^{(k)})\}$ is nonincreasing. Observe then that by definition, $z \in Z^*(\lambda)$ and $y \in Z^*(\mu)$ imply that

$$v(z) - \lambda w(z) \ge v(y) - \lambda w(y) \tag{B.12}$$

and

$$v(y) - \mu w(y) \ge v(z) - \mu w(z) \tag{B.13}$$

respectively. Thus, adding these inequalities yields

$$(\lambda - \mu)(w(y) - w(z)) \ge 0 \tag{B.14}$$

As $\lambda > \mu$, it follows that $w(z) \le w(y)$.

Also, observe that by construction, $z^{(k)} \in Z^*(\lambda^{(k-1)})$ for all $k > 1$. Having established that $\{\lambda^{(k)}\}$ is strictly increasing, we may conclude that for any $k > 1$ we have $\lambda^{(k)} > \lambda^{(k-1)}$, $z^{(k)} \in Z^*(\lambda^{(k-1)})$, and $z^{(k+1)} \in Z^*(\lambda^{(k)})$. This implies that the sequence $\{w(z^{(k)}) : k > 1\}$ is nonincreasing. Thus, as w is strictly positive on Z and $w(y) \le w(z^{(k)})$ for all $y \in Z^*$ and $k \ge 1$, it follows that the limit of $\{w(z^{(k)})\}$ is strictly positive. □

The above results lead to the following conclusion:

B.2.7 Lemma

Should Dinkelbach's Algorithm fail to terminate, the sequence $\{\lambda^{(k)}\}$ *will converge monotonically to c.*

Proof. Recall that *Lemma B.2.4* provided that the sequence $\{\lambda^{(k)}\}$ is strictly increasing. Since by construction $\lambda^{(k)} = r(z^{(k)})$ and $z^{(k)} \in Z$, it follows that $\lambda^{(k)} \le c$, for all k. Hence, $\{\lambda^{(k)}\}$ must converge to some $\lambda" \le c$. I shall now show that $a(\lambda") = 0$, then invoking *Corollary B.1.2* we have $\lambda" = c$. Observe then that by definition,

$$a(\lambda^{(k)}) = \max_{z\in Z}\{v(z) - \lambda^{(k)}w(z)\} \tag{B.15}$$

and by construction $\lambda^{(k)} = r(z^{(k)}) = v(z^{(k)})/w(z^{(k)})$ and $z^{(k+1)} \in Z^*(\lambda^{(k)})$.
Hence,

$$a(\lambda^{(k)}) = v(z^{(k+1)}) - \lambda^{(k)}w(z^{(k+1)}) \tag{B.16}$$

Since $w(z) > 0$, $\forall z \in Z$, it follows that

$$a(\lambda^{(k)}) = \left[\frac{v(z^{(k+1)})}{w(z^{(k+1)}} - \lambda^{(k)}\right] w(z^{(k+1)}) \tag{B.17}$$

$$= (\lambda^{(k+1)}\lambda^{(k)})w(z^{(k+1)}) \tag{B.18}$$

Next, because $\{w(z^{(k)})\}$ is nonincreasing and $w(z) > 0$, $\forall z \in Z$, it follows that the former converges to some $w" > 0$. Moreover, as $\{\lambda^{(k)}\}$ also converges, it follows from (B.18) that the sequence $\{a(\lambda^{(k)})\}$ converges to 0.

Thus, given that *Lemma B.2.2* assures that $a(\lambda)$ is continuous with λ and that *Corollary B.1.2* provides that $a(\lambda) = 0$ if, and only if, $\lambda = c$, the conclusion is that $\{\lambda^{(k)}\}$ converges to c. □

As for the algorithm's rate of convergence, we have the following:

B.2.8 Lemma

If Dinkelbach's Algorithm fails to terminate, then the sequence $\{\lambda^{(k)}\}$ *converges superlinearly to c, that is,*

$$\lim_{k\to\infty}\left|\frac{c - \lambda^{(k+1)}}{c - \lambda^{(k)}}\right| = 0 \tag{B.19}$$

Proof. Let y be any element of Z^*, and let z be any element of $Z^*(\lambda^{(k)})$ for some $k > 1$. Then, by definition, $z \in Z^*(\lambda^{(k)})$ implies that

$$v(y) - \lambda^{(k)}w(y) \le v(z) - \lambda^{(k)}w(z) \tag{B.20}$$

Since $w(z) > 0$, and by construction $\lambda^{(k+1)} = r(z) = v(z)/w(z)$, dividing (B.19) by $w(z)$ yields

$$\frac{v(y)}{w(z)} - \lambda^{(k)}\frac{w(y)}{w(z)} \le \lambda^{(k+1)} - \lambda^{(k)} \tag{B.21}$$

Next, adding c to both sides of (B.21) and rearranging the terms yields

$$c - \lambda^{(k+1)} \leq c - \lambda^{(k)} + \lambda^{(k)} \frac{w(y)}{w(z)} - \frac{v(y)}{w(z)} \tag{B.22}$$

$$= c - \lambda^{(k)} + \frac{w(y)}{w(z)} \left[\lambda^{(k)} - \frac{v(y)}{w(y)} \right] \tag{B.23}$$

$$= c - \lambda^{(k)} + \frac{w(y)}{w(z)} [\lambda^{(k)} - c] \tag{B.24}$$

recalling that $y \in Z^*$ implies that $v(y)/w(y)$ is equal to c. The inference from (B.24) is that

$$c - \lambda^{(k)} \leq \left[c - \lambda^{(k)} \right] \left[1 - \frac{w(y)}{w(z)} \right] \tag{B.25}$$

Now, because $c > \lambda^{(k)}$, dividing (B.25) by $c\lambda^{(k)}$ yields

$$\left[\frac{c - \lambda^{(k+1)}}{c - \lambda^{(k)}} \leq 1 - \frac{w(y)}{w(z)} \right] \tag{B.26}$$

Observe that $c > \lambda^{(k)}$, $\forall k \geq 1$ and that (B.26) holds for any $y \in Z^*$, $k \geq 1$ and $z \in Z^*(\lambda^{(k)})$. Hence,

$$\left| \frac{c - \lambda^{(k+1)}}{c - \lambda^{(k)}} \right| \leq b(y, k) \ , \ \forall k \geq 1, y \in Z^*, z^{(k)} \in Z^*(\lambda^{(k)}) \tag{B.27}$$

where

$$b(y, k) := 1 - \frac{w(y)}{w(z^{(k)})} \tag{B.28}$$

To round out the proof I must still show that there exists a $y \in Z^*$ such that $\{b(y, k)\}$ converges to zero, or equivalently that $\{w(z^{(k)})\}$ converges to $w(y)$. Let then w' denote the limit of $\{w(z^{(k)}\}$, keeping in mind that *Lemma B.2.6* provides that $w' > 0$.

Since $\{\lambda^{(k)}\}$ converges to c and $\lambda^{(k)} = v(z^{(k-1)})/w(z^{(k-1)})$, it follows that there must be a $y \in Z^*$ such that $w(y) = w'$. □

As a final note I wish to call attention to two points. The first has to do with the parametric method's scope of operation, the second with the style of exposition that I used to set it out here.

First, the merit of the parametric method is in its enabling the solution of recalcitrant problems — problems falling under *Problem F* — through the solution of problems that generally prove far easier to solve — problems falling under *Problem F*(λ). Obviously, the underlying assumption here is that, to be able to use this method, the particular instance of *Problem F*(λ) under consideration must be amenable to at least one of the available optimization techniques.N Second, my presentation of the parametric method in this appendix is in line with the accepted treatment thereof in the literature. In *Appendix C,* however, I give this method a thoroughly different phrasing, basing it on a different theoretical foundation. To be precise, parting ways with the standard approach, in *Appendix C* I explain the parametric method in a manner that shows it to be a typical classical optimization method.

B.3 Bibliographic Notes

More on fractional programming's parametric method can be found in Dinkelbach [1967], Schaible [1976], Craven [1977, 1988], Ibaraki [1983], and Schaible and Ibaraki [1983], Sniedovich [1987a, 1987b, 1989].

C

Composite Concave Programming

C.1 Introduction

In this appendix I sum up the essentials of the nonlinear parametric optimization method that I entitled *composite concave programming* (c-programming for short). This method — referred to in *Chapter 12* — is designed to solve problems of the following form:

Target Problem

$$\textit{Problem } T: \qquad \tau := \max_{z\in Z} \ r(z) := \Phi(u(z)) \tag{C.1}$$

where u is a function on some set Z with values in $\mathbb{R}^k$, and Φ is a real-valued function on $u(Z) := \{u(z) : z \in Z\}$. Let Z^* denote the set of optimal solutions to this problem. We assume that Z^* is not empty.

Problems of this type often prove extremely difficult to solve because in many cases they defy the solution techniques of standard optimization methods. So, the tactic that c-programming proposes is to obtain their solution by solving a (surrogate) parametric problem derived from *Problem T* by a linearization of Φ with respect to u. That is, a problem of the following form:

Parametric Problem, $\lambda \in \mathbb{R}^k$:

$$\textit{Problem } T(\lambda): \quad t(\lambda) := \max_{z\in Z} \ r_\lambda(z) := \lambda^t u(z) := \sum_{m=1}^{k} \lambda_m u_m(z) \tag{C.2}$$

where $u_m(z)$ and λ_m denote the $m-th$ component of $u(z)$ and λ respectively, and λ^t denotes the transpose of λ.

Let $Z^*(\lambda)$ denote the set of optimal solutions to Problem $T(\lambda)$. We shall assume that for each $\lambda \in \mathbb{R}^k$ the set $Z^*(\lambda)$ consists of at least one element.

The rationale for seeking the solution of *Problem T* through the parametric problem is simple. Contrary to the target problem, the parametric problem is often readily amenable to standard optimization methods.

To illustrate what kind of problems are subsumed by *Problem T*, consider the following.

C.1.1 Example

Consider the following nonlinear, discrete optimization problem:

$$\begin{aligned} \max_{(z_1,\dots,z_N)} & \sum_{n=1}^{N} a_n z_n + \alpha \left[\sum_{n=1}^{N} b_n z_n\right]^2 \ , \ \alpha > 0 \\ & \sum_{n=1}^{N} c_n z_n \le C \\ & z_n \in \{0,1,2,\dots,C\} \end{aligned}$$

To comply with the format of *Problem T*, set $k = 2$

$$\begin{aligned} u_1(z_1,\dots,z_N) &= \sum_{n=1}^{N} a_n z_n \\ u_2(z_1,\dots,z_N) &= \sum_{n=1}^{N} b_n z_n \\ \Phi(u(z)) &= u_1(z) + \alpha u_2^2(z), z \in Z \ \square \end{aligned}$$

C.1.2 Example

Consider the case where $k = 2$ and

$$\Phi(u(z)) := \frac{u_1(z)}{u_2(z)}, u_2(z) > 0 \ , \ \forall z \in Z \tag{C.3}$$

This, of course, is the family of *fractional programming* problems analyzed in *Appendix B.* □

The discussion is confined to a subclass of *Problem T* whose members satisfy the following conditions.

C.1.3 Regularity conditions

1. There exists an open convex set $U \subseteq \mathbb{R}^k$ such that $u(Z) \subseteq U$.
2. The function Φ is differentiable on U with respect to u.
3. The gradient of Φ with respect to u is bounded on U. □

Let $\nabla\Phi(u(z))$ denote the gradient of Φ with respect to u at $u(z)$; and to keep the notation simple, define

$$d(z) := \nabla\Phi(u(z)) := \nabla\Phi(b)\,\big|_{b=u(z)} \tag{C.4}$$

namely, let $d(z)$ denote the gradient of Φ with respect to u at $b = u(z)$.

Observe that fractional programming problems are characterized by

$$U = \mathbb{R} \times \mathbb{R}^+ \ , \ \mathbb{R}^+ := \{a \in \mathbb{R} : a > 0\} \tag{C.5}$$

and

$$\nabla\Phi(u(z)) = \left(\frac{1}{u_2(z)} \ , \ -\frac{u_1(z)}{u_2^2(z)}\right) \ , \ z \in Z \tag{C.6}$$

The merit of *c*-programming is in the link that it establishes between *Problem T* and *Problem T*(λ) and in the algorithms that it supplies. That is, *c*-programming identifies the conditions under which a $\lambda \in \mathbb{R}^k$ exists such that $Z^*(\lambda) \subseteq Z^*$. By identifying these conditions *c*-programming provides the theoretical justification for seeking the solution for *Problem T* through the solution of *Problem T*(λ). But what is more, by furnishing algorithms capable of recovering such a λ it enables the solution of *Problem T*.

C.2 Preliminary Analysis

The first point that needs to be clarified is on what grounds is *Problem T* sought to be linked to *Problem T*(λ). To do this we need to review those properties of *generalized* convex functions that bear on this question.

So, let (a, b) be an element of U^2. Then, subject to the following generalized convexity and concavity condition, Φ has these properties:

$$\begin{array}{ll} Quasiconvexity: & \Phi(a) \leq \Phi(b) \Longrightarrow (a-b)^t\nabla\Phi(b) \leq 0 \qquad (C.7)\\ Pseudoconvexity: & \Phi(a) < \Phi(b) \Longrightarrow (a-b)^t\nabla\Phi(b) < 0 \qquad (C.8)\\ Quasiconcavity: & \Phi(a) \geq \Phi(b) \Longrightarrow (a-b)^t\nabla\Phi(b) \geq 0 \qquad (C.9)\\ Pseudoconcavity: & \Phi(a) > \Phi(b) \Longrightarrow (a-b)^t\nabla\Phi(b) > 0 \qquad (C.10) \end{array}$$

A function is said to be *pseudolinear* if it is both pseudoconvex and pseudoconcave.

In view of our assumption that Φ is differentiable on U, it must also be pointed out that (Avriel [1976], Bazaraa and Shetty [1979]):

C.2.1 Lemma

$$differentiability + \begin{array}{lcl} Pseudoconvexity & \Longrightarrow & Quasiconvexity \\ Pseudoconcavity & \Longrightarrow & Quasiconcavity \end{array} \tag{C.11}$$

So, an examination of the implications of (C.7)-(C.10) for the relation between *Problem T* and *Problem T*(λ) reveals the formal ties between the two problems.

Let (y, z) be an element of Z^2. Since by definition, $r(z) = \Phi(u(z))$, $d(z) = \nabla\Phi(u(z))$ and $r_\lambda(z) = \lambda^t u(z)$, it follows from (C.7)-(C.10) that:

$$Quasiconvexity: \quad r(y) \leq r(z) \Rightarrow r_{d(z)}(y) \leq r_{d(z)}(z) \tag{C.12}$$

$$Pseudoconvexity: \quad r(y) < r(z) \Rightarrow r_{d(z)}(y) < r_{d(z)}(z) \tag{C.13}$$

$$Quasiconcavity: \quad r(y) \geq r(z) \Rightarrow r_{d(z)}(y) \geq r_{d(z)}(z) \tag{C.14}$$

$$Pseudoconcavity: \quad r(y) > r(z) \Rightarrow r_{d(z)}(y) > r_{d(z)}(z) \tag{C.15}$$

The immediate implications of the above are in the relations between the set Z^* and the sets $\{Z^*(d(z)) : z \in Z^*\}$. These of course, furnish the theoretical foundation for c-programming algorithms. Of particular importance are the following:

C.2.2 Lemma

If Φ is quasiconvex on U, then

$$z \in Z^* \Longrightarrow z \in Z^*(\nabla\Phi(u(z))) \tag{C.16}$$

and

$$\{z \in Z, z \notin Z^*(\nabla\Phi(u(z))), y \in Z^*(\nabla\Phi(u(z)))\} \Longrightarrow r(y) > r(z) \tag{C.17}$$

C.2.3 Lemma

If Φ is pseudoconvex on U, then

$$Z^*(\nabla\Phi(u(z))) \subset Z^* \ , \ \forall z \in Z^* \tag{C.18}$$

C.2.4 Lemma

If Φ is pseudoconcave on U, then

$$z \in Z^*(\nabla\Phi(u(z))) \Longrightarrow z \in Z^* \tag{C.19}$$

and

$$z \in Z^*(\nabla\Phi(u(z))) \Longrightarrow Z^* \subseteq Z^*(\nabla\Phi(u(z))) \tag{C.20}$$

Since these results follow directly from (C.12)-(C.15) and their respective contrapositive relations, it will suffice to set out the proof of *Lemma C.2.2.*

Proof of Lemma C.2.2.
By definition, $z \in Z^*$ implies that $r(y) \leq r(z)$, $\forall y \in Z$. Hence, (C.16) is an immediate consequence of (C.12). Next, let (z, y) be any element of Z^2 satisfying the premise in (C.17). Because the contrapositive of (C.12) implies that

$$r_{d(z)}(y) > r_{d(z)}(z) \Longrightarrow r(y) > r(z) \tag{C.21}$$

and since any pair satisfying the premise in (C.17) also satisfies the premise in (C.21), it follows that $r(y) > r(z)$. $\square$

Finally, because Φ is assumed to be differentiable on U, the foregoing results entail the following:

C.2.5 Theorem

If Φ is pseudolinear on U, then

$$z \in Z^* \Longleftrightarrow z \in Z^*(\nabla\Phi(u(z))) \tag{C.22}$$

and furthermore

$$Z^* = Z^*(\nabla\Phi(u(z))) \ , \ \forall z \in Z^* \tag{C.23}$$

A point particularly worth noting is the affinity of (C.22) to the classical first-order necessary and sufficient condition governing the maximization of differentiable concave functions. Note that (C.22) in fact means that $z^* \in Z$ is an optimal solution to

$$\max_{z \in Z} \Phi(u(z)) \tag{C.24}$$

if, and only if, it constitutes an optimal solution to

$$\max_{z \in Z} \ [\nabla\Phi(u(z^*))]^t u(z) \tag{C.25}$$

An interesting consequence of this is that because the function

$$\Phi(b) := \frac{b_1}{b_2}, b_1 \in \mathbb{R} \ , \ b_2 \in \mathbb{R}^+ \tag{C.26}$$

is pseudolinear on $\mathbb{R} \times \mathbb{R}^+$, *Theorem C.2.5* holds for fractional programming problems.

To see that this is so I need to show that Φ defined by (C.26) is indeed pseudolinear. Observe then that (C.26) implies that

$$\nabla\Phi(b) = \left(\frac{1}{b_2} \ , \ -\frac{b_1}{b_2^2}\right) \ , \ b \in \mathbb{R} \times \mathbb{R}^+ \tag{C.27}$$

so that

$$(a-b)^t \nabla\Phi(b) = (a-b)^t \left(\frac{1}{b_2}\ ,\ -\frac{b_1}{b_2^2}\right) \tag{C.28}$$

$$= (a_1 - b_1, a_2 - b_2)^t \left(\frac{1}{b_2}\ ,\ -\frac{b_1}{b_2^2}\right)$$

$$= \frac{a_1 - b_1}{b_2} + \frac{-b_1(a_2 - b_2)}{b_2^2}$$

$$= \frac{b_2(a_1 - b_1) - b_1(a_2 - b_2)}{b_2^2} = \frac{b_2 a_1 - b_1 a_2}{b_2^2} = \frac{a_2}{b_2}(\frac{a_1}{a_2} - \frac{b_1}{b_2})$$

$$= \frac{a_2}{b_2}(\Phi(a) - \Phi(b)) \tag{C.29}$$

Because b_2 and a_2 are strictly positive it follows from (C.29) that (C.8) and (C.10) are true. This entails that Φ is pseudolinear.

The upshot of the foregoing analysis is then that, in cases where Φ is pseudoconvex, the set

$$\Gamma := \{\nabla\Phi(u(z)) : z \in Z^*\} \tag{C.30}$$

contains a λ such that any optimal solution to *Problem* $T(\lambda)$ is also optimal for *Problem* T. If Φ is pseudolinear, *any* element of Γ has this property.

My next task will be to profile the algorithms that would be capable of identifying such $\lambda's$. I shall do this in the framework of specific sub-cases of *Problem* T, obtained by attributing specific properties to the function Φ.

C.3 Pseudolinear Problems

This title defines a subclass of *Problem* T which consists of problems characterized by Φ being pseudolinear on U. An interesting implication of this grouping is that fractional programming problems are in fact pseudolinear c-programming problems.

Now, *Theorem C.2.5,* as we know, provides the assurance that if $\lambda \in \Gamma$ then the optimal solutions of *Problem* $T(\lambda)$ are also optimal for *Problem* T. The following algorithm is designed to recover such a λ in the case where Φ is pseudolinear.

C.3.1 Algorithm

Step 1: Select an element z from Z and set $\lambda^{(1)} = \nabla\Phi(u(z))$, $z^{(1)} = z$ and $m = 1$.

Step 2: Solve Problem $T(\lambda^{(m)})$ and select an element z from $Z^*(\lambda^{(m)})$.
Step 3: If $z^{(m)} \in Z^*(\lambda^{(m)})$ set $z' = z^{(m)}$ and $\lambda' = \lambda^{(m)}$ and stop. Otherwise, go to Step 4.
Step 4: Set $m = m+1$, $z^{(m)} = z$ and $\lambda^{(m)} = \nabla\Phi(u(z))$, and go to Step 2.□

Several observations ought to be made about this algorithm. First, if the algorithm terminates at the $m-th$ iteration then by construction $z^{(m)} \in Z^*(\lambda^{(m)})$. Thus, since by construction $\lambda^{(m)} = \nabla\Phi(u(z^{(m)}))$, *Theorem C.3.5* implies the following.

C.3.2 Corollary

Assume that Φ is pseudoconcave on U. If Algorithm C.3.1 terminates then $z' \in Z^*$ and $Z^* \subset Z^*(\lambda')$. □

Second, by construction, the sequence $\{z^{(m)}\}$ generated by *Algorithm C.3.1* has the property that $z^{(m+1)} \in Z^*(\nabla\Phi(u(z^{(m)})))$ for all $m \geq 1$. Hence, the following is an immediate implication of (C.17).

C.3.3 Corollary

Assume that Φ is quasiconvex on U. Then, the sequence $\{r(z^{(m)})\}$ generated by Algorithm C.3.1 is strictly increasing. □

Third, granted that Φ is differentiable and pseudolinear, it is both pseudoconcave and quasiconvex. Therefore, *Corollaries C.3.2-3* yield the following.

C.3.4 Corollary

Assume that Φ is pseudolinear and that the set $\{u(z) : z \in Z\}$ is finite. Then Algorithm C.3.1 terminates, $z' \in Z^*$ and $Z^* = Z^*(\lambda')$. □

However, what if *Algorithm C.3.5* does not terminate? To provide for such a situation consider the following.

C.3.5 Lemma

Assume that Φ is pseudolinear on U, that $\nabla\Phi$ is continuous on U, and that the sequence $\{u(z^{(m)}\}$ generated by Algorithm C.3.1 converges to some $u'' \in U$. Then, $\Phi(u'') = \tau$ and $Z^* \subset Z^*(\nabla\Phi(u''))$.

Proof. Since Φ and $\nabla\Phi$ are continuous on U and $\{u^{(m)}\}$ converges to $u'' \in U$, it follows that $\{\Phi(u(z^{(m)}))\}$ converges to $\Phi(u'')$ and $\{\lambda^{(m)}\}$ converges to $\lambda'' = \nabla\Phi(u'')$. Furthermore, as (C.2) implies that $t(\lambda)$ is continuous on $\mathbb{R}^k$, the sequence $\{t(\lambda^{(m)})\}$ must converge to $t(\lambda'')$. Also, considering that by construction $z^{(m+1)}$ is an element of $Z^*(\lambda^{(m)})$, it follows that

$$t(\lambda^{(m)}) = (\lambda^{(m)})^t u(z^{(m+1)}) \qquad \text{(C.31)}$$

Next, since $\{\lambda^{(m)}\}$ converges to λ'' and $\{u(z^{(m)})\}$ converges to u'', it follows that

$$t(\lambda'') = (\lambda'')^t u'' \tag{C.32}$$

Thus, (C.2) in conjunction with (C.32) entail that

$$[u(z) - u'']^t \lambda'' \leq 0 \ , \ \forall z \in Z \tag{C.33}$$

Now, since Φ is pseudoconcave, it follows from (C.33) that

$$\Phi(u(z)) \leq \Phi(u'') \ , \ \forall z \in Z \tag{C.34}$$

so that (C.1) in conjunction with (C.34) yield

$$\tau \leq \Phi(u''). \tag{C.35}$$

On the other hand, because (C.1) implies that $\tau \geq \Phi(u(z))$, $\forall z \in Z$ and since $\{\Phi(u(z^{(m)}))\}$ converges to $\Phi(u'')$,it follows that

$$\tau \geq \Phi(u'') \tag{C.36}$$

Thus, (C.35)-(C.36) imply that $\tau = \Phi(u'')$, thereby entailing that

$$\Phi(u(z)) = \Phi(u'') \ , \ \forall z \in Z^* \tag{C.37}$$

Finally, as Φ is differentiable and pseudoconcave, it is also quasiconcave. It therefore follows from (C.37) that

$$[u(z) - u'']^t \nabla\Phi(u'') \geq 0 \ , \ \forall z \in Z^* \tag{C.38}$$

Thus, since by construction $\lambda'' = \nabla\Phi(u'')$, (C.38) implies that $Z^* \subset Z^*(\lambda'')$. $\square$

Note that pseudolinearity entails that

$$\Phi(a) = \Phi(b) \iff (a-b)^t \nabla\Phi(b) = 0 \tag{C.39}$$

for any pair $(a, b) \in U^2$. Therefore, because (C.33), in conjunction with (C.38), imply that if Φ is pseudoconcave then

$$[u(z) - u'']^t \nabla\Phi(u'') = 0 \ , \ \forall z \in Z^* \tag{C.40}$$

the following is an immediate consequence of *Lemma C.3.5* and (C.40).

C.3.6 Lemma

Assume that Φ is pseudolinear on U, that $\nabla\Phi$ is continuous on U and that the sequence $\{u(z^{(m)})\}$ generated by Algorithm C.3.1 converges to some $u'' \in U$. Then $\Phi(u'') = \tau$ and $Z^* = Z^*(\lambda'')$, where $\lambda'' = \nabla\Phi(u'')$. $\square$

It should be pointed out that fractional programming problems satisfy the conditions assumed by *Lemma C.3.6.*

C.4 Convex Problems

Under this heading are grouped instances of *Problem T* with the characteristic that Φ is convex. Now, as convexity entails quasiconvexity and convexity in conjunction with differentiability entail pseudoconvexity, the foregoing analysis provides the following result:

C.4.1 Lemma

If Φ is convex on U then the sequence $\{r(z^{(m)})\}$ generated by Algorithm C.3.1 is strictly increasing. The algorithm terminates if the set $\{u(z) : z \in Z\}$ is finite; furthermore, the sequence $\{r(z^{(m)})\}$ converges to some $r'' \leq \tau$.
$\square$

The point is, however, that there is no assurance that the solution generated by the algorithm for a problem of this subclass is indeed optimal. This is due to the fact that under convexity $z \in Z^*(\nabla\Phi(u(z)))$ does not necessarily imply that $z \in Z^*$. The following example illustrates this point.

C.4.2 Example

Consider the problem

$$\tau := \max_{z \in Z} \ (z-2)^2 \ , \ Z := [0,3]$$

In line with the (C.1), set $k = 1$, $u(z) := (z2)$, $U = \mathbb{R}$, and $\Phi(b) := b^2$. Clearly, Φ is convex and differentiable on U. By construction, the affiliated parametric problem is of the form

$$t(\lambda) := \max_{z \in Z} \lambda(z-2) \ , \ \lambda \in \mathbb{R}$$

and therefore, by inspection,

$$Z^*(\lambda) := \begin{cases} \{3\} & , \quad \lambda > 0 \\ Z & , \quad \lambda = 0 \\ \{0\} & , \quad \lambda < 0 \end{cases}$$

Consider now the feasible solution $z^{(1)} = 3$. Since $\nabla\Phi(u(z)) = 2(z2)$, it follows that $\lambda^{(1)} = \nabla\Phi(u(z^{(1)})) = 2$, so that $z^{(1)} \in Z^*(\lambda^{(1)})$. That is, Algorithm C.3.1 terminates after the first iteration with $z' = 3$ and $\lambda' = 2$. Notice, however, that the (unique) optimal solution for the problem in question is $z = 0$, meaning that the algorithm produced a non-optimal solution for the problem considered. $\square$

The measures taken by c-programming to overcome this difficulty is to bolster *Algorithm C.3.1* by incorporating in it standard Lagrangian techniques in combination with what I call *coverage functions.*

C.4.3 Definition

Let (λ, z) be a pair such that $\lambda \in \mathbb{R}^k$ and $z \in Z^*(\lambda)$ and let A be a subset of $\mathbb{R}^k$. Then, (λ, z) is said to COVER *A if*

$$(y \in Z,\ \nabla\Phi(u(y)) \in A) \Longrightarrow \Phi(u(z)) \geq \Phi(u(y)) \tag{C.41}$$

Let $C(\lambda, z)$ denote the subset of $\mathbb{R}^k$ that is covered by (λ, z) and define:

$$C(\{(\lambda^{(m)}, z^{(m)}) : 1 \leq m \leq M\}) := \bigcup_{m=1}^{M} C(\lambda^{(m)}, z^{(m)}) \tag{C.42}$$

In accordance with (C.42), a sequence $Q = \{(\lambda^{(m)}, z^{(m)}) : 1 \leq m \leq M\}$ such that $\lambda^{(m)} \in \mathbb{R}^k$ and $z^{(m)} \in Z^*(\lambda^{(m)})$ for all $1 \leq m \leq M$ is said to cover A if every element of A is covered by at least one element of Q. I shall refer to the function C as a coverage function. □

The following is an obvious implication of (C.41)

C.4.4 Lemma

Let $Q = \{(\lambda^{(m)}, z^{(m)}) : 1 \leq m \leq M\}$ be a sequence such that $\lambda^{(m)} \in \mathbb{R}^k$ and $z^{(m)} \in Z^*(\lambda^{(m)})$ for all $1 \leq m \leq M$, and define,

$$Z^\circ(Q) := \left\{z^{(n)} : 1 \leq n \leq M, \Phi(u(z^{(n)})) = \max\left\{\Phi(u(z^{(m)})) : 1 \leq m \leq M\right\}\right\} \tag{C.43}$$

If Q covers the set $\Gamma := \{\nabla\Phi(u(z)) : z \in Z\}$ then $Z^\circ(Q) \subset Z^*$.

Proof. Let $Q = \{(\lambda^{(m)}, z^{(m)}) : 1 \leq m \leq M\}$ be any sequence satisfying the premise of *Lemma C.4.4* and let y be any element of Z^*. I shall show that there must be some $1 \leq m \leq M$ such that $\Phi(u(z^{(m)})) = \Phi(u(y))$, from which it will follow that $z^{(m)} \in Z^*$ and consequently that $Z^\circ(Q) \subset Z^*$.

Observe then that because Q covers Γ and $\nabla\Phi(u(y)) \in \Gamma$, it follows that there must exist some $1 \leq m \leq M$ such that $(\lambda^{(m)}, z^{(m)})$ covers $\{\nabla\Phi(u(y))\}$. Thus, (C.41) implies that $\Phi(u(z^{(m)})) \geq \Phi(u(y))$. Hence, since $z^{(m)} \in Z$ and $y \in Z^*$, it follows that $z^{(m)} \in Z^*$. □

I call attention to the fact that no convexity conditions are premised by *Lemma C.4.4*. Such conditions will, however, have to be imposed on Φ to enable identifying **effective** coverage functions. In any case, *Lemma C.4.4* suggests the following:

C.4.5 Algorithm

Step 1: Construct a set $V \subset R^k$ such that $\nabla\Phi(u(z)) \in V$, $\forall z \in Z$.

Step 2: Select an element λ from V, solve Problem $T(\lambda)$, select an element z from the set $Z^*(\lambda)$ and set $m = 1$, $z' = z$, $\tau' = \Phi(u(z))$, and $V^{(m)} = C(\lambda, z)$.

Step 3: If $V \subset V^{(m)}$, stop. Otherwise go to Step 4.
Step 4: Select an element $\lambda \in V$ such that $\lambda \notin V^{(m)}$.
Step 5: Solve Problem $T(\lambda)$ and select an element z from the set $Z^*(\lambda)$. If $\Phi(u(z)) > \tau'$ set $z' = z$ and $\tau' = \Phi(u(z))$.
Step 6: Set $V^{(m+1)} = V^{(m)} \cup C(\lambda, z)$ and $m = m + 1$, and go to Step 3. $\square$

Note that this algorithm terminates if, and only if, $V \subset V^{(m)}$. Thus, the following is an immediate consequence of *Lemma C.4.4.*

C.4.6 Theorem

Assume that C is a coverage function and that Algorithm C.4.5 terminates. Then, $z' \in Z^*$. $\square$

Two questions require attention.

· What sort of coverage functions can be devised for convex problems?
· How powerful can these functions be?

Let us consider the first question first.

C.4.7 Theorem

If the function Φ is convex on U then the function C defined by

$$C(\lambda, z) := \{\alpha\lambda + (1-\alpha)\nabla\Phi(u(z)) : 0 < \alpha \leq 1\} \tag{C.44}$$

is a coverage function.

Proof. Since Φ is differentiable and convex on U, it follows that

$$\Phi(u(z)) \geq \Phi(u(y)) + [\nabla\Phi(u(y))]^t[u(z) - u(y)] \tag{C.45}$$

and

$$[\nabla\Phi(u(y)) - \nabla\Phi(u(z))]^t[u(z) - u(y)] \leq 0 \tag{C.46}$$

for any pair $(z, y) \in Z^2$ (see Bazaraa and Shetty 1979, *p.* 91).

Now, contrary to the theorem, assume that C is not a coverage function. In this case there must be a triplet (λ, z, y) satisfying the premise in (C.41) and the condition $\Phi(u(y)) > \Phi(u(z))$, thereby implying that (C.45) entails that

$$(\nabla\Phi(u(y))^t[u(z) - u(y)] < 0 \tag{C.47}$$

Multiplying (C.47) by α and (C.46) by $1-\alpha$ and adding the resulting inequalities yield

$$[\alpha\nabla\Phi(u(y)) - (1-\alpha)\nabla\Phi(u(z))]^t[u(z) - u(y)] < 0 \tag{C.48}$$

However, because (C.44) entails that $\nabla\Phi(u(y)) = \alpha\lambda + (1-\alpha)\nabla\Phi(u(z))$ for some $0 < \alpha \leq 1$, it follows that

$$\nabla\Phi(u(y)) - (1-\alpha)\nabla\Phi(u(z)) = \alpha\lambda \tag{C.49}$$

for this α, so that (C.48)-(C.49) imply that

$$\alpha\lambda^t[u(z) - u(y)] < 0 \tag{C.50}$$

Thus, since $\alpha > 0$, it follows that $\lambda^t u(y) > \lambda^t u(z)$ which is in contradiction to the contention that $z \in Z^*(\lambda)$. □

The coverage function specified by (C.44) produces the line segment connecting the points λ and $\nabla\Phi(u(z))$. The implication is then that this function is unlikely to provide effective coverage in cases where $k > 2$. This finding brings us to the second question about these coverage functions, namely what efficacy can such functions have. As we shall shortly see it is indeed possible to formulate far more potent coverage functions by attributing Φ certain separability properties.

C.4.8 Theorem

Assume that Φ admits the following representation

$$\Phi(u(z)) = \sum_{n=1}^{k} \Phi_n(u_n(z)) \tag{C.51}$$

where for each n, $1 \leq n \leq k$, the function Φ_n is convex and differentiable with respect to u_n on some open convex set $U_n \subset \mathbb{R}^k$ such that $u_n(z) \in U_n$, $\forall z \in Z$. Then, the function C defined by

$$C(\lambda, z) := \{\mu \in \mathbb{R}^k : \mu_n = \alpha_n\lambda_n + (1-\alpha)\nabla\Phi_n(u_n(z)),\\ 0 < \alpha_n \leq 1, 1 \leq n \leq k\} \tag{C.52}$$

is a coverage function.

Proof. Let (λ, z, y) be any triplet complying with the premise in (C.41). We need to show that $\Phi(u(z)) \geq \Phi(u(y))$. Observe then that because $\nabla\Phi(u(y)) \in C(\lambda, z)$, it follows from (C.52) that there exists some vector $\alpha \in \mathbb{R}^k$ such that $0 < \alpha_n \leq 1$, $1 \leq n \leq k$, and

$$\nabla\Phi_n(u_n(y)) = \alpha_n\lambda_n + (1-\alpha_n)\nabla\Phi_n(u_n(z)), \forall 1 \leq n \leq k \tag{C.53}$$

Since the functions $\{\Phi_n\}$ are convex and differentiable, it follows that

$$[\nabla\Phi_n(u_n(z)) - \nabla\Phi_n(u_n(y))][u_n(z) - u_n(y)] \geq 0, \forall 1 \leq n \leq k \tag{C.54}$$

Thus, (C.53)-(C.54) yield

$$[\lambda_n - \nabla\Phi_n(u_n(y))][u_n(z) - u_n(y)] \leq 0, \forall 1 \leq n \leq k$$

which in turn implies that

$$\lambda^t[u(z) - u(y)] \leq (\nabla\Phi(u(y))^t[u(z) - u(y)] \tag{C.55}$$

Thus, as $z \in Z^*(\lambda)$, it follows that $\lambda^t u(z) \geq \lambda^t u(y)$ thereby implying that the left-hand side of (C.55) is non-negative, and needless to say, that the right-hand side of (C.55) is non-negative as well. Hence, since Φ is convex and differentiable, it follows from (C.45) that $\Phi(u(z)) \geq \Phi(u(y))$. □

A function Φ admitting the representation given in (C.51) is said to be *convex and additive.*

It should be noted that $C(\lambda, z)$ defined by (C.52) yields the k -dimensional rectangle specified by the vertices λ and $\nabla\Phi(u(z))$ — with the exclusion of the faces connected to $\nabla\Phi_n(u_n(z))$ if $\alpha_n = 1$. Clearly, this coverage function can be counted on to perform effectively even in cases where $k > 2$.

Another point I wish to make about the coverage functions used in conjunction with *Algorithm C.4.5* is that they are Lagrangian in nature. For the purposes of this discussion it will suffice to consider the following.

C.4.9 Lemma

$$\{\lambda, \lambda' \in \mathbb{R}^k, z \in Z^*(\lambda), z \in Z^*(\lambda')\} \Longrightarrow z \in Z^*(\mu), \forall\mu \in [\lambda, \lambda'] \tag{C.56}$$

Proof. By definition, $z \in Z^*(\mu)$ implies that

$$\mu^t u(z) \geq \mu^t u(y), \forall y \in Z \tag{C.57}$$

so that $z \in Z^*(\lambda)$ implies that

$$\lambda^t u(z) \geq \lambda^t u(y) \ , \ \forall y \in Z \tag{C.58}$$

and that $z \in Z^*(\lambda')$ implies that

$$(\lambda')^t u(z) \geq (\lambda')^t u(y) \ , \ \forall y \in Z \tag{C.59}$$

Thus, multiplying (C.58) by $0 \leq \alpha \leq 1$ and (C.59) by $1 - \alpha$, and adding the resulting inequalities yields

$$[\alpha\lambda + (1-\alpha)\lambda']^t u(z) \geq [\alpha\lambda + (1-\alpha)\lambda']^t u(y) \ , \ \forall y \in Z \tag{C.60}$$

This means that

$$z \in Z^*(\mu), \forall\mu \in \{\alpha\lambda + (1-\alpha)\lambda' : 0 \leq \alpha \leq 1\} \ \Box \tag{C.61}$$

C.4.10 Lemma

Let $(\lambda, \lambda', z, z')$ be any quadruplet such that $z \in Z^*(\lambda)$, $z' \in Z^*(\lambda')$, $z \notin Z^*(\lambda')$ and $z' \notin Z^*(\lambda)$ and set

$$\lambda'' = \alpha\lambda + (1-\alpha)\lambda' \tag{C.62}$$

where

$$\alpha = \frac{\lambda'^t[u(z') - u(z)]}{(\lambda - \lambda')^t[u(z) - u(z')]} \tag{C.63}$$

If either $z \in Z^*(\lambda'')$ and/or $z' \in Z^*(\lambda'')$, then

$$z \in Z^*(\mu), \forall \mu \in \{\beta\lambda + (1-\beta)\lambda'' : 0 \leq \beta \leq 1\} \tag{C.64}$$

and

$$z' \in Z^*(\mu), \forall \mu \in \{\beta\lambda' + (1-\beta)\lambda'' : 0 \leq \beta \leq 1\} \tag{C.65}$$

Proof. Assume that the quadruplet $(\lambda, \lambda', z, z')$ satisfies the premise in *Lemma C* 4.10. Now, since $z \in Z^*(\lambda)$, $z' \in Z^*(\lambda')$, $z \notin Z^*(\lambda')$ and $z' \notin Z^*(\lambda)$, it follows that

$$\lambda^t u(z) > \lambda^t u(z') \tag{C.66}$$

and

$$(\lambda')^t u(z') > \lambda'^t u(z) \tag{C.67}$$

from which it follows that

$$0 < (\lambda')^t[u(z') - u(z)] < (\lambda - \lambda')^t[u(z) - u(z')] \tag{C.68}$$

which in turn implies that α defined by (C.63) satisfies the condition $0 < \alpha < 1$. Now, to prove the lemma I must show that if either $z \in Z^*(\lambda'')$ *and/or* $z' \in Z^*(\lambda'')$ then (C.64) and (C.65) hold.

Multiplying (C.63) by its denominator yields

$$\alpha(\lambda - \lambda')^t[u(z) - u(z')] = \lambda'^t[u(z') - u(z)] \tag{C.69}$$

from which it follows that

$$\alpha(\lambda - \lambda')^t[u(z) - u(z')] + \lambda^t[u(z) - u(z')] = 0 \tag{C.70}$$

and consequently that

$$[\alpha\lambda + (1-\alpha)\lambda']^t[u(z) - u(z')] = 0. \tag{C.71}$$

Thus, (C.71), in conjunction with (C.62), implies that

$$(\lambda'')^t u(z) = (\lambda'')^t u(z') \tag{C.72}$$

This in turn entails that

$$z \in Z^*(\lambda'') \Longleftrightarrow z' \in Z^*(\lambda'') \tag{C.73}$$

Finally, since $z \in Z^*(\lambda)$ and $z' \in Z^*(\lambda')$, *Lemma C.4.9,* in conjunction with (C.73), imply that (C.64)-(C.65) hold. □

I now consider a special case of the convex additive type

C.5 One-Dimensional Convex Additive Problems

This subcase comprises problems for which $k = 2$ and

$$r(z) := \Phi(u(z)) := u_1(z) + \varphi(u_2(z)) \tag{C.74}$$

where φ is a differentiable convex function on some open convex set containing $\{u_2(z) : z \in Z\}$. In compliance with the general convex additive format given by (C.51) we can set $\Phi_1(u_1(z)) = u_1(z)$ and $\Phi_2(u_2(z)) = \varphi(u_2(z))$, $\forall z \in Z$.

Observe that the partial derivative of Φ_1 with respect to u_1 at $u_1(z)$, $z \in Z$, is equal to 1 for all $z \in Z$. Hence, the search for $\lambda \in \Gamma$ can be restricted to the second component of λ, namely we can set $\lambda_1 = 1$ at the outset. In short, the parametric problem would be of the following form:

$$\textit{Problem } T(\lambda): \qquad t(\lambda) := \max_{z\in Z}\ r_\lambda(z) := u_1(z) + \lambda u_2(z)\ ,\ \lambda \in \mathbb{R} \tag{C.75}$$

The great attraction of this sub-case is that the parameter λ is a *scalar*. Consequently, *Algorithm C.4.5* reduces here to a *line search.* Before outlining the algorithm designed for this case, I note that in the case under consideration *Lemma C.4.10* makes the following statement:

C.5.1 Lemma

Assume that $(\lambda, \lambda', z, z')$ is a quadruplet such that $z \in Z^*(\lambda)$ and $z' \in Z^*(\lambda')$, where λ and λ' are elements of $\mathbb{R}$(Note that $Z^*(\mu)$ denotes the set of optimal solutions to (C.75) for $\lambda = \mu$.) Also, assume that $u_2(z) \neq u_2(z')$, and define

$$\lambda'' = \frac{u_1(z') - u_1(z)}{u_2(z) - u_2(z')} \tag{C.76}$$

If either $z \in Z^*(\lambda'')$and/or $z' \in Z^*(\lambda'')$ then

$$z \in Z^*(\mu), \forall \mu \in \{\alpha\lambda + (1-\alpha)\lambda'' : 0 \leq \alpha \leq 1\} \tag{C.77}$$

and

$$z' \in Z^*(\mu),\ \forall \mu \in \{\alpha\lambda' + (1-\alpha)\lambda" : 0 \leq \alpha \leq 1\} \tag{C.78}$$

By the same token, *Lemma C.4.9* can be restated as follows:

C.5.2 Lemma

$$\{\lambda, \lambda' \in \mathbb{R}, \lambda \leq \lambda', z \in Z^*(\lambda), z \in Z^*(\lambda')\} \Rightarrow z \in Z^*(\mu)\ ,\ \forall \mu \in [\lambda, \lambda'] \tag{C.79}$$

Finally, *Theorem C.4.8* can be formulated in this case as follows:

C.5.3 Lemma

If Φ admits the representation specified by (C.74) and φ is convex, then the function C defined by

$$C(\lambda, z) := \{\alpha\lambda + (1-\alpha)\nabla\varphi(u(z)) : 0 < \alpha \leq 1\}\ ,\ \lambda \in \mathbb{R},\ z \in Z \quad \text{(C.80)}$$

is a coverage function.

So the algorithm that one would use to track down a λ such that $Z^*(\lambda) \subset Z^*$ will employ (C.76) as a device with which to bisect the subintervals of $\mathbb{R}$. The algorithm will always seek out the lowest subinterval for this purpose.

C.5.4 Algorithm

Comment: The coverage function used in Step 4 is that specified by (C.80).

Step 1: *Find a pair of numbers (λ_l, λ_u) such that*

$$\lambda_l \leq \nabla\varphi(u(z)) \leq \lambda_u,\ \forall z \in Z \quad \text{(C.81)}$$

and set $\Gamma^{(1)} = (\lambda_l, \lambda_u)$.

Step 2: *Solve Problem $T(\lambda)$ for $\lambda = \lambda_l$ and $\lambda = \lambda_u$ and set,*

$$\tau' = \max\{r(z_l), r(z_u)\} \quad \text{(C.82)}$$

$$z' = \begin{cases} z_l & , \quad \tau' = r(z_1) \\ z_u & , \quad otherwise \end{cases} \quad \text{(C.83)}$$

$$Z^{(1)} = (z_l, z_u) \quad \text{(C.84)}$$

where z_l and z_u are the solutions found for Problem $T(\lambda_l)$ and Problem $T(\lambda_u)$ respectively. Set $m = 1$.

Step 3: *Stop if the sequence $\Gamma^{(m)}$ is a singleton. Otherwise go to Step 4.*

Step 4: *If $\{(\lambda_l, z_l), (\lambda_u, z_u)\}$ covers the interval $[\lambda_l, \lambda_u]$, go to Step 8.*

Step 5: *Solve Problem $T(\lambda)$ for*

$$\lambda = \frac{u_1(z_l) - u_1(z_u)}{u_2(z_u) - u_2(z_l)} \quad \text{(C.85)}$$

and select some $z \in Z^*(\lambda)$. If $r(z) > \tau'$, set $z' = z$ and $\tau' = r(z)$.

Step 6: *If either $z_l \in Z^*(\lambda)$ and/or $z_u \in Z^*(\lambda)$, go to Step 8. Otherwise go to Step 7.*

Step 7: *Set*

$$\lambda_u = \lambda,\ z_u = z \quad \text{(C.86)}$$

$$\Gamma^{(m+1)} = (\lambda_u, \Gamma^{(m)}),\ Z^{(m+1)} = (z_u, Z^{(m)}) \quad \text{(C.87)}$$

$m = m + 1$, and go to Step 4.

Step 8: *Set*

$$\lambda_l = \lambda_u, \; z_l = z_u \tag{C.88}$$

$$\Gamma^{(m+1)} = 1 \downarrow \Gamma^{(m)}, \; Z^{(m+1)} = 1 \downarrow Z^{(m)} \tag{C.89}$$

$$\lambda_u = 1 \uparrow \Gamma^{(m+1)}, \; z_u = 1 \uparrow Z^{(m+1)} \tag{C.90}$$

$m = m + 1$, *and go to Step 3.*

Note: The operations denoted by the symbols $\downarrow$ and $\uparrow$ are defined as follows. Let S be a non-empty sequence. Then $1 \uparrow S$ designates the first element of S and $1 \downarrow S$ the remainder of S after its first element is dropped. □

Let us now investigate the behavior of this algorithm. For this purpose it will be convenient to slightly modify it as follows:

C.5.5 Algorithm

The same as Algorithm C.5.4., except that Step 4 is defined thus:

Step 4: *If either* $z_l \in Z^*(\lambda_u)$*and/or* $z_u \in Z^*(\lambda_l)$, *go to Step 8.* □

Note that if the condition tested in Step 4 of *Algorithm C.5.5* is satisfied, then on the strength of *Lemma C.5.1* either $z_l \in Z^*(\lambda)$ for all $\lambda \in [\lambda_l, \lambda_u]$ or $z_u \in Z^*(\lambda)$ for all $\lambda \in [\lambda_l, \lambda_u]$. In either case, the interval $[\lambda_l, \lambda_u]$ is covered by the pair $\{(\lambda_l, z_l), (\lambda_u, z_u)\}$, recalling that the coverage function used is the one defined by (C.80). This means that if *Algorithm C.5.5* terminates, *Algorithm C.5.4* must terminate as well, possibly after fewer iterations. In either case, because the lowest subinterval is discarded only when it is covered, the following result holds.

C.5.6 Lemma

If Algorithm C.5.5 terminates, then $\tau' = \tau$ *and* $z' \in Z^*$. □

The question as to what kind of conditions assure that the algorithm terminates can be answered thus: Step 3 follows immediately one of these events: A *new* solution z was detected in Step 5 or an interval was discarded in Step 8. Since new intervals are created in Step 7 only by new solutions, the inference is that the algorithm must terminate if there are finitely many solutions. In other words, *Lemma C.5.2* entails that the same solution cannot enter more than once into the sequence $Z^{(m)}$. Thus,

C.5.7 Theorem

Assume that the derivative of φ *is bounded on* $\{u_2(z) : z \in Z\}$ *and that the set* Z *is finite. Then Algorithm C.5.5 and Algorithm C.5.4 terminate, each yielding* $\tau' = \tau$ *and* $z' \in Z^*$. □

As a final note, I wish to point out that it is in fact possible to relax somewhat the conditions stipulated by *Theorem C.5.7* by requiring that the set $\{z : z \in Z^*(\lambda),\ \lambda \in \mathbb{R}\}$ rather than the set Z be finite.

C.5.8 Bibliographic Notes

More on c-programming can be found in Sniedovich [1985a, 1986b, 1989], Byrne et al. [1998], Churilov et al. [1998] and Churilov et al [2004]. The relationship between c-programming and fractional programming is discussed in Sniedovich [1987b]. Numerical experiments with c-programmming/linear programming schemes are reported on in Macalalag and Sniedovich [1991].

D

The Principle of Optimality in Stochastic Processes

The objective of this appendix is to examine an argument against Bellman's *Principle of Optimality*, allegedly showing that the principle runs into trouble in the context of *stochastic processes with non-denumerable state spaces*. I show that the alleged difficulty does not arise if one's reading of the principle remains true to Bellman's understanding thereof.

I assume that the reader is familiar with the basic concepts and notation of conditional probability and conditional expectation.

D.1 Preliminary Analysis

To keep the notation simple, the analysis is conducted in the framework of a problem with an *additive* objective function. Assume then that the objective function g is additive, moreover that rather than being defined on $S \times \mathbb{D}^N$, as in the case of the deterministic model, it is now defined on $(S \times \mathbb{D} \times S)^N$. Specifically, consider the case where

$$g(s_1, x_1, s_2, x_2, \dots, s_N, x_N, s_{N+1}) := \sum_{n=1}^{N} w_n(s_n, x_n, s_{n+1}) \tag{D.1}$$

Consequently, the objective functions of the modified problems have this form:

$$g_n(s_n, x_n, s_{n+1}, x_{n+1}, \dots, s_N, x_N, s_{N+1}) := \sum_{m=n}^{N} w_m(s_m, x_m, s_{m+1}) \tag{D.2}$$

Also, assume that the state variables are **random variables** such that the probability distribution function of $\widetilde{s}_{n+1}$ is uniquely determined by the value of $\widetilde{s}_n$ and the value of the decision x_n made at stage n. Note that I abide by the standard distinction between the random variable ($\widetilde{s}_n$) and the value that it takes (s_n). Also for simplicity, it is assumed that the state variable takes values in some finite interval $S = [a, b], a > b$ of the real-line and that $p_{n,s,x}$ denotes the conditional probability *density* function of $\widetilde{s}_{n+1}$ given that $\widetilde{s}_n = s$ and $\widetilde{x}_n = x$, assuming, as we have been doing all along, that $x_n \in D(n, s)$. Thus,

$$Prob\,[u \leq \widetilde{s}_{n+1} \leq v \mid \widetilde{s}_n = s, \widetilde{x}_n = x] := \int\limits_u^v p_{n,s,x}(z)dz \tag{D.3}$$

is the probability that $\widetilde{s}_{n+1}$ will take a value in the interval $[u, v]$ given that $\widetilde{s}_n = s$, and $\widetilde{x}_n = x$. It follows from this that

$$Prob[\widetilde{s}_{n+1} = s' | \widetilde{s}_n = s, \widetilde{x}_n = x] = 0 \tag{D.4}$$

for all $1 \leq n \leq N, s \in S, x \in D(n, s), s' \in S$.

The value of the initial state $\widetilde{s}_1$ is assumed to be known with certainty, say, Prob$[\widetilde{s}_1 = s_1] = 1$, for some $s_1 \in S$ and the decisions are determined by a Markovian policy. This entails that the decisions themselves are random variables, namely $\widetilde{x}_1 = \delta(1, \widetilde{s}_1)$ and

$$\widetilde{x}_n = \delta(n, \widetilde{s}_n) \ , \ 2 \leq n \leq N \tag{D.5}$$

The objective is to find a Markovian policy that optimizes the *expected value* of the objective function g given the initial condition $\widetilde{s}_1 = s_1$. Formally, the problem can be formulated as follows:

$$\textit{Problem } SP(s_1): \quad f(s_1) := \sup_{\delta \in \Delta} E_\delta[g|s_1] \ , \ s_1 \in S \tag{D.6}$$

where $E_\delta[g|s_1]$ denotes the expected value of g that is generated by the process induced by the initial condition $\widetilde{s}_1 = s_1$ and the policy δ. Recall that Δ denotes the set of Markovian policies. Let $\Delta^*(s_1)$ denote the set of optimal policies for this problem.

As we know by now, dynamic programming would view this problem as being imbedded in a family of *modified problems* of the following form:

$$\textit{Problem } SP(n, s): \quad f_n(s) := \sup_{\delta \in \Delta} E_\delta[g_n|n, s] \ , \ 1 \leq n \leq N, s \in S \tag{D.7}$$

where $E_\delta[g_n|n, s]$ denotes the expected value of g_n generated by the process induced by the policy δ and the initial condition $\widetilde{s}_n = s$. Let $\Delta^*(n, s)$ denote the set of all the optimal policies for *Problem* $SP(n, s)$.

The set of *conditional problems* would now be defined as follows:

$$\textit{Problem } SP(n, s, x, s'), 1 \leq n \leq N, s \in S, x \in D(n, s), s' \in S:$$
$$f_n(s, x, s') := \sup_{\delta \in \Delta} E_\delta[g_n|n, s, x, s'] \tag{D.8}$$

where $E_\delta[g_n|n,s,x,s']$ denotes the expected value of g_n generated by the policy δ, given that $\widetilde{s}_n = s$, $\widetilde{x}_n = x$ and $\widetilde{s}_{n+1} = s'$. We shall refer to this problem as the *conditional problem at* (n,s,x,s'). Let $\Delta^*(n,s,x,s')$ denote the set of optimal policies for *Problem* $SP(n,s,x,s'$.

It is important to note that in our deterministic model it was unnecessary to incorporate the state s' resulting from (n,s,x) in the formulation of the conditional problems, as in this model the state resulting from (n,s,x) is uniquely determined by the triplet (n,s,x) through the transition function T, namely $s' = T(n,s,x)$.

Let us now examine how the principle is construed in the context of this model by the common interpretation and the consequences that ensue from this.

D.2 Common Interpretation of the Principle

The common interpretation understands the principle to be making – in the context of our deterministic model – the following statement:

> *If δ is optimal for Problem $SP(n,s)$ for some $1 \le n < N, s \in S$, then it must also be optimal for Problem $SP(n+1,s')$ for any state s' generated by $p_{n,s,x}, x = \delta(n,s)$.*

Now, once the principle is understood to make this assertion, its validity becomes problematic. The source of trouble here is the requirement that δ be optimal for *any* state $s' \in S$ generated by the density $p_{n,s,x}$, $x = \delta(n,s)$.

Let us examine why this is so. As indicated by (D.4), because the states are *continuous* random variables, the probability that $\widetilde{s}_{n+1}$ is equal to a predetermined $s' \in S$ — given that $\widetilde{s}_n = s$ and $x_n = \delta(n,s)$ — *is* equal to zero. So, as far as *Problem* $SP(n,s)$ is concerned, it is immaterial what particular decision the policy δ selects at $(n+1,s')$ for *a given* $s' \in S$, so long as $Prob[\widetilde{s}_{n+1} = s'|\widetilde{s}_n = s, \widetilde{x}_n = \delta(n,s)] = 0$. The immediate implication of this is that δ being optimal with respect to *Problem* $SP(n,s)$ does not necessarily entail that it is optimal with respect to *Problem* $SP(n+1,s')$ for all $s' \in S$. The following example illustrates this point.

D.2.1 Example

Consider the case where

$$\text{opt} = \max \tag{D.9}$$
$$N = 2 \tag{D.10}$$
$$s_1 = 0 \tag{D.11}$$
$$S = [0,1] \tag{D.12}$$

$$D(n,s) = \{0,1\}\ ,\ 1 \le n \le N, s \in S \tag{D.13}$$

$$w(n,s,x,s') = x\ ,\ 1 \le n \le N, s \in S, x \in D(n,s), s' \in S \tag{D.14}$$

$$p_{n,s,x}(s') = 1\ ,\ 1 \le n \le N, s \in S, x \in D(n,s), s' \in S \tag{D.15}$$

Observe that this means that the state variable is a continuous, uniform random variable on $[0,1]$.

In view of (D.14), it is clear that to maximize the expected value of the objective function, the best strategy would be to always choose the decision $x = 1$. Thus, it is obvious that the Markovian policy δ° specified by $\delta^\circ(n,s) = 1$, $1 \le n \le N$, $s \in S$, is optimal everywhere. Obviously, this policy accords with the principle in its common interpretation. Consider, however, the Markovian policy δ'' specified as follows:

$$\delta''(n,s) = \begin{cases} 0 & ,\quad n = 2, s = 0 \\ 1 & ,\quad \text{otherwise} \end{cases} \tag{D.16}$$

In other words, δ'' is identical to δ° except that at $(n = 2, s = 0)$ it selects the decision $x = 0$ rather than $x = 1$. Since the probability that $\widetilde{s}_2 = 0$ given $\widetilde{s}_1 = s_1$ and $x_1 = 1$ is equal to zero, it is clear that δ" is optimal with respect to *Problem* $SP(1,0)$, that is

$$E_{\delta''}[g_1|n = 1, \widetilde{s}_1 = 0] = E_{\delta^\circ}[g_1|n = 1, \widetilde{s}_1 = 0] = f_1(0) = 2$$

However, δ'' is not optimal for *Problem* $SP(2,0)$, as clearly

$$E_{\delta''}[g_2|n = 2, \widetilde{s}_2 = 0] = E_{\delta''}[w_2|n = 2, \widetilde{s}_2 = 0] = \delta''(2,0) = 0$$

whereas

$$f_2(0) = E_{\delta^\circ}[g_2|n = 2, \widetilde{s}_2 = 0] = \delta^\circ(2,0) = 1\ \square$$

So, to prevent this implication, it has been proposed that the phrasing of the principle be modified as follows:

> *If $\delta \in \Delta$ is an optimal policy for Problem $SP(n,s)$ for some $1 \le n < N$ and $s \in S$, then* WITH PROBABILITY ONE *δ is also optimal for Problem $SP(n+1,s')$, $s' \in S$.*

The clause "*with probability one*" is designed to imply the existence of a subset S' of S such that δ is optimal for *Problem* $SP(n+1,s')$, for any $s' \in S'$ and that

$$Prob[\widetilde{s}_{n+1} \in S'|\widetilde{s}_n = s, \widetilde{x}_n = \delta(n,s)] = 1 \tag{D.17}$$

And to be sure, this correction wards off the above illustrated difficulty.

My point is then that in my reading of the *Principle of Optimality*, no such correction is called for. Let us examine this point more carefully.

D.3 Principle of Optimality (stochastic models)

Keeping in mind my analysis of the principle's purport in the framework of a deterministic model (Section 13.2), it is clear that in the framework of a stochastic model the *Principle of Optimality* would read as follows:

D.3.1 Principle of Optimality (stochastic version)

If δ is an optimal policy with respect to Problem $SP(n,s,x,s')$ for some $1 \leq n < N$, $s \in S$, $x \in D(n,s)$ and $s' \in S$, then it must also be optimal for Problem $SP(n+1,s')$. In other words, with regard to stochastic problems, the principle contends the following:

$$\Delta(n,s,x,s') \subseteq \Delta(n+1,s') \tag{D.18}$$

for all $1 \leq n < N, s \in S, x \in D(n,s), s' \in S$. □

To prove the validity of this claim, observe that as the objective function is additive, it follows from (D.8) that

$$\begin{aligned} f_n(s,x,s') &= \sup_{\delta\in\Delta} E_\delta[g_n|n,s,x,s'] && \text{(D.19)}\\ &= \sup_{\delta\in\Delta} E_\delta[\{w_n(s,x,s') + g_{n+1}\}|n,s,x,s'] && \text{(D.20)}\\ &= \sup_{\delta\in\Delta}\{\hat{w}_n(s,x) + E_\delta[g_{n+1}|s,x,s']\} && \text{(D.21)} \end{aligned}$$

where $\hat{w}_n(s,x) := E[w_n(s,x,s')|s,x]$.

And since $\hat{w}_n(s,x)$ is independent of δ, the implication is that

$$f_n(s,x,s') = \hat{w}_n(s,x) + \sup_{\delta\in\Delta} E_\delta[g_{n+1}|s,x,s'] \tag{D.22}$$

Now, because the optimization problem specified by the second term on the right-hand side of (D.21) is identical to *Problem* $SP(n+1,s')$, it follows that *Problem* $P(n,s,x,s')$ and *Problem* $P(n+1,s')$ are equivalent in that both have the same optimal solutions. Thus,

$$\Delta^*(n,s,x,s') = \Delta^*(n+1,s') \tag{D.23}$$

for all $1 \leq n < N, s \in S, x \in D(n,s), s' \in S$, which clearly implies that (D.18) is true. In other words, the formulation obeys the *Principle of Optimality.*

It should also be noted that, as in the case of the deterministic model discussed in *Chapter 13,* the principle can be regarded as articulating a Markovian property of the optimal policies.

D.4 Conclusions

To sum up then, in contrast to the common interpretation of the principle, my interpretation requires no emendation — such as the added clause "*with probability one*" — to safeguard its validity in stochastic models involving non-denumerable state spaces.

D.5 Bibliographic Notes

Discussions of this issue can be found in Yakowitz [1969], Hinderer [1970], Porteus [1975], and Sniedovich [1978b].

A slightly modified version of this appendix appeared as a contribution entitled *Eureka! Bellman's Principle of Optimality Is Valid!* in a book dedicated to the memory of Sid Yakowitz (1937-1999), edited by Dror, L'Ecuyer and Szidarovszky [2002].

E

The Corridor Method

E.1 Introduction

The *Corridor Method* is a dynamic programming inspired meta-heuristic that is designed to reduce the size of the *solution space* of a combinatorial optimization problem under consideration. But, it can be implemented in conjunction with any combinatorial optimization method.

To explain how this method works in conjunction with dynamic programming, consider the following generic dynamic programming functional equation:

$$f(s) = \operatorname*{opt}_{x\in D(s)} \{w(s,x) \oplus f(T(s,x))\} \ , \ s \in S' \tag{E.1}$$

with $f(s) = L_{\oplus}, \forall s \in S''$, where $L_{\oplus}$ denotes the identity element of $\oplus$; S'' is the set of terminal states and S' denotes the set of non-terminal states.

In certain applications the state space S' can grow exceedingly large (*Curse of Dimensionality*) to thus impede the solution this equation.

The *Corridor Method* proposes to alleviate this difficulty by substituting the state $S = S' \cup S''$ space with a smaller set, $\widehat{S} = \widehat{S}' \cup S''$. This smaller set is obtained by constructing a "corridor" around the *state trajectory* that is generated by a feasible solution to the problem. This solution is referred to as the *incumbent solution.*

The end result is a dynamic programming functional equation of the form

$$\hat{f}(s) = \operatorname*{opt}_{x\in \widehat{D}(s)} \left\{w(s,x) \oplus \hat{f}(\widehat{T}(s,x))\right\} \ , \ s \in \widehat{S}' \tag{E.2}$$

where $\widehat{D}(s) \subset D(s), \forall s \in \widehat{S}'$.

The sets $\widehat{S}'$ and $\widehat{D}(s), s \in S'$, as well as and the transition function $\widehat{T}$, are determined on grounds of the incumbent solution.

Formally,

$$\hat{f}(s) := \underset{x_n,\dots,x_k}{\text{opt}} \{w(s_1,x_1) \oplus w(s_2,x_2) \oplus \cdots \oplus w(k,s_k,x_k)\} \ , \ s \in \widehat{S}' \quad \text{(E.3)}$$

$$x_j \in \widehat{D}(s_j) \ , \ j = 1,\dots,k \quad \text{(E.4)}$$

$$s_{j+1} = \widehat{T}(s_j,x_j) \ , \ j = 1,\dots,k \quad \text{(E.5)}$$

$$s_1 = s \quad \text{(E.6)}$$

$$s_{k+1} \in S'' \quad \text{(E.7)}$$

Since the set $\widehat{S}'$ is far smaller than the corresponding set S', obtaining a solution for (E.2) is of course far easier than obtaining a solution for (E.1). However, the price tag attached to this is that the solution obtained for (E.2) is not necessarily a "good" solution to (E.1). For this reason the functional equation (E.2) is solved repeatedly, in a manner that in each iteration, the "corridor" is constructed around the solution obtained in the previous iteration of the algorithm.

Details on the method and its applications can be found in Sniedovich and Voß[2006] and Caserta et al. [2008, 2010]. In this appendix I illustrate its working in the context of the traveling salesman problem.

E.2 Preliminaries

To give a clearer account of why the need for this method arises and of its main ingredients it would be best to explain it in a simple, abstract, dynamic-programming-free, framework. Consider then the optimization problem

$$\text{Problem } P(X): \qquad q^* := \underset{x \in X}{\text{opt}}\, q(x) \quad \text{(E.8)}$$

Suppose that X is a discrete set and that the problem is to be solved with some numerical method, call it $\mathfrak{M}$. The difficulty is that $\mathfrak{M}$ would be able to find a solution for this problem only for a relatively small X. The question is then: what to do in cases where X is extremely large?

The *Corridor Method* proposes to do as follows: Instead of solving (E.8), solve the following far easier problem:

$$\text{Problem } P(X(\mathbf{y})): \qquad q^*(\mathbf{y}) := \underset{x \in X(\mathbf{y})}{\text{opt}}\, q(x) \quad \text{(E.9)}$$

where $\mathbf{y} \in X$ is an incumbent solution to (E.8) and, $X(\mathbf{y})$ is a subset of X containing $\mathbf{y}$ which is much smaller than X. We refer to $\mathbf{y}$ as the *incumbent* and to $X(\mathbf{y})$ as a *corridor around* $\mathbf{y}$.

Let $\mathbf{z}$ denote the optimal solution generated for (E.9) and assume that $q(\mathbf{y}) \neq q(\mathbf{z})$, hence that $\mathbf{y}$ is not an optimal solution of (E.9). Using $\mathbf{z}$ as the new incumbent, create a new subset of X, call it $X(\mathbf{z})$, and solve the new problem

$$\text{Problem } P(X(\mathbf{z})): \qquad q^*(x(\mathbf{z}))) := \underset{x\in X(\mathbf{z})}{\text{opt}}\ q(x) \tag{E.10}$$

Continue in this fashion until a *fixed point* is reached: the optimal solution to the previous problem is optimal for the new problem.

At this point, either terminate the procedure, or continue it using a new "seed" $\mathbf{y}$.

Figure E.1 depicts the above process in the context of a problem where the objective is to find the optimal path from point A to point B. The shaded area represents the "corridors" around the "incumbent" solutions.

Assume that in the framework of *Problem* $P(X)$ there is a map $\mathbb{C}$ from X to the power set of X such that $\mathbb{C}(\mathbf{x})$ is a non-empty strict subset of X. We call $\mathbb{C}(\mathbf{x})$ the *corridor around* x.

The following schema illustrates how the *Corridor Method*, in conjunction with method $\mathfrak{M}$, solve *Problem* $P(X)$:

Procedure:

· **Initialization:** Set $j=1$ and select some $\mathbf{x}^{(j)} \in X$.

· **Iteration:** solve

$$\text{Problem } P(\mathbb{C}(\mathbf{x}^{(j)})): \quad \mathbf{y}^* = \arg \underset{\mathbf{y}\in\mathbb{C}(\mathbf{x}^{(j)})}{\text{opt}}\ q(\mathbf{y})$$

· **Termination:** Stop if $q(\mathbf{x}^{(j)}) = q(\mathbf{y}^*)$. Otherwise, set $j=j+1$, and $\mathbf{x}^{(j)} = \mathbf{y}^*$, and continue with the iteration.

Of course, the corridor $\mathbb{C}(\mathbf{x})$ around an incumbent solution $\mathbf{x}$ must be constructed so that a solution to *Problem* $P(\mathbb{C}(\mathbf{x}))$ would be easily found by method $\mathfrak{M}$.

To illustrate how the *Corridor Method* works in cases where $\mathfrak{M}$ = *Dynamic Programming* let us go back to dynamic programming's treatment of the traveling salesman problem (TSP).

E.3 Example: TSP

Consider the classic *Traveling Salesman Problem* involving k cities and let $C=\{1,2,\dots,k\}$ with 0 denoting the home city. As indicated in §*11.11.1,* this problem can be stated as follows:

$$\min_{x_1,\dots,x_k}\left\{d(0,x_1)+\sum_{j=1}^{k-1} d(x_j,x_{j+1})+d(x_k,0)\right\} \tag{E.11}$$

$$\{x_1,\dots,x_k\}=\{1,\dots,k\} \tag{E.12}$$

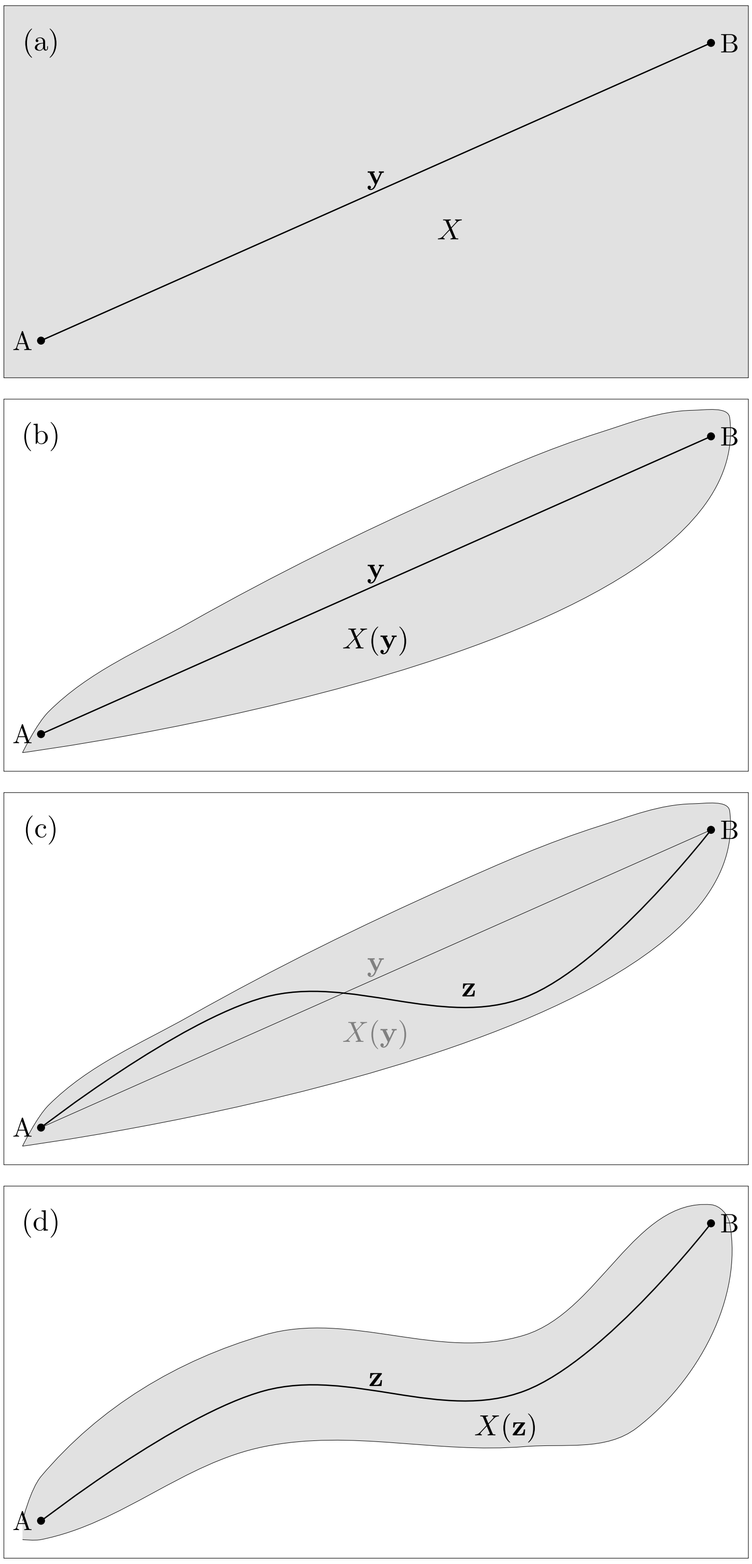


Figure E.1: Illustration of the Corridor Method

where $d(i,j)$ denotes the direct link from city i to city j and city 0 denotes the home city.

The decision variables are construed as follows:

$$x_j = \textit{the j-th city on the tour} \ , \ j = 1,2,\dots,k \tag{E.13}$$

The constraint indicates that the sequence of decisions $(x_1,\dots,x_k)$ must be a PERMUTATION of the cities. Note that implicit in this setup is an additional decision, namely $x_{k+1} = 0$, which implies that the tour ends with the transition from city x_k to the home city 0.

The dynamic programming model for this problem consists of the following elements:

$$\sigma = (0, C) \tag{E.14}$$
$$S'' = \{0, \varnothing\} \tag{E.15}$$
$$S' = \{\sigma\} \cup \{(c,V) : c \in C, V \subseteq C \setminus \{x\}\} \tag{E.16}$$
$$S = S' \cup S'' \tag{E.17}$$
$$D(c,V) = \begin{cases} V & , \ V \neq \varnothing \\ \{0\} & , \ V = \varnothing, c \in C \\ \varnothing & , \ V = \varnothing, c = 0 \end{cases} \tag{E.18}$$
$$T(c,V,x) = (x, V \setminus \{x\}) \tag{E.19}$$
$$w(c,V,x) = d(c,x) \tag{E.20}$$
$$\oplus = + \tag{E.21}$$

The corresponding family of modified problems is as follows:

$$f(c,V) = \min_{x_1,\dots,x_{m+1}} \sum_{j=1}^{m+1} w(s_j,x_j) \ , \ m = |V|, (c,V) \in S' \tag{E.22}$$
$$x_j \in D(s_j) \ , \ j = 1,2,\dots,m+1 \tag{E.23}$$
$$s_{j+1} = T(s_j,x_j) \ , \ j = 1,2,\dots,m+1 \tag{E.24}$$
$$s_1 = (c,V) \tag{E.25}$$
$$s_{m+2} = (0,\varnothing) \tag{E.26}$$

with $f(c,\varnothing) := d(c,0), \forall c \in C$.

Thus, by definition, $f(c,V)$ denotes the length of the shortest tour that commences at city c, visits each city in V exactly once, and ends in the home city. This yields the following dynamic programming functional equation:

$$f(c,V) = \min_{x \in V} \{d(c,x) + f(x, V \setminus \{x\})\} \ , \ (c,V) \in S' \tag{E.27}$$

with $f(c,\varnothing) = d(c,0), c \in C$. The objective is to find the value of $f(\sigma) = f(0,C)$.

Since for each $c \in C$ the set V for which $(c, V) \in S'$ can be any subset of $C \setminus \{c\}$, it follows that V can take 2^{k-1} distinct values for each $c \in C$. This means that $|S'| = 1 + k2^{k-1}$. Therefore, to solve the functional equation we have to determine the value of $f(s)$ for $|S'| = 1 + k2^{k-1}$ values of (c, V) pairs (states). But, as k grows larger, solving the equation becomes a practical impossibility (*Curse of Dimensionality*). $\square$

In the next section I illustrate how the corridors, $\mathbb{C}(\mathbf{x}), \mathbf{x} \in X$, are constructed for the dynamic programming model of the traveling salesman problem.

E.4 Generating Corridors

When it is applied in the context of a dynamic programming model, the term "corridor" designates the sets $\widehat{D}(s)$ and $\widehat{S}$ that are constructed "around" a state trajectory associated with a feasible solution to the problem under consideration. Constructing a "corridor" in actual fact means imposing certain *exogenous constraints* on the decision variables of the dynamic programming model with the view to reduce the number of feasible decisions available at any given state of the process. By design then the "incumbent" problem would be inside the "corridor".

So, making the decision at stage s subject to the constraint $x \in \widehat{D}(s)$, rather than to $x \in D(s)$, where $\widehat{D}(s) \subset D(s)$ has the effect of reducing the size of the state space, recalling that state trajectories are propagated by the transition function as follows:

$$S^{(n+1)}(\sigma) := \{T(s, x) : s \in S^{(n)}(\sigma), x \in D(s)\} \ , \ n = 1, 2, \ldots \tag{E.28}$$

where $S^{(0)}(\sigma) := \{\sigma\}$ and σ denotes the initial state. That is, $S^{(n)}$ denotes the set of states that can be generated by exactly n transitions, commencing at the initial state σ.

As for the question of what type of constraints are to be used in the context of a model considered, the answer is that there is no general formula to prescribe such a choice. These constraints would be chosen on a case by case basis. The overriding consideration in each case would be that the exogenous constraints be instrumental in decreasing the size of the state space, but not detrimental to the recovery of an optimal solution.

I now illustrate how this is done in the context of the traveling salesman problem.

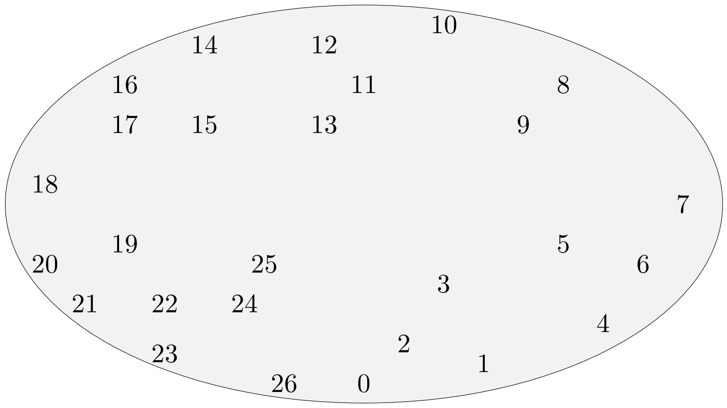


Figure E.2: The traveling salesman problem on a treasure island

E.4.1 Example

Figure E.2 displays a map of the locations[1] of 27 cities on a small island, where 0 is the home city. So let $k = 26$ and $C = \{1, 2, \dots, k\}$, and for simplicity assume that the intercity distances are Euclidean. Namely the road connecting any city to any other is perfectly "straight".

The layout of the cities is such that, by inspection an optimal tour will visit the cities either in a clockwise or a counter-clockwise direction, in an order that is not much different from the one stipulated by the labels of the cities. In fact, it is safe to say, that the optimal tour will not prescribe going from city 7 directly to city 18, or from city 10 directly to city 23.

So, for the initial incumbent let us consider the feasible tour that goes counter-clockwise through the cities, namely let $\mathbf{x}^{(1)} = (x_1^{(1)}, \dots, x_k^{(1)}, 0)$ specified by $x_j^{(1)} = j, j = 1, 2, \dots, 26$, be the initial feasible solution, recalling that $x_{k+1} = 0$ for all feasible tours. This solution (tour) is shown in Figure E.3.

The question is then: How do we create a "corridor" around $\mathbf{x}^{(1)}$?

Suppose that given an incumbent tour $\mathbf{y} = (y_1, \dots, y_k, 0)$, we impose the following exogenous constraints on the problem, where Δ and δ are (relatively small) positive integer:

- **C-1**: For each $j = \Delta + 1, \dots, k$, city y_j can be visited only *after* all the cities in $\{y_1, \dots, y_{j-\Delta-1}\}$ have been visited.
- **C-2**: For each $j = 1, \dots, k$, the tour can proceed directly from city y_j only to cities in $\{y_i : |i - j| \leq \delta, 1 \leq i \leq k, i \neq j\}$.

Then it would follow from **C-1** that in the next iteration of the *Corridor*

[1]The exact location of a city on the island is at the **exact** center of the label representing it on the map!

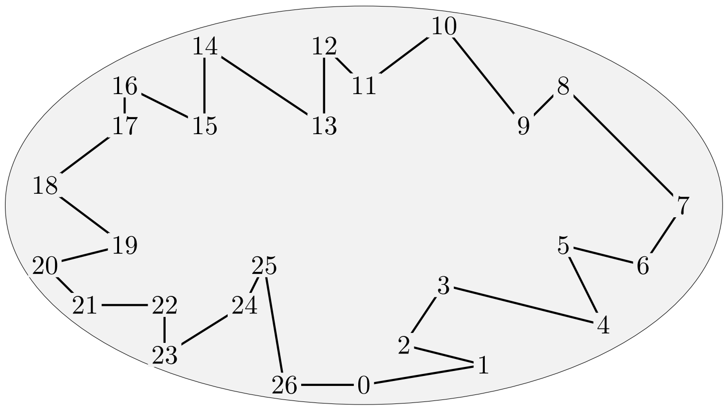


Figure E.3: Initial incumbent: $\mathbf{x}^{(1)} = (1, 2, \ldots, 26, 0)$

method, when the tour is at city y_j it has already visited, among other cities, all the cities in

$$A(y_j) := \{y_1, y_2, \ldots, y_{j-\Delta-1}\} \tag{E.29}$$

And by the same token, the cities not yet visited, are among others, the cities in

$$B(y_j) := \{y_{j+\Delta+1}, y_{j+\Delta+2}, \ldots, y_k\} \tag{E.30}$$

This means that, from a dynamic programming point of view, all that needs to be known about the history of the decision-making process, when the tour is in city y_j is what cities in

$$\mathcal{V}(y_j) : = \{y_{j-\Delta}, y_{j-\Delta+1}, \ldots, y_{j-2}, y_{j-1}, y_{j+1}, y_{j+2}, \ldots, y_{j+\Delta}\} \tag{E.31}$$

are yet to be visited.

For example, suppose that $\Delta = 3$ and that the tour is in city $x_j^{(1)} = 15$ on the incumbent tour $\mathbf{x}^{(1)}$ given above. Then, all that need be known is what cities in

$$\mathcal{V}(15) : = \{x_{15-3}^{(1)}, \ldots, x_{14}, x_{16}, \ldots x_{15+3}^{(1)}\} = \{12, 13, 14, 16, 17, 18\} \tag{E.32}$$

are yet to be visited. This is shown in Figure E.4.

The point is that **C-1** entails that — given the incumbent $\mathbf{x}^{(1)}$ — if the tour is at city 15 then it must have visited already all the cities in

$$A(15) = \{x_1^{(1)}, x_2^{(1)}, \ldots, x_{j-\Delta-1}^{(1)}\} \tag{E.33}$$

$$= \{x_1^{(1)}, x_2^{(1)}, \ldots, x_{11}^{(1)}\} = \{1, \ldots, 11\} \tag{E.34}$$

and it has yet to visit the cities in

$$B(15) = \{x_{15+\Delta+1}^{(1)}, \ldots, x_k^{(1)}\} = \{19, \ldots, 26\} \tag{E.35}$$

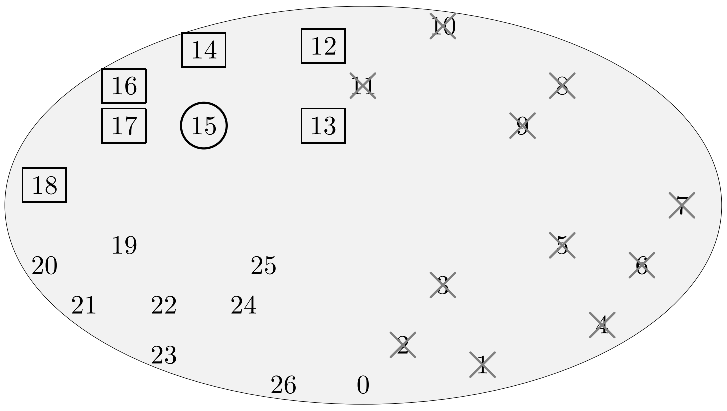


Figure E.4: $\mathcal{V}(15)$ for $\Delta = 3$

Hence, to have a full picture of the status of the tour say at city 15, we must know what cities in $\mathcal{V}(15)$ are yet to be visited, or what cities in $\mathcal{V}(15)$ have already been visited.

In short, a state in this case can be a pair (c, V) where c denotes the *current city* being visited and V denotes the subset of $\mathcal{V}(c)$ consisting of the cities yet to be visited — in addition to the cities in $B(c)$. Alternatively, V could be the set of $\mathcal{V}(c)$ consisting of the cities already visited — in addition to the cities in $A(c)$. The choice between these two alternatives is a matter of convenience. In this discussion we assume that

$$V = \textit{ subset of } \mathcal{V}(c) \textit{ consisting of the cities yet to be visited.} \tag{E.36}$$

Observe that $|\mathcal{V}(c)| = 2\Delta$, hence for each $c \in C$ there are at most $2^{2\Delta}$ relevant values for V. Therefore, altogether there are no more than $k2^{2\Delta}$ relevant states $(c, V), c \in C, V \subseteq V(c)$. It follows then that for small values of Δ the number of states in the corridor around the incumbent solution is much smaller than $|S'| = 1 + k2^{k-1}$. For instance, for $k = 26$ and $\Delta = 3$, we have $1 + 26 \times 2^{25} = 34,080,769 >>> 26 \times 2^6 = 1664$.

Moreover, it should be noted that not all the subsets of $\mathcal{V}(c)$ are actually feasible with respect to **C-1**, hence the figure $|\mathcal{V}(c)| = 2\Delta$ is an upper bound on the number of feasible value of V associated with a given c. For example, consider again the initial incumbent $\mathbf{x}^{(1)}$ and let $c = x_{15}^{(1)} = 15$. Obviously, $V = \{12, 13, 14, 16, 18\}$ is not a feasible companion for c because V entails that city 17 has already been visited. But this clearly violates **C-1**. Because, for $\Delta = 3$ visiting city 17 before city 12 is disallowed.

For any given value of c, the smallest element of V, call it $V_{\min}$, can take at most 2Δ values. For each value of $V_{\min}$ the largest element in V, call it $V_{\max}$, can take at most Δ values, hence $|V| \leq \Delta$. It follows then that there are at most $k(2\Delta)2^{\Delta}$ feasible values for the state $s = (c, V)$ described above. For $k = 26$ and $\Delta = 3$ this amounts to $26 \times 6 \times 8 = 1248$ states.

The exogenous constraint **C-2** can be used to further reduce the size of the state space. For example, if $\Delta = 3$ and $\delta = 2$, then $V = \{12, 13, 14, 16, 17, 18\}$ is not a feasible companion for $c = 15$ because ... it would be impossible to reach city 15 without violating **C-2**.

It follows then that subject to the two exogenous constraints under consideration, the set of decisions available at state (c, V) is as follows:

$$\widehat{D}(c,V) = \left\{ x \in V : \overset{\text{by C-2}}{|x - c| \leq \delta}, \ \overset{\text{by C-1}}{x < \Delta + V_{\min}} \right\} \tag{E.37}$$

Let us now examine the other elements of the dynamic programming model.

E.4.1.1 Dynamic programming model for the Corridor method

To formulate the dynamic programming model for the problem based on an incumbent tour $\mathbf{y}$, it is convenient to use the indices of $\mathbf{y}$ as decision variables. Thus, a tour is a sequence of indices $\mathbf{x} = (x_1, \dots, x_k, 0)$:

$$x_j := \text{ index of } y \text{ representing the j-th city on the tour } \mathbf{x}, j = 1, 2, \dots, k$$

For example, with $k = 5$, the sequence of decisions $\mathbf{x} = (4, 2, 3, 1, 5, 0)$ represents the tour $(y_4, y_2, y_3, y_1, y_5, 0)$: the tour begins at the home city (0) goes to city y_4, from there to y_2, from there to y_3, from there to y_1, from there to y_5 — and then back home.

With this in mind, consider the following dynamic programming model:

$$\hat{\sigma} = (0, \{1, 2, \dots, \Delta\}) \tag{E.38}$$

$$\widehat{S}' = \{\hat{\sigma}\} \cup \{(c, V) : c \in C, V \subseteq \mathcal{V}(c)\} \tag{E.39}$$

$$\widehat{S}'' = \{0, \varnothing\} \tag{E.40}$$

$$\widehat{D}(c, \varnothing) = \begin{cases} \{0\} & , \quad c > k - \min\{\Delta, \delta\} \\ \varnothing & , \quad c \leq k - \min\{\Delta, \delta\} \end{cases} \tag{E.41}$$

$$\widehat{D}(c, V) = \{x \in V : |x - c| \leq \delta, x < \Delta + V_{\min}\} \ , \ V \neq \varnothing \tag{E.42}$$

$$\widehat{T}(c, V, x) = \begin{cases} (x, (V \setminus \{x\}) \cup (\mathcal{V}(x) \setminus \mathcal{V}(c))) & , \quad x > c \\ (x, ((V \setminus \{x\}) \setminus (\mathcal{V}(c) \setminus \mathcal{V}(x))) & , \quad x < c \end{cases} \tag{E.43}$$

$$w(c, V, x) = d(c, x) \tag{E.44}$$

$$\oplus = + \tag{E.45}$$

recalling that $\min V$ denotes the smallest element in V. Observe that

$$\mathcal{V}(x) \setminus \mathcal{V}(c) = \{c + \Delta + 1, c + \Delta + 2, \dots, x + \Delta\} \ , \ x > c \tag{E.46}$$

$$\mathcal{V}(c) \setminus \mathcal{V}(x) = \{x + \Delta + 1, x + \Delta + 2, \dots, c + \Delta\} \ , \ x < c \tag{E.47}$$

The modification in the transition function is required due to the fact that V does not contain all the cities yet to be visited, but only those cities that are to be visited that are elements of $\mathcal{V}(c)$. So, if the tour proceeds from city c to city $x > c$, it is necessary to append to $V \setminus \{x\}$ the cities that are in $\mathcal{V}(x)$ but are not in $\mathcal{V}(c)$. For example, consider the incumbent solution $\mathbf{x}^{(1)}$ and the state (c, V) where $c = 15$ and $V = \{14, 16, 17, 18\}$, assuming that $\Delta = \delta = 3$. Now, suppose that we consider the decision $x = 18$, in which case the usual transition, namely $T(c, V, x) = (x, V \setminus \{x\})$ will yield the new state $(c', V') = (15, \{14, 16, 17\})$.

But this ignores the fact that if the tour is in city 18, then the cities that must be visited are in $\{19, 20, 21\}$, furthermore that these can be reached directly from city 18. Thus,

$$\begin{aligned}
\widehat{T}(c, V, x) &= \widehat{T}(15, \{14, 16, 17, 18\}, 18) && \text{(E.48)}\\
&= (\{14, 16, 17, 18\} \setminus \{18\}) \cup \{15 + 3 + 1, \dots, 18 + 3\} && \text{(E.49)}\\
&= \{14, 16, 17, 19, 20, 21\} && \text{(E.50)}
\end{aligned}$$

Similarly, to comply with the convention that $V \subset \mathcal{V}(x)$, if $x < c$ it is necessary to delete from $\mathcal{V}(c)$ the elements that extend beyond the range of $\mathcal{V}(x)$.

Remarks:

· Since the incumbent solution is in the corridor, the initial problem at $\sigma = (0, \{1, 2, \dots, \Delta\}$ has at least one feasible solution, hence at least one optimal solution.

· Note that the above model does not exclude the possibility of dead-ends, namely states (c, V) such that $V \neq \varnothing$ and $\widehat{D}(c, V) = \varnothing$. For example, set again $\Delta = \delta = 3$ and consider the state $(c, V) = (15, \{14, 18\})$ and the decision $x = 14$. This will generate the state $\widehat{T}(15, \{14, 18\}, 14) = (14, \{18\}$ for which $\widehat{D}(14, \{18\}) = \varnothing$. For such states we can assign $\hat{f}(c, V) = \infty$.

· The forward *Push* method (see *Chapter 15*) is particularly suitable for the treatment of such problems.

The following example illustrates the application of the Corridor method in the solution of a small scale traveling salesman problem.

E.4.2 Example

This example is taken from Sniedovich and Voß [2006]. It consists of $k = 12$ cities with 1 being the home city. The distance matrix is given in Table E.1. The results generated by the *Corridor Method* with $\Delta = \delta = 4$ are summarized in Table E.2, where $TD(\mathbf{x}^{(j)})$ denotes the total distance associated with tour $\mathbf{x}^{(j)}$.

The initial incumbent $\mathbf{x}^{(1)}$ was generated by the *Nearest Neighbor* heuristic. Note that the fixed point $\mathbf{x}^* = \mathbf{x}^{(4)} = \mathbf{x}^{(5)}$ was reached in 4 iterations. It turns out that this solution is actually the global optimal solution to the problem.

Table E.1: Distance Matrix

$i\backslash j$	1	2	3	4	5	6	7	8	9	10	11	12
1	–	8	21	5	24	20	10	13	21	28	9	22
2	7	–	16	10	18	18	15	8	14	24	6	23
3	19	16	–	23	6	7	23	9	8	13	16	20
4	7	13	25	–	27	24	9	15	24	30	15	22
5	23	19	5	27	–	5	24	11	12	9	20	17
6	22	17	7	23	6	–	20	10	13	9	20	15
7	12	15	22	9	22	21	–	15	26	25	18	14
8	12	11	8	15	10	10	14	–	12	17	11	17
9	19	13	8	23	12	13	26	12	–	19	12	27
10	26	26	12	28	8	8	23	18	21	–	26	16
11	9	5	18	15	21	18	19	11	12	27	–	26
12	20	23	21	22	18	15	15	16	25	15	27	–

Table E.2: Results generated by the Corridor method: $\Delta = \delta = 4$

	1	2	3	4	5	6	7	8	9	10	11	12	1	$TD(\mathbf{x}^{(j)})$
$\mathbf{x}^{(1)}$	1	4	7	12	6	5	3	9	8	2	11	10	1	144
$\mathbf{x}^{(2)}$	1	4	7	12	6	5	3	9	10	8	11	2	1	122
$\mathbf{x}^{(3)}$	1	4	7	12	6	10	5	3	8	9	11	2	1	108
$\mathbf{x}^{(4)}$	1	4	7	12	10	6	5	3	9	8	11	2	1	105
$\mathbf{x}^{(5)}$	1	4	7	12	10	6	5	3	9	8	11	2	1	105

Table E.3 provides the results for $\Delta = \delta = 3$. Note that as in the case $\Delta = \delta = 4$, here the fixed point is a global optimal solution. Table E.4 provides a summary of the results generated for this problem when $\Delta = \delta = 2$. Note that in this case the fixed-point $\mathbf{x}^* = \mathbf{x}^{(2)} = \mathbf{x}^{(3)}$ is not a global optimal solution.

Observe that the choice of the home city can be an important factor here because the home city can lead only to δ cities whereas any city in position j on the current tour such that $n - m > j > m$, can lead to 2δ cities.

In this sense the home city and its immediate neighbors, on any incumbent tour, are disadvantaged relative to cities further away from the home city on that tour. One of the implications of this simple observation is that the choice of the home city can be deployed as a *seeding mechanism* to enable the search to "get out of" *fixed points*.

Table E.3: Results generated by the Corridor method: $\Delta = \delta = 3$

	1	2	3	4	5	6	7	8	9	10	11	12	1	$TD(\mathbf{x}^{(j)})$
$\mathbf{x}^{(1)}$	1	4	7	12	6	5	3	9	8	2	11	10	1	144
$\mathbf{x}^{(2)}$	1	4	7	12	6	5	3	9	8	10	11	2	1	129
$\mathbf{x}^{(3)}$	1	4	7	12	6	5	3	10	8	9	11	2	1	121
$\mathbf{x}^{(4)}$	1	4	7	12	6	10	5	3	9	8	11	2	1	108
$\mathbf{x}^{(5)}$	1	4	7	12	10	6	5	3	9	8	11	2	1	105
$\mathbf{x}^{(6)}$	1	4	7	12	10	6	5	3	9	8	11	2	1	105

Table E.4: Results generated by the Corridor method: $\Delta = \delta = 2$

	1	2	3	4	5	6	7	8	9	10	11	12	1	$TD(\mathbf{x}^{(j)})$
$\mathbf{x}^{(1)}$	1	4	7	12	6	5	3	9	8	2	11	10	1	144
$\mathbf{x}^{(2)}$	1	4	7	12	6	5	3	9	8	11	2	10	1	140
$\mathbf{x}^{(3)}$	1	4	7	12	6	5	3	9	8	11	2	10	1	140

E.5 Bibliographic Notes

The origins of the *Corridor Method* go back to early 1970s works describing DP applications to reservoir control problems where the objective was solutions capable of coping with the *Curse of Dimensionality* (see the survey by Yakowitz [1982]). Indeed, a very simple version of this method, called *Discrete Differential Dynamic Programming (DDDP)* had been used extensively to support DP in the solution of large reservoir operation problems (Heidari et al. [1971], Chow et al. [1975], Yakowitz [1982]). Iterative dynamic programming (Luus 2000) is based on similar ideas.

I should also note, that there are practical applications of instances of the traveling salesman problem where *precedence constraints* similar to **C-1** and **C-2** are already satisfied by optimal tours and where δ and Δ are much smaller than k. In such cases linear time dynamic programming algorithms can be devised to generate optimal solutions to the traveling salesman problem (Balas [1999], Balas and Simonetti [2001]).

This suggests the deployment of the *Corridor Method* with small values of δ and Δ — to speed up the algorithm — with the caveat that the optimality of the solutions recovered is not assured.

More on the application of the Corridor method can be found in Sniedovich and Voß [2006] and Caserta et al. [2008, 2010].

Bibliography

[1] Abdul-Razaq, T.S. and Potts, C.N., Dynamic programming state-space relaxation for single-machine scheduling, *Journal of the Operational Research Society,* 39(2), 141-152, 1988.

[2] Abrams, R.A. and Karmarkar, U.S., Infinite horizon investment-consumption policies, *Management Science,* 25(10), 1005-1013, 1979.

[3] Aczel, J., *Lectures on Functional Equations and Their Applications,* Academic Press, NY, 1966 (see p. 2).

[4] Agresti, W.W., Dynamic programming for computer register allocation, *Computers and Operations Research,* 4, 101-110, 1977.

[5] Ahuja, R.K., Magnanti, T.L. and Orlin, J.B., *Network Flow Theory, Algorithms, and Applications,* Prentice-Hall, Englewood-Cliffs, NJ, 1993.

[6] Alden, J.M. and Yano, C.A., *A forward dynamic programming approach for general uncapacitated multi-stage lot-sizing problems,* Technical Report 86-11, Department of Industrial and Operations Engineering, University of Michigan, Ann Arbor, MI, 1986.

[7] Alstrup, J., Boas, S., Madsen, O.B.G. and Vidal, R.V.V., Booking policy for flights with two types of passengers, *European Journal of Operational Research,* 27, 274-288, 1986.

[8] Andreatta, G. and Romero, L., Stochastic shortest paths with recourse, *Networks,* 18, 193-204, 1988.

[9] Angel, E., Dynamic programming for noncausal problems, *IEEE Transactions on Automatic Control,* 26(5), 1041-1047, 1981.

[10] Angel, E., From dynamic programming to fast transforms, *Journal of Mathematical Analysis and Applications,* 119, 82-89, 1986.

[11] Angel, E. and Bellman, R., *Dynamic Programming and Partial Differential Equations,* Academic Press, NY, 1972.

[12] Aoki, M., Adaptive control theory: Survey and potential applications to decision processes, *Decision Science,* 10, 666-687, 1979.

[13] Archibald, T.W., McKinnon, K.I.M. and Thomas, L.C., An aggregate stochastic dynamic programming model of multireservoir systems, *Water Resources Research,* 33(2), 333-340, 1997.

[14] Archibald, T.W., Buchanan, C.S., McKinnon, K.I.M. and Thomas, L.C., Nested benders decomposition and dynamic programming for reservoir optimisation, *Journal of the Operational Research Society,* 50(5), 468-479, 1999.

[15] Archibald, T.W., McKinnon, K.I.M. and Thomas, L.C., Modelling the operation of multi-reservoir systems using decomposition and stochastic dynamic programming, *Naval Research Logistics,* 53(3), 217-225, 2006.

[16] Aris, R., *The Optimal Design of Chemical Reactors, a Study in Dynamic Programming,* Academic Press, NY, 1961.

[17] Aris, R., *Discrete Dynamic Programming,* Blaisdell, NY, 1964.

[18] Arunkumar, S., Characterization of optimal operating policies for finite dams, *Journal of Mathematical Analysis And Applications,* 49, 2, 267-274, 1975.

[19] Aust, R.J., Dynamic programming branch and bound algorithm for pure integer programming, *Computers and Operations Research,* 3, 27-28, 1976.

[20] Avriel, M., *Nonlinear Programming: Analysis and Methods,* Prentice-Hall, Englewood Cliffs, NJ, 1976.

[21] Axsater, S., State aggregation in dynamic programming – An application to scheduling of independent jobs on parallel processors, *Operations Research Letters,* 2, 171-176, 1983.

[22] Axsater, S., An extension of the extended basic period approach for economic lot scheduling problems, *Journal of Optimization Theory and Applications,* 52(2), 179-189, 1987.

[23] Azhmyakov, V., Boltyanski, V. and Poznyaka, A., The dynamic programming approach to multi-model robust optimization, *Nonlinear Analysis,* 72, 1110-1119, 2010.

[24] Baetz, B.W., Pas E.I. and Neebe, A.W., Trash management: Sizing and timing decisions for incineration and landfill facilities, *Interfaces,* 19(6), 52-61, 1989.

[25] Balas, E., New classes of efficiently solvable generalized traveling salesman problems, *Annals of Operations Research,* 86, 529-558, 1999.

[26] Balas, E. and Simonetti N., Linear time dynamic programming algorithms for new classes of restricted TSP's: A computational study, *INFORMS Journal on Computing,* 13(1), 56-75, 2001.

[27] Balcer, Y., Partially controlled demand and inventory control: An additive model, *Naval Research Logistics Quarterly,* 27(2), 273-288, 1980.

[28] Balder, E.J., A new look at the existence of p-optimal policies in dynamic programming, *Mathematics of Operations Research,* 6, 513-317, 1981.

[29] Baldwin, J.F. and Pilsworth, B.W., Dynamic programming for fuzzy systems with fuzzy environment, *Journal of Mathematical Analysis and Applications,* 85, 1-23, 1982.

[30] Bather, J. and Bather, J.A., *Decision Theory: An Introduction to Dynamic Programming and Sequential Decisions,* Wiley, Chichester, UK, 2000.

[31] Barto, A.G., Bradtke, S.J. and Singh, S.P., Learning to act using real-time dynamic programming, *Artificial Intelligence,* 72(1-2), 81-138, 1995.

[32] Bazaraa, M.S. and Shetty, C.M., *Nonlinear Programming: Theory and Algorithms,* Wiley, NY, 1979.

[33] Baziw, J. and Scherer, C.R., Wastewater treatment capacity expansion with time varying assimilative constraints, *Water Resources Research,* 15(2), 219-227, 1979.

[34] Bean, J.C. and Smith, R.L., *Conditions for the existence of planning horizons,* Technical Report #81-8, Department of Industrial and Operations Engineering, The University of Michigan, Ann Arbor, MI, 1981.

[35] Bean, J.C., Birge, J.R. and Smith, R.L., Aggregation in dynamic programming, *Operations Research,* 35(2), 215-220, 1987.

[36] Beckmann, M., *Dynamic Programming of Economic Decisions,* Springer-Verlag, NY, 1968.

[37] Bellman, R., On the theory of dynamic programming, *Proceedings of the National Academy of Sciences,* 38, 716-719, 1952.

[38] Bellman, R., *An introduction to the theory of dynamic programming,* R-245, The Rand Corporation, Santa Monica, CA, 1953a.

[39] Bellman, R., Bottleneck problems and dynamic programming, *Proceedings of the National Academy of Sciences,* 39, 947-951,1953b.

[40] Bellman, R., Some functional equations in the theory of dynamic programming, *Proceedings of the National Academy of Sciences,* 39, 1077-1082, 1953c.

[41] Bellman, R., The theory of dynamic programming, *Bulletin of the American Mathematical Society,* 60, 503-515, 1954a.

[42] Bellman, R., Some problems in the theory of dynamic programming, *Econometrica,* 22, 37-48, 1954b.

[43] Bellman, R., Some applications of the theory of dynamic programming, *Operations Research,* 2, 275-288, 1954c.

[44] Bellman, R., Dynamic programming and Lagrange multipliers,*Proceedings of the National Academy of Sciences,* 42, 767-769, 1956a.

[45] Bellman, R., A problem in the sequential design of experiments, *Sankhya,* 16, 221-229, 1956b.

[46] Bellman, R., Dynamic programming and the smoothing problem, *Management Science,* 3, 11-113, 1956-7.

[47] Bellman, R., On a dynamic programming approach to the caterer problem – I, *Management Science,* 3, 270-278, 1956-7.

[48] Bellman, R., *Dynamic Programming,* Princeton University Press, Princeton, NJ, 1957a.

[49] Bellman, R., Dynamic programming and the numerical solution of variational problems, *Operations Research,* 5, 277-288, 1957b.

[50] Bellman, R., A Markovian decision process, *Journal of Mathematics and Mechanics,* 6(5), 679-684, 1957c.

[51] Bellman, R., On a routing problem, *Quarterly of Applied Mathematics,* 16(1), 87-90, 1958.

[52] Bellman, R., On the concepts of a problem and problem-solving, *American Mathematical Monthly,* 67, 119-134, 1960.

[53] Bellman, R., On the reduction of dimensionality for classes of dynamic programming processes, *Journal of Mathematical Analysis and Applications,* 3, 358-360, 1961a.

[54] Bellman, R., *Adaptive Control Processes: A Guided Tour,* Princeton University Press, Princeton, NJ, 1961b.

[55] Bellman, R., On the application of dynamic programming to the determination of optimal play in chess and checkers, *Proceedings of the National Academy of Sciences,* 53, 244-247, 1965a.

[56] Bellman, R., Dynamic programming, generalized states, and switching systems, *Journal of Mathematical Analysis and Applications,* 12, 360-363, 1965b.

[57] Bellman, R., An application of dynamic programming to the coloring of maps, *ICC Bulletin,* 4, 3-6, 1965c.

[58] Bellman, R., On the application of dynamic programming to the determination of optimal play in chess and checkers, *Proceedings of the National Academy of Sciences,* 53, 244-247, 1965.

[59] Bellman, R., Dynamic programming, pattern recognition and location of faults in complex systems, *Journal of Applied Probability,* 3, 268-271, 1966.

[60] Bellman, R., Dynamic programming: A reluctant theory. In Zwicky F. and Wilson, A.G., (Eds.), *New Methods of Thought and Procedure,* pp. 99-122, Springer-Verlag, NY, 1967.

[61] Bellman, R., Dynamic programming, and Lewis Carrol's game of doublets, *Bulletin of the Institute of Mathematics and its Applications,* 17, 86-87, 1968.

[62] Bellman, R., *Introduction to the Mathematical Theory of Control Processes,* Vol. 2, Academic Press, NY, 1971.

[63] Bellman, R., A note on cluster analysis and dynamic programming, *Mathematical Bioscience,* 18, 311-312, 1973.

[64] Bellman, R., *Eye of the Hurricane: an Autobiography,* World Scientific, Singapore, 1984.

[65] Bellman, R. and Dreyfus, S.E., *Applied Dynamic Programming,* Princeton University Press, Princeton, NJ, 1962.

[66] Bellman, R. and Kalaba, R., On the principle of invariant imbedding and the propagation through inhomogeneous media, *Proceedings of the National Academy of Sciences,* 42, 629-632, 1956.

[67] Bellman, R. and Kalaba, R., An inverse problem in dynamic programming and automatic control, *Journal of Mathematical Analysis and Applications,* 7, 322-325, 1963.

[68] Bellman, R., Kalaba, R. and Kotkin, B., Polynomial approximation – A new computational technique in dynamic programming allocation processes, *Mathematics and Computation,* 155-161, 1963.

[69] Bellman, R. and Lee, E.S., Functional equations in dynamic programming, *Aequationes Mathematicae,* 17, 1-18, 1978.

[70] Bellman, R., Cooke, K.L. and Lockett, A., *Algorithms Graphs and Computers,* Academic Press, NY, 1970.

[71] Bellman, R. and Roosta, M., A technique for the reduction of dimensionality in dynamic programming, *Journal of Mathematical Analysis and Applications,* 88, 543-546, 1982.

[72] Bellman, R., Sugiyama, H. and Kashef, B., Applications of dynamic programming and scan-rescan processes to nuclear medicine and tumor detection, *Mathematical Biosciences,* 21, 1-30, 1974.

[73] Bellman, R. and Zadeh, L.A., Decision-making in a fuzzy environment, *Management Science,* 17, 141-164, 1970.

[74] Ben-Tal, A., Golany, B., Nemirovski, A. and Vial, J-P., Retailer-supplier flexible commitments contracts: a robust optimization approach, *Manufacturing and Service Operations Management,* 7(3), 248-271, 2005.

[75] Benzig, H., Bounds for the approximation of dynamic programming, *Zeitschrift fur Operations Research,* 30, 65-77, 1986.

[76] Benzig, H., Kalin, D. and Heodorescu, R., On the Bernoulli three-armed bandit problem, *Optimization,* 17, 807-817, 1986.

[77] Berge, C., *Theorie des Graphes et ses Applications,* Dunod, Paris, 1958.

[78] Berge, C. and Ghouila-Houri, A., *Programming, Games, and Transportation Networks,* Wiley, NY, 1965.

[79] Bertele, U. and Brioschi, F., *Nonserial Dynamic Programming,* Academic Press, NY, 1972.

[80] Bertsekas, D.P., *Dynamic Programming and Stochastic Control,* Academic Press, NY, 1976.

[81] Bertsekas, D.P., *Linear Network Optimization,* MIT Press, Cambridge, MA, 1991.

[82] Bertsekas, D.P. and Shreve, S.E., *Stochastic Optimal Control: The Discrete Time Case,* Academic Press, NY, 1978.

[83] Bertsekas, D.P. and Tsitsiklis, J.N., *Neuro-Dynamic Programming,* Athena Scientific, Nashua, NH, 1996.

[84] Bhakta, P.C. and Mitra, S., Some existence theorems for functional equations arising in dynamic programming, *Journal of Mathematical Analysis and Applications,* 98, 348-362, 1984

[85] Bhatt, B.K. and Rosenbloom, E.S., A dynamic programming approach to generalized linear fractional programs, *Cahier du CERO,* 27, 207-212, 1985.

[86] Bird, R. and de Moor, O., *The Algebra of Programming,* Prentice Hall, NY, 1997.

[87] Blackwell, D., On the functional equation of dynamic programming, *Journal of Mathematical Analysis and Applications,* 2, 273-276, 1961.

[88] Blackwell, D., Discrete dynamic programming, *Annals of Mathematical Statistics,* 33, 719-726, 1962.

[89] Blackwell, D., Memoryless strategies in finite-stage dynamic programming, *Annals of Mathematical Statistics,* 35, 863-865, 1964.

[90] Blackwell, D., Discounted dynamic programming, *Annals of Mathematical Statistics,* 36, 226-235, 1965.

[91] Blot, J., An infinite-horizon stochastic discrete-time Pontryagin principle, *Nonlinear Analysis: Theory, Methods and Applications,* 71(12), e999-e1004, 2009.

[92] Bolt, R.A., *The Human Interface,* Lifetime Learning Publications, Belmont, CA, 1984, (p. 36, 42).

[93] Boltyanskii, V.G., Sufficient conditions for optimality and the justification of the dynamic programming method, *SIAM Journal of Control,* 4, 326-361, 1966.

[94] Boudarel, R., Delmas, J. and Guichet, P., *Dynamic Programming and its Applications to Optimal Control,* Academic Press, NY, 1971.

[95] Bouzaher, A., Braden, J.B. and Johnson, G.V., A dynamic programming approach to a class of nonpoint source pollution control problems, *Management Science,* 36(1), 1-15, 1990.

[96] Bradford, A.D. and Gero, J.S., Quantized feedforward and feedback information for dynamic programming problems, *Engineering Optimization,* 4, 227-228, 1980.

[97] Brassard, G. and Bratley, P., *Algorithmics,* Prentice-Hall, Englewood Cliffs, NJ, 1988.

[98] Bricker, D.L., Coding dynamic programming in APL, Paper WED/A.15.01, *Proceedings of the 12th International Symposium on Mathematical Programming,* Cambridge, Massachusetts, August 5-9, 1985.

[99] Broin, M. and Morin, T.L., DIP-Interactive dynamic programming software, Paper No. WA27.5, TIMS/ORSA Meeting, Chicago, April 1983, *TIMS/ORSA Bulletin,* 15, p. 187, 1983.

[100] Brown, Q.E., *Dynamic programming in computer science,* Technical Report CMU-CS-79-106, Computer Science Department, Carnegie-Mellon University, Pittsburgh, PA, 1979.

[101] Brown, T.A. and Strauch, R.E., Dynamic programming in multiplicative lattices, *Journal of Mathematical Analysis and Applications,* 12, 364-370, 1965.

[102] Buchanan, J.T., Alternative solution methods for the inventory replenishment problem under increasing demand, *Journal of the Operational Research Society,* 31, 615-620, 1980.

[103] Buhl, H.U., Generalizations and applications of a class of dynamic programming problems, *European Journal of Operational Research,* 31, 127-131, 1987.

[104] Byrne A., Sniedovich M. and Churilov, L., Handling soft constraints via composite concave programming, *Journal of the Operational Research Society,* 49, 870-877, 1998.

[105] Caccetta, L. and Giannini, L.M., An application of discrete mathematics in the design of an open pit mine, *Discrete Applied Mathematics,* 21, 1-19, 1988.

[106] Caplin, D.A. and Kornbluth, J.S.H., Multiobjective investment planning under uncertainty, *Omega,* 3(4), 423-441, 1975.

[107] Capuzzo, D.I., Fleming, W.H. and Zolezzi, T., (Eds.), *Recent Mathematical Models in Dynamic Programming,* Springer, Berlin, Germany, 1985.

[108] Carraway, R.L. and Morin, T.L., Theory and applications of generalized dynamic programming: An overview, *International Journal of Computers and Mathematics with Applications,* 16, 779-788, 1988.

[109] Carraway, R.L., Morin, T.L. and Moskowitz, H., Generalized dynamic programming for multicriteria optimization, *European Journal of Operational Research,* 44, 95-104, 1990.

[110] Carraway, R.L. and Schmidt, R.L., An improved discrete dynamic programming algorithm for allocating resources among interdependent projects, *Management Science,* 37(9), 1195-1200, 1991.

[111] Caserta, M., Voss, S. and Sniedovich, M., The corridor method – A general solution concept with application to the blocks relocation problem. In Bruzzone, A., Longo, F., Merkuriev, Y., Mirabelli, G. and Piera, M.A., (Eds.), *Proceedings of the 11th International Workshop on Harbour, Maritime and Multimodal Logistics Modeling and Simulation,* DIPTEM, Genova, 89-94, 2008.

[112] Caserta, M., Voss, S. and Sniedovich, M., Applying the Corridor method to a blocks relocation problem, *OR Spectrum,* in press, 2010.

[113] Cervellera, C. and Muselli, M., Deterministic learning and an application in optimal control, dynamic programming approach to approx-

imate dynamic programming. In Hawkes, P.W., (Ed.), *Advances in Imaging and Electron Physics,* 140, pp. 61-119, Academic Press, 2006.

[114] Chand, S. and Schneeberger, H., Single machine scheduling to minimize weighted earliness subject to no tardy jobs, *European Journal of Operational Research,* 34, 221-230, 1988.

[115] Chankong, V. and Haimes,Y.Y., A multiobjective dynamic programming method for capacity expansion, *IEEE Transactions on Automatic Control,* 26(5), 1195-1207, 1981.

[116] Cherkassky, B.V., Goldberg, A.V. and Radzik, T., Shortest paths algorithms: Theory and experimental evaluation. In *Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms,* Arlington, VA, pp. 516-525, 1994.

[117] Chou, Y-L., Pollock, S.M., Romeijn, H.E. and Smith, R.L., A Formalism for Dynamic Programming, working paper, Department of Industrial and Operations Engineering, The University of Michigan, Ann Arbor, MI, 2001.

[118] Chow, V.T., Maidment, D.R. and Tauxe, G.W., Computer time and memory requirements for DP and DDDP in water Resources systems analysis, *Water Resources Research,* 11, 621-628, 1975.

[119] Christensen, G.S. and Soliman, S.A., Long-term optimal operation of a parallel multireservoir power system, *Journal of Optimization Theory and Applications,* 50(3), 383-395, 1986.

[120] Christofides, N., Mingozzi, A. and Toth, P., Exact algorithms for the vehicle routing problem, based on spanning tree and shortest path relaxations, *Mathematical Programming,* 20, 255-282, 1981.

[121] Churilov, L., Ralph, D. and Sniedovich, M., A note on composite concave quadratic programming, *Operations Research Letters,* 23, 163-169, 1998.

[122] Churilov, L., Bomze, I., Sniedovich, M. and Ralph, D., Hypersensitivity analysis of portfolio optimization problems, *Asia Pacific Journal of Operations Research,* 21(3), 297-318, 2004.

[123] Clement, M.F., Categorical axiomatics of dynamic programming, *Journal of Mathematical Analysis and Applications,* 51, 47-55, 1975.

[124] Cohen, M.A., Pierskalla, W.P. and Namias, S., A dynamic inventory system with cycling, *Naval Research Logistics Quarterly,* 27(2), 289-296, 1980.

[125] Collins, D.C., Reduction of dimensionality in dynamic programming via the method of diagonal decomposition, *Journal of Mathematical Analysis and Applications,* 30, 223-234, 1970.

[126] Collins, D.C., Terminal state dynamic programming for differential equations, *Journal of Mathematical Analysis and Applications,* 31, 487-503, 1970.

[127] de Cooman, G. and Troffaes, M.C.M., Dynamic programming for deterministic discrete-time systems with uncertainty gain, *International Journal of Approximate Reasoning,* 39, 257-278, 2005.

[128] Cooper, M.W., The use of dynamic programming methodology for the solution of a class of nonlinear programming problems, *Naval Research Logistics Quarterly,* 27, 89-95, 1980.

[129] Cooper, L. and Cooper, M.W., *Introduction to Dynamic Programming,* Pergamon Press, NY, 1981.

[130] Cormen, T.H., Leiserson, C.E. and Rivest, R.L., *Introduction to Algorithms,* MIT Press, Cambridge, MA, 1990.

[131] Coskunoglu, O. and Parent, P., Piecewise smooth dynamic programming application to a casing design for ultradeep drills, *Engineering Optimization,* 5, 249-256, 1982.

[132] Craven B.D., *Mathematical Programming and Control Theory,* Chapman & Hall, London, 1977.

[133] Craven, B.D., *Fractional Programming,* Helderman Verlag, Berlin, Germany, 1988.

[134] Crowder, H., Mathematical programming algorithms in APL, in Mulvey, J.M., (Ed.), *Evaluating Mathematical Programming Techniques,* Springer Verlag, NY, 290-304, 1982.

[135] Daellenbach, H.G. and De Kluyver, C.A., Note on multiple objective dynamic programming, *Journal of the Operational Research Society,* 31, 591-594, 1980.

[136] Daellenbach, H.G., George, J.A. and McNickle, D.C., *Introduction to Operations Research Techniques,* 2nd Edition, Allyn and Bacon, Boston, MA, 1983.

[137] Dajani, J.S., Hasit, Y. and McCullers, S.D., Mathematical programming in sewer network design, *Engineering Optimization,* 3, 27-35, 1977.

[138] Dantzig, G.B., On the shortest path route through a network, *Management Science,* 6, 187-190, 1960.

[139] Dantzig, G.B., *Linear Programming and Extensions,* Princeton University Press, Princeton, NJ, 1963.

[140] Dantzig, G.B. and Thapa, N.M., *Linear Programming 2: Theory and Extensions,* Springer Verlag, Berlin, Germany, 2003.

[141] Davis, K.R. and Simmons, L.F., Improving assembly line efficiency: A dynamic programming-heuristic approach, *Computers and Operations Research,* 4, 75-87, 1977.

[142] Dawn, R.V., Pointwise and uniformly good stationary strategies for dynamic programming models, *Mathematics of Operations Research,* 11(3), 521-535, 1986.

[143] Dechter, A., Dechter, R. and Pearl, J., Optimization in constraint networks. In *Influence Diagrams, Belief Nets and Decision Analysis,* 411-425 , Wiley, NY, 1990.

[144] Dechter, R., Bucket elimination: A unifying framework for reasoning, *Artificial Intelligence 113(1-2),* 41-85, 1999.

[145] Denardo, E.V., Sequential Processes, Ph.D thesis, North-Western University, Evanston, IL, 1965.

[146] Denardo, E.V., Contraction mappings in the theory of dynamic programming, *SIAM Review,* 9, 165-177, 1968.

[147] Denardo, E.V., *Dynamic Programming Models and Applications,* Prentice-Hall, Englewood Cliffs, NJ, 1982.

[148] Denardo, E.V., *Dynamic Programming,* Dover, Mineola, NY, 2003.

[149] Denardo, E.V. and Fox, B.L., Shortest-route methods: 1. Reaching, pruning, and buckets, *Operations Research,* 27, 161-186, 1979a.

[150] Denardo, E.V. and Fox, B.L., Shortest-route methods: 2. Group knapsacks, expanded networks, and branch-and-bound, *Operations Research,* 27, 548-566, 1979b.

[151] Denardo, E.V. and Mitten, L.G., Elements of sequential decision processes, *Journal of Industrial Engineering,* 18, 106-112, 1967.

[152] Dianicch, D.F. and Gupta, J.N.D., Polynomial approximation technique for dynamic optimization problems, *Computers and Operations Research,* 13(4), 437-442, 1986.

[153] Diehl, M. and Björnberg, J., Robust dynamic programming for minmax model predictive control of constrained uncertain systems, *IEEE Transactions on Automatic Control,* 49(12), 2253-2257, 2004.

[154] Dijkstra, E.W., A note on two problems in connexion with graphs, *Numerische mathematik,* 1, 269-271, 1959.

[155] Dinkelbach, W., On nonlinear fractional programming, *Management Science,* 13, 492-498, 1967.

[156] Dolcetta, I.C., On a discrete approximation of the Hamilton-Jacobi equation of dynamic programming, *Applied Mathematics and Optimization,* 10, 367-377, 1983.

[157] Domingo, A. and Sniedovich, M., *Experiments with dynamic programming algorithms for nonseparable problems,* Preprint Series No. 8-1991, Department of Mathematics, The University of Melbourne, Melbourne, Australia, 1991.

[158] Dreyfus, S.E., An appraisal of some shortest-path algorithms, *Operations Research,* 17, 395-412, 1969.

[159] Dreyfus, S.E. and Law, A.M., *The Art and Theory of Dynamic Programming,* Academic Press, NY, 1977.

[160] Dror M., L'Ecuyer P. and Szidarovszky, F., (Eds.) *Essays on Uncertainty,* Kluwer, Boston, 2002.

[161] Duff, W.S., Minimum cost solar thermal electric power systems: A dynamic programming based approach, *Engineering Optimization,* 2, 83-95, 1976.

[162] Edmonds, J. and Karp, R.M., Theoretical improvements in algorithmic efficiency for network flow problems, *Journal of ACM,* 19, 2, 248-264, 1972.

[163] Ellis, D., *An abstract setting of the notion of dynamic programming,* P-783, The Rand Corporation, Santa Monica, CA, 1955.

[164] Elmaghraby, S.E., The concept of "state" in discrete dynamic programming, *Journal of Mathematical Analysis and Applications,* 29, 523-553, 1970.

[165] Elmaghraby, S.E., Comments on a DP model for the optimal inspection strategy, *IIE Transactions,* 18(1), 104-108, 1986.

[166] Epperson, J.E., Dynamic programming estate transfer model – The closely held farm business, *Computers and Operations Research,* 8(4), 347-353, 1981.

[167] Ergun, Ö. and Orlin, J.B., A dynamic programming methodology in very large scale neighborhood search applied to the traveling salesman problem, *Discrete Optimization,* 3, 78-85, 2006.

[168] Erikson, R.E. and Luss, H., Optimal sizing of records used to store messages of various lengths, *Management Science,* 26(8), 1980.

[169] Erlenkotter, D., Sequencing expansion projects, *Operations Research,* 21, 542-553, 1973.

[170] Esogbue, A.O. (Ed.), *Dynamic Programming for Optimal Water Resources Systems Analysis,* Prentice Hall, Englewwod Cliffs, NJ, 1989.

[171] Esogbue, A.O. and Lee, C.Y., Optimal design of large complex conveyance systems via nonserial dynamic programming. In in Esogbue, A.O., (Ed.), *Dynamic Programming for Optimal Water Resources Systems Analysis,* pp. 234-258, Prentice Hall, Englewwod Cliffs, NJ, 1989.

[172] Esogbue, A.O. and Marks, B.R., Non-serial dynamic programming – A survey, *Operations Research Quarterly,* 25, 253-265, 1974.

[173] Esogbue, A.O. and Marks, B.R., Dynamic programming models of the nonserial critical path-cost problem, *Management Science,* 24, 200-209, 1977.

[174] Esogbue, A.O. and Warsi, N.A., A high-level computing algorithm for diverging and converging branch nonserial dynamic programming systems, *Computers and Mathematics with Applications,* 12A(6), 719-732, 1986.

[175] Evans, J.R. and Minieka, E., *Optimization Algorithms for Networks and Graphs,* Marcel Dekker, NY, 1992.

[176] Evans, G.W., and Morin, T.L., Hybrid dynamic programming/branch and bound strategies for electric power generation planning, *IIE Transactions,* 18(2), 138-147, 1986.

[177] Feldman, R.M., Deuermeyer, B.L. and Curry, G.L., A dynamic programming optimization procedure for a two-product biomass-to-methane conversion system, *Journal of Mathematical Analysis and Applications,* 125, 203-212, 1987.

[178] Ferreira, J.A.S. and Vidal, R.V.V., Optimization of a pump-pipe system by dynamic programming, *Engineering Optimization,* 7, 241-251, 1984.

[179] Fleischer, I. and Kooharian, A., Optimization and recursion, *Journal of the Society of Industrial and Applied Mathematics,* 12(1), 186-188, 1964.

[180] Flynn, J., On optimality criteria for dynamic programs with long finite horizons, *Journal of Mathematical Analysis and Applications,* 76, 202-208, 1980.

[181] Florian, M., Lenstra, J.K. and Kan, R., Deterministic production planning algorithms and complexity, *Management Science,* 26(7), 669-679, 1980.

[182] Fontane, D.G. and Labadie, J.W., Optimal control reservoir discharge quality through selective withdrawal, *Water Resources Research,* 17(6), 1596-1604, 1981.

[183] Ford, L.R., *Network Flow Theory,* RAND paper P-923, Rand Corporation, Santa Monica, CA, 1956.

[184] Ford, L.R. and Fulkerson, D.R., Consructing maximal dynamic flows from static flows, *Operations Research,* 6, 419-433, 1958.

[185] Ford, L.R. and Fulkerson, D.R., *Flows in Networks,* Princeton University Press, Princeton, NJ, 1962.

[186] Fox, B.L., Finite-state approximations to denumerable-state dynamic programs, *Journal of Mathematical Analysis and Applications,* 34, 665-670, 1971.

[187] Fox, B.L., Discretizing dynamic programs, *Journal of Mathematical Analysis and Applications,* 11, 228-234, 1973.

[188] Fujita, T., Re-examination of Markov policies for additive decision process, *Bulletin of Informatics and Cybernetics,* 29(1), 51-65, 1997.

[189] Fujita, T., An optimal path problem with fuzzy expectation, *Proceedings of 8th Bellman Continuum,* pp. 191-195, December 2000.

[190] Fujita, T., An optimal path problem with threshold probability, *Frontiers in Artificial Intelligence and Applications,* 69, 792-796, 2001.

[191] Fujita, T., On policy classes in dynamic programming theory, *Proceedings of 9th Bellman Continuum International Workshop on Uncertain System and Soft Computing,* Series of Information and Management Sciences, Vol. 2, 39-43, July 2002.

[192] Fujita, T., On nondeterministic dynamic programming, Mathematical analysis in economics, *RMIS Kôkyûroku*, No. 1488, 15-24, 2006.

[193] Fujita, T., Infinite stage nondeterministic stopped decision processes and maximum linear equation, *Bulletin of the Kyushu Institute of Technology,* 55, 15-24, 2008.

[194] Fujita, T., Deterministic decision process under range constraint through all stages, *Proceedings of Modeling Decisions for Artificial Intelligence 2008*, (CD-ROM), pp. 60-70.

[195] Fujita, T. and Iwamoto, S., Fractional Markov decision processes, *Proceedings of International Workshop on Intelligent Systems Resolutions, The 8th Bellman Continuum,* pp. 7-11, National Tsing Hua University, Hsinchu, Taiwan, ROC, December 11-12, 2000,.

[196] Fujita, T. and Iwamoto, S., An optimistic decision-making in fuzzy environment, *The Bellman continuum,* (Santa Fe, NM, 1999). *Applied Mathematics and Computation,* 120(1-3), 123-137, 2001.

[197] Fujita, T. and Tsurusaki, K., Stochastic optimization of multiplicative functions with negative value, *Journal of the Operations Research Society of Japan,* 41(3), 351-373, 1998.

[198] Fujita, T., Ueno, T. and Iwamoto, S., A nondeterministic dynamic programming model. In Negoita et al., (Eds.) *Proceedings of The Eighth International Conference on Knowledge-Based Intelligent Information and Engineering Systems,* (KES 2004), Wellington, New Zealand, September 20-24, 2004. *Lecture Notes in Artificial Intelligence* (LNIA) 3214, 1208-1214, 2004.

[199] Furukawa, N., Characterization of optimal policies in vector-valued Markovian decision processes, *Mathematics of Operations Research,* 5, 271-279, 1980.

[200] Furukawa, N. and Iwamoto, S., Stopped decision processes on complete separable metric spaces, *Journal of Mathematical Analysis and Applications,* 31, 615-658, 1970.

[201] Furukawa, N. and Iwamoto, S., Markovian decision processes with recursive reward functions, *Bulletin of Mathematical Statistics,* 15(1-2), 79-91, 1973. Correction to this paper: *Bulletin of Mathematical Statistics,* 16(3-4), 127, 1974.

[202] Furukawa, N. and Iwamoto, S., Dynamic programming on recursive reward systems, *Bulletin of Mathematical Statistics,* 17(1-2), 103-126, 1976.

[203] Gal, S., Optimal management of a multireservoir water supply system, *Water Resources Research,* 15, 737-749, 1979.

[204] Gal, S., The parameter iteration method in dynamic programming, *Management Science,* 35(6), 675-684, 1989.

[205] Gallo, G. and Pallottino, S., Shortest path algorithms, *Annals of Operations Research* , 13, 3-79, 1988.

[206] Galperin, E.A., Reflections on Bellman's optimality principle, *Nonlinear Analysis,* 63, e707-e713, 2005.

[207] Galperin, E.A., Reflections on optimality and dynamic programming, *Computers and Mathematics with Applications,* 52, 235-257, 2006.

[208] Garcia de la Banda, M. and Stuckey, P.J., Dynamic programming to minimize the maximum number of open stacks, *INFORMS Journal of Computing,* 19(4), 607-617, 2007.

[209] Garey, M. and Johnson, D., *Computers and Intractability: A Guide to the Theory of NP-Completeness,* W.H. Freeman, San Francisco, CA, 1979.

[210] Gass, S.I. and Harris, C.M, *Encyclopedia of Operations Research and management Science,* Kluwer, Boston, Mass, 1996.

[211] Geering, H.P., *Optimal Control with Engineering Applications,* Springer, 2007.

[212] Gerchak, Y. and Henig, M., The basketball shootout: Strategy and winning probabilities, *Operations Research Letters,* 5(3), 241-244, 1986.

[213] Gero, J.S., Architectural optimization – A review, *Engineering Optimization,* 1, 189-199, 1975.

[214] Gero, J.S., Sheehan, P.J. and Becker J.M., Building design using feedforward nonserial dynamic programming, *Engineering Optimization,* 3, 183-192, 1978.

[215] Gero, J.S. and Bradford, A.D., A dynamic programming approach to the optimum lighting problem, *Engineering Optimization,* 3, 71-82, 1978.,

[216] Gero, J.S. and Kaneshalingam, K., A method for the optimum design of tradition al formwork, *Engineering Optimization,* 3, 249-251, 1978.

[217] Giegerich, R., A systematic approach to dynamic programming in bioinformatics, *Bioinformatics,* 16(8), 665-677, 2000.

[218] Gilmore, P.C. and Gomory, R.E., The theory and computation of knapsack functions, *Operations Research,* 14, 1045-1074, 1966.

[219] Goldstein, T., Ladany, S.P. and Mehrez, A., A discounted machine replacement model with an expected future technological breakthrough, *Naval Research Logistics,* 35, 209-220, 1988.

[220] Greenberg, H., An algorithm for the periodic solutions in the knapsack problem, *Journal of Mathematical Analysis and Applications,* 111, 327-331, 1985.

[221] Greenberg, H., On equivalent knapsack problems, *Discrete Applied Mathematics,* 14, 263-268, 1986.

[222] Grossman, D. and Marks, D.H., Capacity expansion for water resource facility: A control theoretic algorithm for nonstationary uncertain demands, *Advances in Water Resources,* 1(1), 31-40, 1977.

[223] Gunasekaran, S. and Shove, G.C., Dynamic programming optimization of total airflow requirement for low temperature maize (corn) drying, *Journal of Agricultural Engineering and Research,* 34(1), 41-51, 1986.

[224] Gunter, S. and Swanson, L., Semi-Markov dynamic programming approach to competitive bidding with state space reduction considerations, *European Journal of Operational Research,* 32, 435-447, 1986.

[225] Harley, M.J. and Chidley, T.R.E., Policy iteration dynamic programming applied to reservoir control, *Engineering Optimization,* 2, 161-171, 1976.

[226] Harley, M.J. and Chidley, T.R.E., Deterministic dynamic programming for long term reservoir operating policies, *Engineering Optimization,* 3, 63-70, 1978.

[227] Harmanec, D., Generalizing Markov decision processes to imprecise probabilities, *Journal of Statistical Planning and Inference,* 105(1), 199-213, June 2002.

[228] Hartley, R., Laercombe, A.C. and Thomas, L.C., Computational comparison of policy iteration algorithms for discounted Markov decision processes, *Computers and Operations Research,* 13, 411-420, 1986.

[229] Hartman, J.C. and Perry, T.C., Approximating the solution of a dynamic, stochastic multiple knapsack problem, *Control and Cybernetics,* 35(3), 535-550, 2006.

[230] Hastings, N.A.J., *DYNACODE Dynamic Programming Systems Handbook,* Management Center, University of Bradford, Bradford, UK, 1974.

[231] Heidari, M., Chow, V.T., Kokotovic, P.V. and Meredith, D.D., Discrete differential dynamic programming approach to water resources systems optimization, *Water Resources Research,* 7(2), 273-282, 1971.

[232] Helgason, R.V., Kennigton, J.L. and Stewart, B.D., The one-to-one shortest-path problem: An empirical analysis with the two-tree Hijkstra algorithm, *Computational Optimization and Applications,* 1, 47-75, 1993.

[233] Held, M. and Karp, R.M., A dynamic programming approach to sequencing problems, *Journal of the Society of Industrial and Applied Mathematics,* 10, 196-210, 1962.

[234] Held, M., Karp, R.M. and Shareshian, R., Assembly-line balancing dynamic programming with precedence constraints, *Operations Research,* 11, 442-459, 1963.

[235] Helman, P., The principle of optimality in the design of efficient algorithms, *Journal of Mathematical Analysis and Applications,* 119, 97-127, 1986.

[236] Helman, P., A common schema for dynamic programming and branch and bound algorithms, *Journal of the Association for Computing Machinery,* 36(1), 97-128, 1989.

[237] Helman, P. and Rosenthal, A., A comprehensive model of dynamic programming, *SIAM journal of Algebraic Discrete Methods,* 6, 319-334, 1985.

[238] Henig, M.I., Vector-valued dynamic programming, *SIAM Journal of Control and Optimization,* 21, 490-499, 1983.

[239] Henig, M.I., The shortest path problem with two objective functions, *European Journal of Operational Research,* 25, 281-291, 1986.

[240] Henig, M.I., Extensions of the dynamic programming method in the deterministic and stochastic assembly-line balancing problems, *Computers and Operations Research,* 13, 443-449, 1986.

[241] Henig, M.I., The principle of optimality in dynamic programming with returns in partially ordered sets, *Mathematics of Operations Research,* 10(3), 462-470, 1987.

[242] Henig, M.I., Risk criteria in a stochastic knapsack problem, *Operations Research,* 38(5), 820-825, 1990.

[243] Henig, M. and Gerchak, Y., The structure of periodic review policies in the presence of random yield, *Operations Research,* 38(4), 634-643, 1990.

[244] Hernandez-Lerma, O., Finite-state approximations for denumerable multi dimensional state discounted Markov decision processes, *Journal of Mathematical Analysis and Applications,* 113, 382-389, 1986.

[245] Hill, T.P. and Van der Wal, J., Monotonically improving limit-optimal strategies in finite-state decision processes, *Mathematics of Operations Research,* 12(3), 463-473, 1987.

[246] Hillier, F.S. and Lieberman, G.J., *Introduction to Operations Research,* (5th Edition), Holden Day, Oakland, CA, 1990.

[247] Hinderer, K., *Foundations of Non-Stationary Dynamic Programming with Discrete Time Parameter,* Springer-Verlag, NY, 1970.

[248] Hinxman, A.I., The trim-loss and assortment problems: A survey, *European Journal of Operational Research,* 5, 8-18, 1980.

[249] Hirschberg, D.S. and Larmore, L.L., The least weight subsequence problem, *SIAM Journal of Computing,* 16(4), 628-638, 1987.

[250] Ho, J.K., *Linear and Dynamic Programming with Lotus 1-2-3,* MIS Press, Portland, OR, 1987.

[251] Ho, J.K., OPTIMACROS: Optimization with spreadsheet macros, *Operations Research Letters,* 6(3), 99-103, 1987.

[252] Howard, R., *Dynamic Programming and Markov processes,* Wiley, NY, 1960.

[253] Howson, H.R. and Sancho, N.G.F., A new algorithm for the solution of multistate dynamic programming problems, *Mathematical Programming,* 8, 104-116, 1975.

[254] Hwang, C.L. and Fan, L.T., A discrete version of Pontryagin's maximum principle, *Operations Research,* 15(1), 139-146, 1967.

[255] Ibaraki, T., Representation theorems for equivalent optimization problems, *Information and Control,* 21, 397-435, 1972.

[256] Ibaraki, T., Solvable classes of discrete dynamic programming, *Journal of Mathematical Analysis and Applications,* 43, 642-693, 1973.

[257] Ibaraki, T., Classes of discrete optimization problems and their decision problems, *Journal of Computer and System Sciences,* 8, 84-116, 1974.

[258] Ibaraki, T., Minimal representations of some classes of dynamic programming, *Information and Control,* 27, 289-328, 1975.

[259] Ibaraki, T. Branch and bound procedure and state-space representation of combinatorial optimization problems, *Information and Control,* 36(1), 1-27, 1978.

[260] Ibaraki, T., Parametric approaches to fractional programming, *Mathematical Programming,* 26, 345-362, 1983.

[261] Ibaraki, T., *Enumerative Approaches to Combinatorial Optimization,* J.C. Baltzer AG, Basel, Switzerland, 1987.

[262] Ikem, F.M. and Reisman, A.M., An approach to planning for physician requirements in developing countries using dynamic programming, *Operations Research,* 38(4), 607-618, 1990.

[263] Iyengar, G.N., Robust dynamic programming, *Mathematics of Operations Research,* 30(2), 57-280, 2005.

[264] Isaacs, R., *Games of pursuit,* P-257, The Rand Corporation, Santa Monica, CA, 1951.

[265] Iwamoto, S., Stopped decision processes on compact metric spaces, *Bulletin of Mathematical Statistics,* 14(1-2), 51-60, 1970.

[266] Iwamoto, S., Average reward Markovian decision processes in the completely ergodic case, *Bulletin of Mathematical Statistics,* 15(3-4), 55-69, 1973.

[267] Iwamoto, S., Discrete dynamic programming with recursive additive system, *Bulletin of Mathematical Statistics,* 15(1-2), 49-66, 1974.

[268] Iwamoto, S., Linear programming on recursive additive dynamic programming, *Journal of the Operations Research Society of Japan,* 18(3-4), 125-151, 1975.

[269] Iwamoto, S., Minimax inverse theorems in dynamic programming, *Bulletin of Mathematical Statistics,* 17(1-2), 93-101, 1976.

[270] Iwamoto, S., Inverse theorem in dynamic programming I, *Journal of Mathematical Analysis and Applications,* 58(1), 113-134, 1977.

[271] Iwamoto, S., Inverse theorem in dynamic programming II, *Journal of Mathematical Analysis and Applications,* 58(2), 249-279, 1977.

[272] Iwamoto, S., Inverse theorem in dynamic programming III, *Journal of Mathematical Analysis and Applications,* 58(3), 439-448, 1997.

[273] Iwamoto, S., Dynamic programming approach to inequalities, *Journal of Mathematical Analysis and Applications,* 59(3), 687-704, 1977.

[274] Iwamoto, S., A class of inverse theorem on recursive programming with monotonicity, *Journal of the Operations Research Society of Japan,* 20(2), 94-112, 1977.

[275] Iwamoto, S., The second principle of optimality, *Bulletin of Mathematical Statistics,* 17(3-4), 101-114, 1977.

[276] Iwamoto, S., Mathematical programming problems represented by dynamic programming. Mathematical programming and decision processes (Japanese), *RMIS Kôkyûroku*, No. 299, 27-53, 1997.

[277] Iwamoto, S., Dynamic programming as sequential decision processes I (Japanese), *Communications of the Operations Research Society of Japan,* 22(7), 427-434, 1977.

[278] Iwamoto, S., Dynamic programming as sequential decision processes II (Japanese), *Communications of the Operations Research Society of Japan,* 22(8), 496-501, 1977.

[279] Iwamoto, S., An inverse theorem between main and inverse dynamic programming: Infinite-stage case. In Puterman, M.L., (Ed.), *Dynamic Programming and Its Application*, pp. 319-334, Academic Press, NY, 1978.

[280] Iwamoto, S., Some operations on dynamic programming with one-dimensional state space, *Journal of Mathematical Analysis and Applications,* 69(1), 263-282, 1979.

[281] Iwamoto, S., Sequential decision processes and dynamic programming. Decision processes and its related fields (Japanese), *RMIS Kôkyûroku*, No. 299, 193-218, 1979.

[282] Iwamoto, S., Theory of dynamic programming and applications (Japanese), *Sugaku,* 31(4), 331-348, 1979.

[283] Iwamoto, S., Reversed control processes, *Bulletin of Mathematical Statistics,* 19(1-2), 1-11, 1980.

[284] Iwamoto, S., An inverse control process and an inverse allocation process, *Journal of the Operations Research Society of Japan,* 24(1), 1-18, 1981.

[285] Iwamoto, S., Inversion of dynamic program and its applications to allocation processes, *Journal of Mathematical Analysis and Applications,* 81(2), 474-496, 1981.

[286] Iwamoto, S., A new inversion of continuous-time optimal control processes, *Journal of Mathematical Analysis and Applications,* 82(1), 49-65, 1981.

[287] Iwamoto, S., On three allocation processes. Markov game and its related fields (Japanese), *RMIS Kôkyûroku*, No. 460, 160-183, 1982.

[288] Iwamoto, S., Inverse functional equations for Bellman's allocations, *Bulletin of Informatics and Cybernetics,* 20, 57-68, 1983.

[289] Iwamoto, S., Reverse function, reverse program and reverse theorem in mathematical programming, *Journal of Mathematical Analysis and Applications,* 95(1), 1-19, 1983.

[290] Iwamoto, S., Multi-stage allocation processes – Reward maximization versus cost minimization, *Proceeding of The 4th Mathematical Programming Symposium,* (Kobe, 1983), pp. 91-108, 1983.

[291] Iwamoto, S., A dynamic inversion of the classical variational problems, *Journal of Mathematical Analysis and Applications,* 100(2), 354-374, 1984.

[292] Iwamoto, S., Sequential minimaximization under dynamic programming structure, *Journal of Mathematical Analysis and Applications,* 108(1), 267-282, 1985.

[293] Iwamoto, S., On Bellman's allocation processes, *Journal of Mathematical Analysis and Applications,* 111(1), 65-89, 1985.

[294] Iwamoto, S., *Theory of Dynamic Program* (Japanese), Kyushu University Press, Fukuoka, 1987.

[295] Iwamoto, S., The graphic method of dynamic programming (Japanese), *Communications of the Operations Research Society of Japan,* 32(6), 399-403, 1987.

[296] Iwamoto, S., Dynamic proggramming versus inequalities. Operator theory and its related fields (Japanese), *RMIS Kôkyûroku*, No. 653, 109-129, 1988.

[297] Iwamoto, S., A three-mirror problem on dynamic programming. Modeling and control of systems in engineering, quantum mechanics, economics and biosciences (Sophia-Antipolis, 1988), *Lecture Notes in Control and Information Sciences,* 121, 363-382, Springer, 1989.

[298] Iwamoto, S., A multi-parametric linear program and its dual and converse. Decision theory and its related fields (Japanese), *RMIS Kôkyûroku*, No. 726, 153-166, 1990.

[299] Iwamoto, S., Recent progress in dynamic programming, *Proceedings of the 2nd RAMP(Research Association of Mathematical Programming) Symposium,* pp. 129-140, Kyoto Research Park, Japan, 1990.

[300] Iwamoto, S., Parallel composition of dynamic proggramming. Optimization theory and its related fields (Japanese), *RMIS Kôkyûroku*, No. 747, 108-121, 1991.

[301] Iwamoto, S., Iterative integral versus dynamic programming, *Proceedings of the 4th Bellman Continuum Workshop,* Kansas State University, May 21-22, 1990. *Computers and Mathematics with Applications,* 21(11-12), 23-39, 1991.

[302] Iwamoto, S., From dynamic programming to bynamic programming, *Journal of Mathematical Analysis and Applications,* 177(1), 56-74, 1993a.

[303] Iwamoto, S., On Markov bidecision processes. Mathematics of optimization and its applications (Japanese), *RMIS Kôkyûroku*, No. 835, 165-172, 1993b.

[304] Iwamoto, S., On stochastic fuzzy decision processes. Optimization theory and mathematical structure (Japanese), *RMIS Kôkyûroku*, No. 864, 203-213, 1994a.

[305] Iwamoto, S., On bi-decision processes, *Journal of Mathematical Analysis and Applications,* 187(2), 676-699, 1994b.

[306] Iwamoto, S., On minimax linear equations. Optimization: Modeling and algorithm (Japanese) (Tokyo, 1993). The Institute of Statistical Mathematics, Joint Research Report No. 53, 56-72, 1994.

[307] Iwamoto, S., Fuzzy dynamic programming through invariant imbedding, *Proceedings of 33rd symposium of Japan Operations Research Society, Fuzzy mathematical programming and its applications,* pp. 25-33, Hiroshima, Japan, 1995.

[308] Iwamoto, S., Associative dynamic programs, *Journal of Mathematical Analysis and Applications,* 201(1), 195-211, 1996.

[309] Iwamoto, S., Constrained sums. Discrete and continuous structure in mathematical optimization (Japanese), *RMIS Kôkyûroku*, No. 945, 163-172, 1996.

[310] Iwamoto, S., Fuzzy dynamic programming. Teminology in soft computing (Japanese), *Asakura*, pp. 42, Tokyo, 1996.

[311] Iwamoto, S., Optimal policies for optimization of associative functionals. *Mathematical analysis in economics* (Japanese), *RMIS Kôkyûroku*, No. 987, 143-163, 1997.

[312] Iwamoto, S., Decision-making in fuzzy environment: A survey from stochastic decision process. In Jain, L.C. and Jain, R.K., *Proceedings of the 2nd International Conference on Knowledge-based Intelligent Electronics Systems,* (KES '98), pp. 542-546, Adelaide, Australia, 1998.

[313] Iwamoto, S., Dual fuzzy dynamic programming. Dynamic decision systems in uncertain environments (Japanese), *RMIS Kôkyûroku*, No. 1048, 72-85, 1998.

[314] Iwamoto, S., On expected values of Markov statistics, *Bulletin of Mathematics and Cybernetics,* 30(1) , 1-24, 1998.

[315] Iwamoto, S., Conditional decision processes with recursive reward function, *Journal of Mathematical Analysis and Applications,* 230(1), 193-210, 1999.

[316] Iwamoto, S., A priori and a posteriori conditional decision processes with nonadditively recursive utility. Mathematical analysis in economics (Japanese), *RIMS Kôkyûroku*, No. 1108, 29-43, 1999.

[317] Iwamoto, S., Euler partition rule. In *Proceedings of International Conference on Nonlinear Analysis and Convex Analysis,* pp. 173-180, Niigata, Japan, July 28-31, 1998, World Scientific, Singapore, 1999.

[318] Iwamoto, S., Recurrence formulas and decision tree tables in stochastic optimization. Mathematical decision theory under uncertainty and ambiguity (Japanese), *RIMS Kôkyûroku*, No. 1132, 15-23, 2000.

[319] Iwamoto, S., Nearest route problem, *Journal of Mathematical Analysis and Applications,* 249(1), 160-178, 2000.

[320] Iwamoto, S., Maximizing threshold probability through invariant imbedding, *Proceedings of International Workshop on Intelligent Systems Resolutions, The 8th Bellman Continuum,* pp. 17-22, National Tsing Hua University, Hsinchu, Taiwan, ROC, December 11-12, 2000.

[321] Iwamoto, S., Fuzzy decision-making through three dynamic programming approaches, *Proceedings of International Workshop on Intelligent Systems Resolutions, The 8th Bellman Continuum,* pp. 23-27, National Tsing Hua University, Hsinchu, Taiwan, ROC, December 11-12, 2000.

[322] Iwamoto, S., A nearest route problem. (Japanese) Mathematical science of optimization (Japanese), *RMIS Kôkyûroku*, No. 1174, 32-44, 2000.

[323] Iwamoto, S., Recursive method in stochastic optimization under compound criteria, *Advances in Mathematical Economics,* 3, 63-82, 2001.

[324] Iwamoto, S., A class of dual fuzzy dynamic programs, *The Bellman continuum,* (Santa Fe, NM, 1999).*Applied Mathematics and Computation,* 120(1-3), 91-108, 2001.

[325] Iwamoto, S., Fuzzy dynamic programming in stochastic environment. In Yoshida, Y., (Ed.), *Dynamic Aspects in Fuzzy Decision Making, Studies in Fuzziness and Soft Computing,* Vol. 73), 27-51, Physica-Verlarg, Heidelberg, Germany, 2001.

[326] Iwamoto, S., Fuzzy decision-making through three dynamic programming approaches, special issue on "The trend of the resolutions of the intelligent systems" (Hsinchu, 2000), *International Journal of Fuzzy Systems,* 3(4), 520-526, 2001.

[327] Iwamoto, S., When to stop accumulating reward/cost. Mathematical analysis in economics (Japanese), *RMIS Kôkyûroku*, No. 1337, 32-43, 2000.

[328] Iwamoto, S., Fuzzy dynamic programming, *Communications of the Operations Research Society of Japan* (Japanese), 47(5), 309-314, 2002.

[329] Iwamoto, S., Optimal stopping in fuzzy environment, *Proceedings of the 9th Bellman Continuum, International Workshop on Uncertain Systems and Soft Computing,* pp. 264-269, Beijing, China, July 24-27, 2002.

[330] Iwamoto, S., On controlled difference equations (Japanese). In Kawasaki, H., (Ed), *Proceedings of OR and Mathematics,* Fukuoka, Japan, pp. 57-75, 2003.

[331] Iwamoto, S., Recursive methods in probability control, *Advances in Mathematical Economics,* 6, 55-68, 2004.

[332] Iwamoto, S., Stochastic optimization of forward recursive functions, *Journal of Mathematical Analysis and Applications,* 292(1), 73-83, 2004.

[333] Iwamoto, S., A dynamic pricing of exotic options. In Pan, H., Sornette, D. and Kortanek, K., *Proceedings of The 1st International Workshop on Intelligent Finance,* (IWIF1), pp. 252-265, Melbourne, Australia, December 13-14, 2004.

[334] Iwamoto, S., Primitive dynamic programming, *Bulletin of Mathematical Statistics,* 36(1-2), 163-172, 2004.

[335] Iwamoto, S., Cross dual on the Golden optimum solutions, *RIMS Kôkyûroku* No. 1443, 27-43, 2005.

[336] Iwamoto, S., Prominent Books and Articles in the 20th Century: Richard E. Bellman: Dynamic Programming (Japanese), *Information Processing Society of Japan,* Vol. 46, No.7, p. 842, 2005.

[337] Iwamoto, S., The Golden trinity – Optimality, inequality, identity. Mathematical analysis in economics (Japanese), *RMIS Kôkyûroku*, No. 1488, 1-14, 2006.

[338] Iwamoto, S., The Golden optimum solution in quadratic programming. In Takahashi, W. and Tanaka, T., (Eds.), *Proceedings of the International Conference on Nonlinear Analysis and Convex Analysis,* (Okinawa, 2005), pp. 109-205, Yokohama Publishers, Yokohama, Japan, 2007.

[339] Iwamoto, S., Golden optimal policy in calculus of variation and dynamic programming, *Advances in Mathematical Economics,* 10, 65-89, 2007.

[340] Iwamoto, S., Golden optimal policies on dynamic optimization, (Japanese) Mathematical Optimization in Economic Analysis, *Mita Gakkai Journal* (Keio Economic Society), 101-125, 2007.

[341] Iwamoto, S., Golden quadruplet: Optimization - inequality - identity - operator, Modeling Decisions for Artificial Intelligence, *Proceedings of the Fourth International Conference,* (MDAI 2007), Kitakyushu, Japan, August 16-18, 2007, Torra, V., Narukawa, Y. and Y. Yoshida, Y., (Eds.), *Springer-Verlag Lecture Notes in Artificial Intelligence,* Vol. 4617, 14-23, 2007.

[342] Iwamoto, S., Golden optimal processes on three dynamics: Deterministic, stochastic and non-deterministic. Mathematical analysis in economics (Japanese), *RIMS Kôkyûroku*, to appear.

[343] Iwamoto, S. and Fujita, T., Stochastic decision process in fuzzy environment (Japanese), *Communications of Operations Research Society of Japan,* 39(10), 517-521, 1994.

[344] Iwamoto, S. and Fujita, T., Stochastic decision-making in a fuzzy environment, *Journal of the Operations Research Society of Japan,* 38(4), 467-482, 1995.

[345] Iwamoto, S. and Fujita, T., Markov decision processes with fractional values. Decision theory in mathematical modeling (Japanese), *RMIS Kôkyûroku*, No. 1079, 153-163, 1999.

[346] Iwamoto, S., Fujita, T. and Tsurusaki, K., Conditional decision-making in a fuzzy environment. Optimization theory in discrete and continuous mathematics (Japanese), *RMIS Kôkyûroku*, No. 1015, 150-162, 1997.

[347] Iwamoto, S. and Iki, T., An inverse assignment problem. Optimization theory in discrete and continuous mathematics (Japanese), *RMIS Kôkyûroku*, No. 1015, 163-175, 1997.

[348] Iwamoto, S. and Kai, Y., On deterministic stationary strategies for Markov games, *Bulletin of Mathematical Statistics,* 16(1-2), 71-82, 1975.

[349] Iwamoto, S. and Kira, A., On Golden inequalities, the development of information and decision processes, *RMIS Kôkyûroku*, No. 1504, pp. 168-176, Research Institute for Mathematical Sciences, Kyoto University, Kyoto Japan, July 2006.

[350] Iwamoto, S., Kira, A. and Yasuda, M., Golden duality in dynamic optimization. In Matsuhisa, T., (Ed.), *Proceedings of the Second Kosen Workshop, Mathematics, Technology and Education,* (MTE2008 Ibaraki), Ibaraki National College of Technology, February 15-18, pp. 35-47, 2008.

[351] Iwamoto, S., Tomkins, R.J. and Wang, C.L., Some theorems on reverse inequalities, *International Journal of Fuzzy Systems,* 119(1), 282-299, 1986.

[352] Iwamoto, S. and Sniedovich, M., Sequential decision making in fuzzy environment, *Journal of Mathematical Analysis and Applications,* 222(1), 208-224, 1998.

[353] Iwamoto, S., Tomkins, R.J. and Wang, C.L., Sequential control for unknown influences, *Proceedings of the 15th Conference on Numerical Mathematics and Computing,* Univ. of Manitoba, Winnipeg, Oct. 1985. *Congressus Numerantium,* 51, 193-208., 1986.

[354] Iwamoto, S., Tomkins, R.J. and Wang, C.L., A generalization of Hölder inequalities through dynamic programming approach, *Journal of Mathematical Analysis and Applications,* 135(1), 8-33, 1988.

[355] Iwamoto, S. and Tsurusak, K., On maximum processes. Perspective and problems for dynamic programming with uncertainty (Japanese), *RMIS Kôkyûroku*, No. 1207, 101-113, 2001.

[356] Iwamoto, S., Tsurusaki, K., and Fujita, T., Conditional decision-making in a fuzzy environment, *Journal of the Operations Research Society of Japan,* 42(2), 198-218, 1999.

[357] Iwamoto, S., Tsurusaki, K. and Fujita, T., On Markov policies for minimax decision processes, *Journal of Mathematical Analysis and Applications,* 253(1), 58-78, 2001.

[358] Iwamoto, S., and Ueno, T., Inverse partition problems, *Bulletin of Informatics and Cybernetics,* 31(1), 67-90, 1999.

[359] Iwamoto, S. and Ueno, T., Fuzzy decision-making under threshold membership criterion. In Baba et al., (Eds.), *Proceedings of The 5th International Conference on Knowledge-Based Intelligent Information Engineering Systems and Allied Technologies,* (KES' 2001), Osaka, Japan, September 6-8, 2001. *Frontiers in Artificial Intelligence and Applications,* 69, Part.1, 797-801, IOS Press, Amsterdam, The Netherlands, 2001.

[360] Iwamoto, S. and Wang, C.L., Continuous dynamic programming approach to inequalities, *Journal of Mathematical Analysis and Applications,* 96(1), 119-129, 1983.

[361] Iwamoto, S., and Wang, C.L., Continuous dynamic programming approach to inequalities II, *Journal of Mathematical Analysis and Applications,* 118(2), 276-286, 1986.

[362] Iwamoto, S. and Yasuda, M., Golden optimal values in discrete-time dynamic optimization processes, in T. Matsuhisa, T., (Ed.), *Proceedings of the first Kosen Workshop – Mathematics, technology and education,* (MTE2006), pp. 29-39, Ibaraki National College of Technology, Ibaraki, Japan, December 5-7, 2006.

[363] Iwamoto, S., and Yasuda, M., Golden optimal values in discrete-time dynamic optimization processes, *The Development of Probability Models on Optimizatio Problems, RIMS Kôkyûroku 1559,* pp. 56-66, Research Institute for Mathematical Sciences, Kyoto University, Kyoto Japan, June 2006.

[364] Iwamoto, S. and Yasuda, M., Dynamic programming creates the Golden ratio, too, Decision-making and mathematical model under uncertainty (Japanese) (Kyoto, 2005). *RIMS Kôkyûroku* No. 1477, 136-140, 2006.

[365] Iwamoto, S., and Yasuda, M., Golden optimal path in discrete-time dynamic optimization processes. In Elaydi, S., Nishimura, K., and Shishikura, M., (Eds.), *Advanced Studies in Pure Mathematics,* 53, 2009, ICDEA2006, pp. 99-108. *Proceedings of The International Conference on Differential Equations and Applications,* (ICDEA06), Kyoto University, Kyoto, Japan, July 24-28, 2006.

[366] Jacobson, D. and Mayne, D., *Differential Dynamic Programming,* Elsevier, NY, 1970.

[367] Jensen, R.E., A dynamic programming algorithm for cluster analysis, *Operations Research Society of America,* 17, 1034-1057, 1969.

[368] Jensen, P.A., Modeling with microcomputers for operations research, *Computers and Operations Research,* 13, 359-366, 1986.

[369] Jensen, P.A. and Bard, J.F., *Operations Research Models and Methods,* Wiley, NY, 2008.

[370] Jeroslow, R.G. and Wang, J., Dynamic programming, integral polyhedra and Horn clause knowledge bases, *ORSA Journal on Computing,* 1(1), 7-19, 1989.

[371] Johnson, D.B., A note on Dijkstra's shortest path algorithm, *Journal of the Association for Computing Machinery,* 20(3), 385-388, 1973.

[372] Jones, L., Willis, R. and Yeh, W.W-G., Optimal control of nonlinear groundwater hydraulics using differential dynamic programming, *Water Resources Research,* 23(11), 2097-2106, 1987.

[373] Judd, K.L., *Numerical Methods in Economics,* MIT Press, Cambridge, MA, 1998.

[374] Kamien, M.I. and Schwartz, N.L., *Dynamic Optimization, the Calculus of Variations and Optimal Control in Economics and Management,* North-Holland, NY, 1981.

[375] Karamouz, M. and Houck, M.H., Comparison of stochastic and deterministic dynamic programming for reservoir operating rules generation, *Journal of the American Water Resources Association,* 23(1), 1-9, 1987.

[376] Karlin, S., The structure of dynamic programming models, *Naval Research Logistics Quarterly,* 2, 285-294, 1955.

[377] Karlin, S. and Shapiro, H.N., *Decision processes and functional equations,* RM-933, The Rand Corporation, Santa Monica, CA, 1952.

[378] Karp, R.M., Dynamic programming meets the principle of inclusion and exclusion, *Operations Research Letters,* 1, 49-51, 1982.

[379] Karp, R.M. and Held, M., Finite-state processes and dynamic programming, *SIAM Journal of Applied Mathematics,* 15, 693-718, 1967.

[380] Kashyap, R.L., Optimization of stochastic finite state systems, *IEEE Transactions on Automatic Control,* 11, 685-692, 1966.

[381] Kátai, Z. and Csiki, A., Automated dynamic programming, *Acta Universitatis Sapientiae,* 1(2), 149-164, 2009.

[382] Katehakis, M.N. and Veinott, A.F., The multi-armed bandit problem: Decomposition and computation, *Mathematics of Operations Research,* 12(2), 262-268, 1987.

[383] Kawai, H. and Katoh, N., Variance constrained Markov decision process, *Journal of the Operations Research Society of Japan,* 30(1), 88-100, 1987.

[384] Kaufmann, A., *Graphs, Dynamic Programming and Finite Games,* Academic Press, NY, 1967.

[385] Kaufmann, A. and Cruon, R., *Dynamic Programming,* Academic Press, NY, 1967.

[386] Kellerer, H., Pferschy, U. and Pisinger, D., *Knapsack Problems,* Springer, 2004.

[387] Kelman, J., Stedinger, J.R., Hsu, E. and Yuan, S.Q., Sampling stochastic dynamic programming applied to reservoir operation, *Water Resources Research,* 26(3), 447-454, 1990.

[388] Kim, C.O., Park, Y. and Baek, J-G., Signal control using adaptive dynamic programming, *Lecture Notes in Computer Science,* 3483, 148-160, 2005.

[389] Kim, S.H. and Jeong, B.H., A partially observable Markov decision process with lagged information, *Journal of the Operational Research Society,* 38(5), 439-446, 1987.

[390] Kim, S.K. and Yeh, W.W-G., A heuristic solution procedure for expansion sequencing problems, *Water Resources Research,* 22(8), 1197-1206, 1986.

[391] Kimeldorf, G. and Smith, F.H., Binomial searching for a random number of multinomially hidden objects, *Management Science,* 25(11), 1115-1126, 1979.

[392] Kira, A., Fujita, T. and Iwamoto, S., Double versus triple competitive processes: non-deterministic model, Mathematical analysis in economics (Japanese) (Kyoto, 2009). *RIMS Kôkyûroku*, to appear.

[393] Kirk, D.E., *Optimal Control Theory, an Introduction,* Prentice Hall, 1970 (also Dover Publications, 2004).

[394] Klein Haneveld, W.K., On the behavior of the optimal value operator of dynamic programming, *Mathematics of Operations Research,* 5(2), 308-320, 1980.

[395] Kolmogorov, A.N. and Fomin S.V., *Elements of the Theory of Functions and Functional Analysis,* Volume 1, (Translation from Russian), Graylock Press, Rochester, NY, 1957.

[396] Kornbluth, J.S.H., Multiple objective dynamic programming with forward filtering, *Computers and Operations Research,* 13, 517-524, 1986.

[397] Kreps, D.M., Decision problems with expected utility criteria, I: Upper and lower convergent utility, *Mathematics of Operations Research,* 2, 45-53, 1977.

[398] Kreps, D.M., Decision problems with expected utility criteria, II: Stationarity, *Mathematics of Operations Research,* 2, 266-274, 1977.

[399] Kreimer, J., Golenko-Ginsburg, D. and Mehrez, A., Allocation of control points in stochastic dynamic programming models, *Journal of the Operational Research Society,* 39(9), 847-853, 1988.

[400] Krozel, J., Lee, C. and Mitchell, J.S.B., Turn-constrained route planning for avoiding hazardous weather, *Air Traffic Control Quarterly,* 14(2), 159-182, 2006.

[401] Kruskal, J.B., Jr., On the shortest spanning tree of a graph and the traveling salesman problem, *Proceedings of the American Mathematical Society,* 7(1), 48-50, 1956.

[402] Krzystofowicz, R., Optimal water supply planning based on seasonal forecasts, *Water Resources Research,* 22(3), 313-321, 1986.

[403] Kucia, A. and Nowak, A., On ϵ-optimal selectors and their application in discounted dynamic programming, *Journal of Optimization Theory and Applications,* 54(2), 289-302, 1987.

[404] Kuhn, K. and Madanat, S., Robust maintenance policies for Markovian systems under model uncertainty, *Computer-Aided Civil and Infrastructure Engineering,* 21, 171-178, 2006.

[405] Kushner, H., *Introduction to Stochastic Control,* Holt, Rinehart and Winston, NY, 1971.

[406] Kydland, F. and Prescott, E., Rules rather than discretion: The inconsistency of optimal plans, *Journal of Political Economy,* 85(3), 473-490, 1977.

[407] Kydland, F. and Prescott, E., Time to build and aggregate fluctuations, *Econometrica,* 50(6), 1345-1371, 1982.

[408] Kydland, F. and Prescott, E., The workweek of capital and its cyclical implications, *Journal of Monetary Economics,* 21(2-3), 343-360, 1988.

[409] Labadie, J., Dynamic programming with the microcomputer. In Kent, A. and Williams, J., (Eds.), *Encyclopedia of Microcomputers,* Vol. 5, 275-338, Marcel Dekker, NY, 1990.

[410] Ladany, S.P., A dynamic model for optimal segmentation of walls built on non-linear slopes, *Engineering Optimization,* 5, 19-26, 1980.

[411] Lageweg, B.J., Lenstra, J.K. and Stougie, L., Stochastic integer programming by dynamic programming, *Statistica Neerlandica,* 39(2), 97-113, 1985.

[412] Langen, H.J., Convergence of dynamic programming models, *Mathematics of Operations Research,* 6, 493-511, 1981.

[413] Larson, R.E., *State Increment Dynamic Programming,* Elsevier, NY, 1968.

[414] Larson, R.E., A survey of dynamic programming computational procedures, *IEEE Transactions on Automatic Control,* 767-774,1967.

[415] Larson, R.E. and Korsak, A.J., A dynamic programming successive approximations technique with convergence proofs, *Automatica,* 6, 245-252, 1970.

[416] Larson, R.E. and Casti, J.L., *Principles of Dynamic Programming,* Part 2, Marcel Dekker, NY, 1982.

[417] Larson, R.E. and Peschon, J., A dynamic programming approach to trajectory estimation, *IEEE Transactions on Automatic Control,* 537-540, 1966.

[418] Lasserre, J.B., A mixed forward-backward dynamic programming method using parallel computation, *Journal of Optimization Theory and Applications,* 45(1), 165-168, 1985.

[419] Lawler, E.L., *Combinatorial Optimization,* Holt, Rinehart and Winston, NY, 1976.

[420] Lawler, E.L., *Efficient implementation of dynamic programming algorithms for sequencing problems,* BW 106/79, Department of Operations Research, Mathematical Center, Amsterdam, The Netherlands, 1979.

[421] Lee, C.Y. and Esogbue, A.O., Optimal procedures for dynamic programs with complex loop structure, *Journal of Mathematical Analysis and Applications,* 119, 300-339, 1986.

[422] Lee, J.M. and Lee, J.H., Approximate dynamic programming strategies and their applicability for process control: A review and future directions, *International Journal of Control, Automation, and Systems,* 2(3), 263-278, 2004.

[423] Lehman, R.S., Dynamic programming and Gaussian elimination, *Journal of Mathematical Analysis and Applications,* 15, 499-501, 1962.

[424] Lembersky, M.R. and Chi, U.H., Weyerhaeuser decision simulator improves timber profits, *Interfaces,* 16, 6-15, 1986.

[425] Lew, A., Richard Bellman's contributions to computer science, *Journal of Mathematical Analysis and Applications,* 119, 90-96, 1986.

[426] Lew, A., N degrees of separation: Influences of dynamic programming on computer science, *Journal of Mathematical Analysis and Applications,* 249, 232-242, 2000.

[427] Lew, A., Canonical greedy algorithms and dynamic programming, *Control and Cybernetics,* 35(3), 2006.

[428] Lew, A. and Mauch, H., *Dynamic Programming: A Computational Tool,* Springer-Verlag, NY, 2007.

[429] Lew, A. and Sniedovich, M., *Control and Cybernetics,* special issue on dynamic programming, 35(3), 2006.

[430] Li, D. and Haimes, Y.Y., New approach for nonseparable dynamic programing problems, *Journal of Optimization Theory and Applications,* 64(2), 311-330, 1990.

[431] Lie, W-N., Lin, T.C-I., Lin, T-C. and Hung, K-S., A robust dynamic programming algorithm to extract skyline in images for navigation, *Pattern Recognition Letters,* 26(2), 221-230, 2005.

[432] Lin, E.Y.H. and Bricker, D.L., Implementing the recursive APL code for dynamic programming, *APL Quote Quad,* 20(4), 239-250, 1990.

[433] Lindsay, G.F. and Bishop, A.B., Allocation of screening inspection effort – A dynamic programming approach, *Management Science,* 10(2), 342-352, 1964.

[434] Liu, Y.A. and Stoller, S.D., Dynamic programming via static incrementalization, *Higher-Order and Symbolic Computation,* 16(1-2), 37-62, 2003.

[435] Ljunquist, L. and Sargent, T., *Recursive Macroeconomics Theory,* MIT Press, Cambridge, MA, 2000.

[436] Love, C.E., Purchase/replacement rules for decaying service facilities, *Computers and Operations Research,* 4, 111-118, 1977.

[437] Love, C.E., Optimal equipment transfer pricing in service systems, *Journal of the Operational Research Society,* 31, 657-666, 1980.

[438] Lovejoy, W.S., Ordered solutions for dynamic programs, *Mathematics of Operations Research,* 12(2), 269-276, 1987.

[439] Lubow, B.C., SDP: Generalized software for solving stochastic dynamic optimization problems, *Wildlife Society Bulletin,* 23(4), 738-742, 1995.

[440] Luus, R., *Iterative Dynamic Programming,* Chapman & Hall, CRC, 2000.

[441] Luus, R., Optimal control of oscillatory systems by iterative dynamic programming, *Journal of Industrial Management Optimization,* 4(1), 2008.

[442] Macalalag, E. and Sniedovich, M., *On the importance of being a sensitive LP package,* Reprint Series No. 3-1991, Department of Mathematics, The University of Melbourne, Melbourne, Australia, 1991.

[443] Maliphant S.A. and Smith D.K., Mini-Risk: Strategies for a simplified board games, *Journal of the Operational Research Society,* 41(1), pp. 9-16, 1990.

[444] Markland, R.E. and Sweigart, J.R., *Quantitative Methods: Applications to Managerial Decision Making,* Wiley, NY, 1987.

[445] Martelli, A. and Montanari, U., *From dynamic programming to search algorithms with functional costs,* Internal Report B75-1, Instituto di Elaborazione Della Informazione, Piza, Italy, 1975.

[446] Martin, J.J., *Bayesian Decision Theory and Markov Chains,* Wiley, NY, 1967.

[447] Martin, R.K., Rardin, R.L. and B.A. Campbell, Polyhedral characterization of discrete dynamic programming, *Operations Research,* 38(1), 127-138, 1990.

[448] Maruyama, Y., Second-order necessary conditions for nonlinear optimization problems in Banach spaces and their application to an optimal control problem, *Mathematics of Operations Research,* 15, 467-482, 1990.

[449] Maruyama, Y., Second-order necessary conditions for an optimal control problem with state constraints, *Bulletin of Informatics and Cybernetics,* 24, 53-69, 1990.

[450] Maruyama, Y., Second-order necessary conditions for nonlinear optimization problems in Banach spaces by the use of Neustadt derivative, *Mathematica Japonica,* 40(3), 509-522, 1994.

[451] Maruyama, Y., On shortest and longest path problems, *Optimization,* 38, 287-299, 1996.

[452] Maruyama, Y., On a associative shortest path problems, *Bulletin of Informatics and Cybernetics,* 29, 67-81, 1997.

[453] Maruyama, Y., An invariant imbedding approach to associative shortest path problems, *Mathematica Japonica,* 50(3), 469-480, 1999.

[454] Maruyama, Y., A duality theorem in multiobjective routing problems with associative path costs. In Takahashi, W. and Tanaka, T., (Eds.), *Nonlinear Analysis and Convex Analysis: Proceedings of the International Conference,* World Scientific Publishers, pp. 245-252, 1999.

[455] Maruyama, Y., Associative shortest and longest path problems, *Bulletin of Informatics and Cybernetics,* 31(2), 147-163, 1999a.

[456] Maruyama, Y., On a negative-equivalency theorem in associative optimal path problems, *Optimization,* 48 , 137-155, 2000.

[457] Maruyama, Y., Duality theorems in parametric associative optimal path problems, *Asia-Pacific Journal of Operations Research,* 17, 149-168, 2000.

[458] Maruyama, Y., Strong representation of a discrete decision process by a bitone sequential decision process. In Takahashi, W. and Tanaka, T., (Eds.), *Nonlinear Analysis and Convex Analysis: Proceedings of the International Conference,* pp. 263-273, Yokohama Publishers,, 2003.

[459] Maruyama, Y., Strong representation theorems for bitone sequential decision processes, *Optimization Methods and Software,* 18(4), 475-489, 2003a.

[460] Maruyama, Y., Associative sequential decision process. In Takahashi, W. and Tanaka, T., (Eds.), *Nonlinear Analysis and Convex Analysis: Proceedings of the International Conference,* pp. 255-264, Yokohama Publishers, 2005.

[461] Maruyama, Y., Weak representation of a discrete decision process by a bitone sequential decision process. In Rubinov, A. and Sniedovich, M., (Eds.), *The International Conference on Optimization: Techniques and Applications, Proceedings of the International Conference,* University of Ballarat (2004), CD-ROM (no. 39).

[462] Maruyama, Y., Positively bitone sequential decision process. In Takahashi, W. and Tanaka, T., (Eds.), *Nonlinear Analysis and Convex Analysis, Proceedings of the International Conference,* pp. 341-353, Yokohama Publishers, 2006.

[463] Maruyama, Y., Strong representation of a discrete decision process by positively/negatively bitone sequential decision process, *Asia-Pacific Journal of Operational Research,* 24(2), 181-202, 2007.

[464] Maruyama, Y., Algorithms for recursive bitone sequential decision process. In Matsuhisa, T. and Koibuchi, H., (Eds.), *Mathematics, Tech-*

nology and Education: Proceedings of Kosen workshop, MTE2008, pp. 55-75, 2008.

[465] Maruyama, Y., Algorithm to obtain optimal policies of recursive bitone sequential decision process. In Takahashi, W., and Tanaka, T., (Eds.), *Nonlinear Analysis and Convex Analysis: Proceedings of the International Conference,* pp. 105-119, Yokohama Publishers, 2009.

[466] Masse, P., *Les Reserves et la Regulation de l'Avenir dans la vie Economique,* Herman, Paris 1946.

[467] Masse, P., *Le Choix des Investissements,* Dunod, Paris, 1959.

[468] Matanaga, G.B. and Marino, M.A., Irrigation planning 2: Water allocation for leaching and irrigation purposes, *Water Resources Research,* 15(3), 679-683, 1979.

[469] Mauch H., DP2PN2Solver: A flexible dynamic programming solver software tool, *Control and Cybernetics,* 35(3), 687-702, 2006.

[470] Mayne, D., A second-order gradient method for determining optimal trajectories of non-linear discrete-time systems, *International Journal of Control,* 3, 85-95, 1966.

[471] McGechan, M.B. and Glasbey, C.A., Combine speed strategies in cereal harvesting. Part 2: Adjustment for weather variability, *Journal of Agricultural Engineering and Research,* 33(1), 13-22, 1986.

[472] McLeod, A.I., The cargo loading or knapsack problem by dynamic programming, *APL Quote Quad,* 14(1), 21-22, 1983.

[473] Mehlamann, A., An approach to optimal recruitment and transition strategies for manpower systems using dynamic programming, *Journal of the Operational Research Society,* 31, 1009-1015, 1980.

[474] Minty, E.F., A comment on the shortest-route problem, *Operations Research,* 5, 724, 1957.

[475] Mitten, L.G., Composition principles for synthesis of optimal multistage processes, *Operations Research,* 12, 414-424, 1964.

[476] Mitten, L.G., Preference order dynamic programming, *Management Science,* 21, 43-46, 1974.

[477] Mitten, L.C. and Nemhauser, G.L., Multistage optimization, *Chemical Engineering Progress,* 59(1), 52-60, 1963.

[478] Mitten, L.C. and Nemhauser, G.L., Optimization of multistage separation processes by dynamic programming, *The Canadian Journal of Chemical Engineering,* 41, 187-194, 1963.

[479] Moore, E.F., The shortest path through a maze. In *Proceedings of an International Synposium on the Theory of Switching,* (Cambridge, Massachusetts, 2-5 April, 1957), pp. 285-292, Harvard University Press, Cambridge, MA, 1959.

[480] Moore, R.G., *Computational Functional Analysis,* Ellis Horwood Limited, Chichester, UK, 1985.

[481] Moore, J.C. and Whinston, A.B., A model of decision-making with sequential information-acquisition Part 1, *Decision Support System,* 2, 285-307, 1986.

[482] Moore, J.C. and Whinston, A.B., A model of decision-making with sequential information-acquisition Part 2, *Decision Support System,* 2, 47-72, 1987.

[483] Moores, B., Dynamic programming in transormer design, *Journal of the Operational Research Society,* 37(10), 967-969, 1986.

[484] Morin, T.L., Pathology of a dynamic programming sequencing algorithm, *Water Resources Research,* 9, 1178-1185, 1973.

[485] Morin, T.L., Solution of some combinatorial optimization problems encountered in water resources development, *Engineering Optimization,* 1, 155-167, 1975.

[486] Morin, T.L., Computational advances in dynamic programming. In Puterman, M.L., (Ed.), *Dynamic Programming and its Applications,* pp. 53-90, Academic Press, NY, 1978.

[487] Morin, T.L., Monotonicity and the principle of optimality, *Journal of Mathematical Analysis and Applications,* 88, 665-674, 1982.

[488] Morin, T.L. and Esogbue, A.O, The imbedded state space approach to reducing dimensionality in dynamic programs of higher dimensions, *Journal of Mathematical Analysis and Applications,* 48, 801-810, 1974.

[489] Morin, T.L. and Marsten, R.E., Branch and bound strategies for dynamic programming, *Operations Research,* 24, 611-627, 1976.

[490] Mozerov, M., Kober, V. and Ovseevich, I.A., A robust dynamic programming algorithm for motion detection and estimation, *Pattern Recognition and Image Analysis,* 17(12), 175-182, 2007.

[491] Murray, D.M. and Yakowitz, S.J., Constrained differential dynamic programming and its application to multireservoir control, *Water Resources Research,* 15, 1017-1027, 1979.

[492] Murray, D.M. and Yakowitz, S.J., The application of optimal control methodology to nonlinear programming problems, *Mathematical Programming,* 21, 331-347, 1981.

[493] Murray, D.M. and Yakowitz, S.J., Differential dynamic programming and Newton's method for discrete optimal control problems, *Journal of Optimization Theory and Applications,* 43, 395-414, 1984.

[494] Nachman, D.C., Optimal stopping with a horizon constraint, *Mathematics of Operations Research,* 5(1), 126-134, 1980.

[495] Naguleswaran, S. and White, L.B., Planning without state space explosion: Petri net to Markov decision process, *International Transactions in Operational Research,* 16(2), 243-255, 2009.

[496] Nandalal, K.D.W. and Bogardi, J.J., *Dynamic Programming Based Operation of Reservoirs: Applicability and Limits,* Cambridge University Press, NY, 2007.

[497] Nemhauser, G.L., *Introduction to Dynamic Programming,* Wiley, NY, 1966.

[498] Nemhauser, G.L. and Ullman, Z., Discrete dynamic programming and capital allocation, *Management Science,* 15, 494-505, 1969.

[499] Nicole, D.M., Parallel solution of sparse one-dimensional dynamic programming problems, *ORSA Journal of Computing,* 2(2), 162-173, 1990.

[500] Nicholson, T.A.J., Finding the shortest route between two points in a network, *The Computer Journal,* 6, 275-280, 1966.

[501] Nilim, A. and El Ghaoui, L., Robust control of Markov decision processes with uncertain transition matrices, *Operations Research,* 53(5), 780-798, 2005.

[502] Nilsson, N.J., *Problem Solving Methods in Artificial Intelligence,* McGraw-Hill, NY, 1971.

[503] Nishimura, N., A general theory of branch-and-bound procedures and its relation to dynamic programming, MSc thesis, Department of Applied Mathematics and Physics, Kyoto University, Kyoto, Japan, 1977.

[504] Noe, N.H. and Ehrenfeld, S., A Bayesian sequential multi-decision problem, *Management Science,* 20(3), 274-281, 1973.

[505] Nollau, V., A stochastic decision model with vector-valued reward, *Optimization,* 16, 733-742, 1985.

[506] Norman, J., *Heuristic Procedures in Dynamic Programming,* Manchester Unity Press, Manchester, 1972.

[507] Norman, J.M. and White, D.J., A method for approximate solutions to stochastic dynamic programming problems using expectations, *Operations Research,* 16, 296-306, 1968.

[508] Norman, J.M. and Shearn, D.C.S., Optimal claiming on vehicle insurance revisited, *Journal of the Operational Research Society,* 31, 181-186, 1980.

[509] Nwannenna, C.C., A computer model for power system control, *Computers and Operations Research,* 14(4), 325-340, 1987.

[510] Ohno, K. and Ichiki, K., Computing optimal policies for controlled tandem queueing systems, *Operations Research,* 35(1), 121-126, 1987.

[511] Olorunniwo, F.O. and Jensen, P.A., Optimal capacity expansion policy for natural gas transmission network – A decomposition approach, *Engineering Optimization,* 6, 13-30, 1982.

[512] Ozden, M., A new algorithm for multi-dimensional dynamic programming problems, *Operations Research Letters,* 2, 84-89, 1983.

[513] Ozden, M., A solution procedure for general knapsack problems with a few constraints, *Computers and Operations Research,* 15(2), 145-155, 1988.

[514] Paul, H., Law, S.S. and Leong, H.K., A method for single facility multiple products scheduling, *Journal of the Operational Research Society,* 31, 825-832, 1980.

[515] Panossian, H., and Bellman, R., Stochastic control systems, *Computers and Mathematics with Applications,* 12A(6), 825-827, 1986.

[516] Papadaki, K.P. and Powell, W.B., Exploiting structure in adaptive dynamic programming algorithms for a stochastic batch service problem, *European Journal of Operational Research,* 142, 108-127, 2002.

[517] Papadimitriou, C.H. and Tsitsiklis, J.N., The complexity of Markov decision processes, *Mathematics of Operations Research,* 12(3), 441-450, 1987.

[518] Parlar, M., Dynamic programming on an electronic spreadsheet, *Computers and Industrial Engineering,* 10(3), 203-213, 1986.

[519] Petersen, E.R. and Taylor, A.J., Probabilistic optimization spreadsheets: PROPS, *Operations Research Letters,* 9, 271-274, 1990.

[520] Piccardi, C. and Soncini-Sessa, R., Stochastic dynamic programming for reservoir optimal control: Dense discretization and inflow correlation assumption made possible by parallel computing, *Water Resources Research,* 27(2), 729-741, 1991.

[521] Piunovskiy A.B., DP in constrained Markov decision processes, *Control and Cybernetics,* 35(3), 645-660, 2006.

[522] Pohl, I., Bi-directional search. In Meltzer, B. and Michie, D., (Eds.), *Machine Intelligence,* 6, 127-140, 1971.

[523] Pollock, S.M. and Smith, R.L., *A formalism of dynamic programming,* Technical Report 85-8, Department of Industrial and Operations Engineering, The University of Michigan, Ann Arbor, MI, 1985.

[524] Pollack M. and Wiebenson, W., Solution of the shortest-route problem – A review, *Operations Research,* 8, 224-230, 1960.

[525] Polster, B., *The Shoelace Book,* Mathematical World (24), AMS, 2006.

[526] Pontryagin, L.S. et al., *The Mathematical Theory of Optimal Processes,* vol. 4. Interscience, 1962.

[527] Pope, D.N. and Curry, G.L., Computerized analytical solutions in dynamic programming, *Computers and Industrial Engineering,* 5, 257-270, 1981.

[528] Pope, D.N., Curry, G.L. and Phillips, D.T., Closed form solutions to nonserial nonconvex quadratic programming problems using dynamic programming, *Journal of Mathematical Analysis and Applications,* 86, 628-647, 1982.

[529] Popova-Zeugmann L., Time Petri nets state space reduction using dynamic programming, *Control and Cybernetics,* 35(3), 721-748, 2006.

[530] Porteus, E., An informal look at the principle of optimality, *Management Science,* 21, 1346-1348, 1975.

[531] Porteus, E., Conditions for characterizing the structure of optimal strategies in infinite-horizon dynamic programs, *Journal of Optimization Theory and Applications,* 36, 419-432, 1982.

[532] Potts, C.N. and Van Wassonhove, L.N., Dynamic programming and decomposition approaches for the single machine total tardiness problem, *European Journal of Operational Research,* 32, 405-414, 1987.

[533] Powell, W.B., *Approximate Dynamic Programming: Solving the Curses of Dimensionality,* Wiley, NY, 2007.

[534] Prastacos, G.P., Optimal sequential investment decisions under conditions of uncertainty, *Management Science,* 29, 118-134, 1983.

[535] Puchinger, J. and Stuckey, P.J., Automating branch-and-bound for dynamic programs, *PEPM'08 Conference,* San Francisco, CA, January 7-8, 2008, .

[536] Puterman, M.L., (Ed.), *Dynamic Programming and its Applications,* Academic Press, NY, 1978.

[537] Ram, B. and Babu, A.J.G., Reduction of dimensionality in dynamic programming based solution methods for nonlinear integer programming, *International Journal for Mathematics and Mathematical Sciences,* 11(4), 811-814, 1988.

[538] Raffensperger, J.F. and Richard, P., Implementing dynamic programs in spreadsheets, *INFORMS Transactions on Education,* 5(2), 25-46, 2005.

[539] Rayward-Smith, V.J., Mckeown, G.P. and Burton, F.W., The general problem solving algorithm and its implementation, *New Generation Computing,* 6, 41-66, 1988.

[540] Richter, K., Stability of the constant cost dynamic lot size model, *European Journal of Operational Research,* 31, 61-65, 1987.

[541] Richter, R., Stochastically maximizing the number of successes in a sequential assignment problem, *Journal of Applied Probability,* 27, 351-364, 1990.

[542] Roberts, S.M., *Dynamic Programming in Chemical Processes,* Academic Press, NY, 1964.

[543] Rom, W.O. and Winter, F.W., New product evaluation using a Bayesian dynamic program, *Journal of the Operational Research Society,* 32, 223-232, 1981.

[544] Roosta, M., On the connectivity of a network, *Journal of Mathematical Analysis and Applications,* 83, 209-211, 1981.

[545] Roosta, M., Routing through a network with maximum reliability, *Journal of Mathematical Analysis and Applications,* 88, 341-347, 1982.

[546] Rosenman, M.A. and Gero, J.S., Heuristic nonserial dynamic programming for large problems, *Engineering Optimization,* 4, 167-178, 1980.

[547] Rosenman, M.A., Bradford, A.D. and Gero, J.S., Postoptimality analysis in dynamic programming, *Engineering Optimization,* 4, 207-214, 1980.

[548] Rosenman, M.A. and Gero, J.S., Reducing the pareto optimal set in multicriteria optimization (with applications to Pareto optimal dynamic programming), *Engineering Optimization,* 8, 189-206, 1985.

[549] Rosenthal, A., Dynamic programming is optimal for certain sequential decision processes, *Journal of Mathematical Analysis and Applications,* 73, 134-137, 1980.

[550] Rosenthal, A., Dynamic programming is optimal for nonserial optimization problems, *SIAM Journal of Computation,* 11, 47-59, 1982.

[551] Ross, I.M., *A Primer on Pontryagin's Principle in Optimal Control,* Collegiate Publishers, Cambridge, MA, 2009.

[552] Ross, S.M., Dynamic programming and gambling models, *Advances in Applied Probability,* 6, 607-621, 1974.

[553] Ross, S.M., *Introduction to Stochastic Dynamic Programming,* Academic Press, NY, 1983.

[554] Rossman, L.A., Reliability constrained dynamic programming and randomized release rules in reservoir management, *Water Resources Research,* 13(2), 247-255, 1977.

[555] Rothblum, U.G., Karni, R. and Gelfand, E., A dynamic programming formulation of a production sequencing problem, *Computers and Industrial Engineering,* 7, 69-75, 1983.

[556] Rudin, W., *Principles of Mathematical Analysis,* (2nd Edition), McGraw Hill, NY, 1964.

[557] Russell, M.J., Moore, R.K. and Tomlinson, M.J., Dynamic programming and statistical modelling in automatic speech recognition, *Journal of the Operational Research Society,* 37, 21-30, 1986.

[558] Sagot, M-F. and Walter, M.E.M.T., (Eds.), *Advances in Bioinformatics and Computational Biology, Lecture Notes in Bioinformatics,* 4643, Springer, 2007.

[559] Sancho, N.G.F., A multi-objective routing problem, *Engineering Optimization,* 10, 71-76, 1986.

[560] Sankoff, D., The early introduction of dynamic programming to computational biology, *Bioinformatics,* 16(1), 41-47, 2000.

[561] Sargent, T., *Dynamic Macroeconomic Theory,* Harvard University Press, Cambridge, MA, 1987.

[562] Satia, J.K. and Lave, R.E. Jr., Markovian decision processes with uncertain transition probabilities, *Operations Research,* 21(3), 728-740, 1973.

[563] Sawaki, K., Piecewise linear dynamic programs with applications, *Journal of the Operations Research Society of Japan,* 23(2), 91-109, 1980.

[564] Schaible S., Survey of fractional programming. In Schaible, S. and Ziemba, T.A., (Eds.), *Generalized Concavity in Optimization and Economics,* pp. 417-440, Academic Press, NY, 1981.

[565] Schaible, S. and Ibaraki, T., Fractional programming, *European Journal of Operational Research,* 12, 325-338, 1983.

[566] Schal, M., An operator-theoretical treatment of negative dynamic programming. In Puterman, M.L., (Ed.), *Dynamic Programming and its Applications,* Academic Press, NY, 1978.

[567] Schal, M., On the chance to visit a goal set infinitely often, *Optimization,* 21(4), 585-592, 1990.

[568] Schrage, L. and Baker, K.R., Dynamic programming solutions of sequencing problems with precedence constraints, *Operations Research,* 26, 444-449, 1978.

[569] Sedgewick, R. and Vitter, J.S., Shortest paths in Euclidean graphs, *Algorithmica,* 1(1), 34-48, 1986.

[570] Seinfeld, J.H. and Lapidus, L., Aspects of forward dynamic programming algorithm, *Industrial and Engineering Chemistry Process Design and Development,* 7(3), 475-478, 1968.

[571] Sengupta, J.K., *Stochastic Programming,* North-Holland, Amsterdam, The Netherlands, 1972.

[572] Sethi S. and Thompson, G.L., *Optimal Control Theory: Applications of Management Science and Economics,* (2nd edition), Kluwer, 2000.

[573] Shapiro, J.F., Dynamic programming algorithms for the integer programming problem I: The integer programming problem viewed as a knapsack type problem, *Operations Research,* 16, 103-121, 1968.

[574] Shapiro, J.F., Turnpike theorems for integer programming problems, *Operations Research,* 18, 432-440, 1970.

[575] Shapiro, J.S. and Wagner, H.M., A finite renewal algorithm for the knapsack and turnpike models, *Operations Research,* 15, 319-341, 1967.

[576] Shoemaker, C.A., Applications of dynamic programming and other optimization methods in pest management, *IEEE Transactions on Automatic Control,* 26, 1125-1132, 1981.

[577] Shoemaker, C.A, and Fan, D.L., Regression dynamic programming for large stochastic reservoir systems, *American Geophysical Union,* Fall Meeting 2001.

[578] Si, J., Barto, A.G., Powell, W.B. and Wunsch, D., (Eds.), *Handbook of Learning and Approximate Dynamic Programming,* Wiley, NY, 2004.

[579] Simms, B.W., Lamarre, B.G. and Jardine, A.K.S., Optimal buy, operate and sell policies for fleet of vehicles, *European Journal of Operational Research,* 15, 183-195, 1984.

[580] Sklenar, J. and Popela, P., Application of general dynamic programming engine, *Proceedings of the 11th International Conference on Soft*

Computing, Mendel 2005, Brno, 2005, pp. 190-195, ISBN 80-214-2961-5.

[581] Sklenar, J. and Popela, P., Generation of prototype dynamic programming solutions, *Proceedings of the 12th International Conference on Soft Computing,* Mendel 2006, Brno, 2006, pp. 157-162, ISBN 80-214-3195-4.

[582] Smart, D.R., *Fixed Point Theorems,* Cambridge University Press, Cambridge, UK, 1974.

[583] Smith, D.K., *Dynamic Programming: A Practical Introduction,* Ellis Horwood, Chichester, UK, 1991.

[584] Smith, D.K., Dynamic programming and inventory management: What has been Learnt in the last generation?, *Proceedings of the 1999 ISIR Workshop on Inventory Management,* Exeter, UK, 2000.

[585] Smith, D.K., *Networks and Graphs: Techniques and Computational Methods,* Horwood Publishing Limited, Chichester, UK, 2003.

[586] Smith, D.K., Decision making and card collecting, *IMA Journal of Management Mathematics,* 16, 89-97, 2005.

[587] Smith, D.K, Dynamic programming and board games: A survey, *European Journal of Operational Research,* 176, 1299-1318, 2007.

[588] Sniedovich, M., Dynamic programming and the principle of optimality: A systematic approach, *Advances in Water Resources,* 1(3), 131-139, 1978a.

[589] Sniedovich, M., Dynamic programming and principles of optimality, *Journal of Mathematical Analysis and Applications,* 65(3), 586-606, 1978b.

[590] Sniedovich, M., Reliability constrained reservoir control problems 1: Methodological issues, *Water Resources Research,* 15(6), 1574-1582, 1979.

[591] Sniedovich, M., A variance constrained reservoir control problem, *Water Resources Research,* 16(2), 271-274, 1980a.

[592] Sniedovich, M., Preference order stochastic knapsack problems: Methodological issues, *Journal of the Operational Research Society,* 31, 1025-1032, 1980b.

[593] Sniedovich, M., Analysis of a preference order assembly line problem, *Management Science,* 27(9), 1067-1080, 1981a.

[594] Sniedovich, M., Analysis of a preference order travelling salesman problem, *Operations Research,* 29(6), 1234-1237, 1981b.

[595] Sniedovich, M., Use of APL in operations research: An interactive dynamic programming model, *APL Quote Quad,* 12(1), 291-297, 1981c.

[596] Sniedovich, M., A class of variance constrained problems, *Operations Research,* 31(2), 338-353, 1983.

[597] Sniedovich, M., C-programming: an outline, *Operations Research Letters,* 4(1), 19-21, 1985a.

[598] Sniedovich, M., A new look at Bellman's principle of optimality, *Journal of Optimization Theory and Applications,* 49(1), 161-176, 1986a.

[599] Sniedovich, M., C-programming and the minimization of pseudolinear and additive concave functions, *Operations Research Letters,* 5(4), 185-189, 1986b.

[600] Sniedovich, M., A class of nonseparable dynamic programming problems, *Journal of Optimization Theory and Applications,* 52(1), 111-121, 1987.

[601] Sniedovich, M., On fractional programming problems in engineering optimization, *Engineering Optimization,* 1987a.

[602] Sniedovich, M., Fractional programming revisited, *European Journal of Operational Research,* 1987b.

[603] Sniedovich, M., A multi-objective routing problem revisited, *Engineering Optimization,* 13, 99-108, 1988.

[604] Sniedovich, M., Analysis of a class of fractional programming problems, *Mathematical Programming,* 43, 329-347, 1989.

[605] Sniedovich, M., *Dynamic Programming,* Marcel Dekker, NY, 1992.

[606] Sniedovich, M., Eureka! Bellman's principle of optimality is valid! In Dror, M., L'Ecuyer, P. and Szidarovszky, F., (Eds.), *Essays on Uncertainy,*, pp. 735-749, Kluwer, 2002.

[607] Sniedovich M., Dynamic programming. In Encyclopedia of Life Support Systems (EOLSS), Eolss Publishers, Oxford ,UK, 2002.

[608] Sniedovich M., OR/MS Games: 1. A neglected educational resource, *INFORMS Transactions on Education,* Vol. 2(3), 86-95, 2002.

[609] Sniedovich M., OR/MS Games: 2. The Towers of Hanoi, *INFORMS Transactions on Education,* Vol. 3(1), 34-51, 2002.

[610] Sniedovich M., OR/MS Games: 3. The Counterfeit coin problem, *INFORMS Transactions in Education,* Vol. 3(2), 32-41, 2003a.

[611] Sniedovich, M., OR/MS Games: 4. The joy of egg-dropping in Braunschweig and Hong Kong, *INFORMS Transactions on Education,* 4(1), 48-64, 2003b.

[612] Sniedovich, M., The Role of games and puzzles in OR/MS education. In Jardim-Goncalves, R., Cha, J. and Steiger-Garcao, A., (Eds.), *Concurrent Engineering: Enhanced Interoperable Systems,*, pp. 1235-1240, CRC Press, 2003.

[613] Sniedovich, M., On-line educationally rich modules: An overview of IFORS tutORial project. In Jardim-Goncalves, R., Cha, J. and Steiger-Garcao, A., (Eds.), *Concurrent Engineering: Enhanced interoperable Systems,* pp. 1207-1211, CRC Press, 2003.

[614] Sniedovich, M., Dynamic Programming Revisited, Challenges and Opportunities. *Proceedings of ICOTA 6,* paper 41, pp. 1-11, Dec 9 -11, 2004, Ballarat, VIC, Australia.

[615] Sniedovich, M., Towards an AoA-Free Courseware for the Critical Path Method, *INFORMS Transactions on Education,* 5(2), 2005.

[616] Sniedovich, M., Dijkstra's Algorithm: the Dynamic Programming Connexion, *Journal of Control and Cybernetics,* 35(3), 599-620, 2006.

[617] Sniedovich, M., Dynamic programming: Basic concepts. In *Wiley Encyclopedia of Operations Research and Management Science,* (EORMS), Wiley, NY, in press.

[618] Sniedovich, M. and Lew, A., Dynamic Programming: An overview, *Journal of Control and Cybernetics,* 35(3), 513-533, 2006.

[619] Sniedovich, M. and Nielsen, P.A., A heuristic solution procedure for a joint reservoir control capacity expansion problem, *Water Resources Research,* 19(1), 15-20, 1983.

[620] Sniedovich, M. and Voß, S., The Corridor method: A dynamic programming inspired metaheuristic, *Journal of Control and Cybernetics,* 35(3), 551-578, 2006.

[621] Sobel, J.M., Ordinal dynamic programming, *Management Science,* 21, 967-975, 1975.

[622] Sobel, J.M., Reservoir management models, *Water Resources Research,* 11, 767-776, 1975.

[623] Soliman, S.A., Christensen, G.S. and Abdel-Halim, M.A., Optimal operation of multireservoir power system, *Journal of Optimization Theory and Applications,* 49, 449-461, 1986.

[624] Speyer, J.L. and Jacobson, D.H., *Primer on Optimal Control Theory,* SIAM, 2010.

[625] Stanfel, L.E., A recursive Lagrangian method for clustering problems, *European Journal of Operational Research,* 27, 332-342, 1986.

[626] Steinberg, E. and Parks, M.S., A preference order dynamic program for a knapsack problem with stochastic reward, *Journal of the Operational Research Society,* 30(2), 141-147, 1979.

[627] Steiner, E.W., Dynamic programming using local optimality conditions for action elimination, PhD thesis, University of Edinburgh, UK, 2000.

[628] Steudel, H.J., Generating pallet loading patterns: A special case of the two dimensional cutting stock problem, *Management Science,* 25(10), 997-1004, 1979.

[629] Steudel, H.J. and Ballakur, A., A dynamic programming based heuristic for machine grouping in manufacturing cell formation, *Computers and Industrial Engineering,* 12(3), 215-222, 1987.

[630] Stokey, N.L., Lucas, R.E. Jr. and Prescott, E.C., *Recursive Methods in Economic Dynamics,* Harvard University Press, Cambride, MA, 1989.

[631] Striebel, C., Sufficient statistics in the optimum control of stochastic systems, *Journal of Mathematical Analysis and Applications,* 12, 576-592, 1965.

[632] Stuckless, T., Brouwer's fixed point theorem: Methods of proof and applications, MSc thesis, Department of Mathematics, Simon Fraser University, Canada, 2003.

[633] Sung, C.S. and Chang, S.H., A capacity planning model for a multi-product facility, *Engineering Optimization,* 10, 263-270, 1987.

[634] Sutherland, W.R.S., Optimality in transient Markov chains and linear programming, *Mathematical Programming,* 18, 1-6, 1980.

[635] Sutton, R.S., Integrated architectures for learning, planning, and reacting based on approximating dynamic programming, *Proceedings of the 7th International Conference on Machine Learning,* 216-224, 1990.

[636] Sutton, R.S. and Barto, A.G., *Reinforcement Learning: An Introduction,* MIT Press, 1998.

[637] Sutherland, W.R.S., Wolkowicz, H. and Zeidan, V., An explicit linear solution for the quadratic dynamic programming problem, *Journal of Optimization Theory and Applications,* 58(2), 319-330, 1988.

[638] Szidarovszky, F. and Duckstein, L., Dynamic multiobjective optimization: A framework with application to regional water and mining management, *European Journal of Operational Research,* 24, 305-317, 1986.

[639] Szidarovsky, F., Gershon, M. and Bardossy, A., Application of multiobjective dynamic programming to regional natural resource management, *Applied Mathematics and Computation,* 24, 281-301, 1987.

[640] Tanga, L., Xuana, H., and Liub, J., Hybrid backward and forward dynamic programming based Lagrangian relaxation for single machine scheduling, *Computers and Operations Research,* 34, 2625-2636, 2007.

[641] Tarvainent, K., An elimination condition to check the validity of the principle of optimality, *Computers and Operations Research,* 23(2), 183-92, 1996.

[642] Tauxe, G.W., Inman, R.R. and Mades, D.M., Multiobjective dynamic programming: A classic problem redressed, *Water Resources Research,* 15, 1398-1402, 1979.

[643] Tauxe, G.W., Inman, R.R. and Mades, D.M., Multiobjective dynamic programming with applications to a reservoir, *Water Resources Research,* 15, 1403-1408, 1979.

[644] Teixeira, A.D.S. and Marino, M.A., Coupled reservoir operation-irrigation scheduling by dynamic programming, *Journal of Irrigation and Drainage Engineering,* 128(2), 63-73, 2002.

[645] Tereso, A.P., Mota J.R.M. and Lameiro R.J.T., Adaptive resource allocation to stochastic multimodal projects: A distributed platform implementation in Java, *Control and Cybernetics,* 35(3), 661-686, 2006.

[646] Terry, L.A., Pereira, M.V.F., Araripe Neto, T.A., Silva, L.F.C.A. and Sales, P.R.H., Coordinating the energy generation of the Brazilian national hydrothermal electrical generating system, *Interfaces,* 16, 16-38, 1986.

[647] Thiemann, J.G.F., *Analytic spaces and dynamic programming: A measure-theoretic approach,* CWI Tracts, CWI, Amsterdam, 1985.

[648] Thiriet, L. and Deledicq, A., Applications of suboptimality in dynamic programming, *Industrial and Engineering Chemistry,* 60(2), 23-28, 1968.

[649] Thomas, L.C., The best banking strategy when playing the Weakest Link, *Journal of the Operational Research Society,* 54(7), 747-750, 2003.

[650] Thomas, M.E., A survey of the state of the art in dynamic programming, *AIIE Transactions,* 8, 56-69, 1976.

[651] Thompson, M.G. and Granum, E., Dynamic programming inference of Markov networks from finite sets of sample strings, *IEEE Transactions on Pattern Analysis and Machine Intelligence,* 8(4), 491-501, 1986.

[652] Tracz, G.S., A selected bibliography on the application of optimal control theory to economic and business systems, management science, and operations research, *Operations Research,* 16(1), 174-186, 1968.

[653] Trezos, T. and Yeh, W.W-G., Use of stochastic dynamic programming for reservoir management, *Water Resources Research,* 23(6), 983-996, 1987.

[654] Troffaes, M.C.M., Learning and optimal control of imprecise Markov decision processes by dynamic programming using the imprecise Dirichlet model. In Lopez-Diaz, M., Gil, M.A., Grzegorzewski, P., Hyrniewicz, O. and Lawry, J., (Eds.), *Soft Methodology and Random-Information Systems,* pp. 141-148, Springer, Berlin, Germany, 2004.

[655] Troffaes, M.C.M., Optimality, uncertainty, and dynamic programming with lower previsions, PhD thesis, University of Gent, Belgium, 2005.

[656] Tsurusak, T. and Iwamoto, S., Optimal stopping under range criterion. Some problems under uncertainty and mathematical decision (Japanese) (Kyoto, 2004). *RIMS Kôkyûroku* No. 1373, 229-237, 2004.

[657] Tsurusaki, K. and Iwamoto, S., When to stop range process – An expanded state space approach. In Negoita et al., (Eds.), *Proceedings of The 8th International Conference on Knowledge-Based Intelligent Information and Engineering Systems,* (KES 2004), Wellington, New Zealand, September 20-24, 2004. *Lecture Notes in Artificial Intelligence,* (LNIA), 3214, 1201-1207, 2004.

[658] Turgeon, A., Optimal operation of multireservoir power systems with stochastic inflows, *Water Resources Research,* 16, 275-283, 1980.

[659] Ueno, T. and Iwamoto, S., Maximizing order probabilities on controlled Markov chains. Mathematics of decision-making under uncertainty (Japanese) (Kyoto, 2002). *RIMS Kôkyûroku* No. 1306, 201-209, 2003.

[660] Ueno, T. and Iwamoto, S., Maximizing order probabilities on controlled Markov chains. Mathematical decision-making under uncer-

tainty (Japanese) (Kyoto, 2002). *RIMS Kôkyûroku* No. 1306, 201-209, 2003.

[661] Ueno, T. and Iwamoto, S., On past accumulation values and future threshold values in stochastic optimization. Perspective and problems for dynamic programming with uncertainty (Japanese) (Kyoto, 2001). *RMIS Kôkyûroku* No. 1207, 79-100, 2001.

[662] Van Dawen, R., Finite state dynamic programming with the total reward criterion, *Zeitschrift fur Operations Research,* 30, 1-14, 1986.

[663] Van, C.L. and Dana, R.A., *Dynamic Programming in Economics,* Springer-Verlag, NY, 2003.

[664] Van der Vet, R.P., On the Bellman principle for decision problems with random decision policies, *Journal of Engineering Mathematics,* 10(2), 107-114, 1976.

[665] Veinott, A.F. Jr., On finding optimal policies in discrete dynamic programming with no discounting, *Annals of Mathematical Statistics,* 37, 1284-1295, 1966.

[666] Venezia, I., The optimal frequency of information purchases, *European Journal of Operational Research,* 4, 118-123, 1980.

[667] Verdu, S. and Poor, H.V., Abstract dynamic programming models under commutativity conditions, *SIAM Journal of Control and Optimization,* 25(4), 990-1006, 1987.

[668] Vidal, R.V.V., A global maximum principle for discrete-time control problems, *Engineering Optimization,* 10, 77-84, 1986.

[669] Vidal, R.V.V., On the sufficiency of the linear maximum principle for discrete-time control problems, *Journal of Optimization Theory and Applications,* 54(3), 583-589, 1987.

[670] Villarreal, B. and Karwan, M.H., Multicriteria dynamic programming with an application to the integer case, *Journal of Optimization Theory and Applications,* 38, 43-69, 1982.

[671] Viswanathan, B., Aggarwal, V. and Nair, K.P.K., Multiple criteria Markov decision processes, *TIMS Studies in the Management Sciences,* 6, 263-272, 1977.

[672] Wald, A., *Sequential Analysis,* Wiley, NY, 1947.

[673] Wakuta, K., The Bellman's principle of optimality in the discounted dynamic programming, *Journal of Mathematical Analysis and Applications,* 125, 213-217, 1987.

[674] Walters, G.A., The design of the optimal layout for a sewer network, *Engineering Optimization,* 9, 37-50, 1985.

[675] Walters, G.A., Generating the optimum tree network with nonlinear flow dependent arc costs, *Engineering Optimization,* 9, 121-126, 1985.

[676] Walters, G.A. and Templeman, A.B., Non-optimal dynamic programming algorithms in the design of minimum cost drainage systems, *Engineering Optimization,* 4, 139-148, 1979.

[677] Wang, C.L., Functional equation approach to inequalities III, *Journal of Mathematical Analysis and Applications,* 80, 31-35, 1981.

[678] Wang, C.L., The principle and models of dynamic programming, *Journal of Mathematical Analysis and Applications,* 130, 287-308, 1986.

[679] Wang, C.L., The principle and models of dynamic programming II, *Journal of Mathematical Analysis and Applications,* 135, 268-283, 1988.

[680] Wang, C.L., The principle and models of dynamic programming III, *Journal of Mathematical Analysis and Applications,* 135, 284-296, 1988.

[681] Wang, D. and Adams, B.J., Optimization of real-time reservoir operations with Markov decision processes, *Water Resources Research,* 22(3), 345-352, 1986.

[682] Waterman, M.S. and Byers, T.H., A dynamic programming algorithm to find all solutions in a neighborhood of the optimum, *Mathematical Biosciences,* 77, 179-188, 1985.

[683] Warwick, J. and Phelps, B., An application of dynamic programming to pattern recognition, *Journal of the Operational Research Society,* 37, 35-40, 1986.

[684] Werner, M., A timed Petri net framework to find optimal IRIS schedules, *Control and Cybernetics,* 35(3), 703-720, 2006.

[685] White, C.C. and Eldeib, H.K., Markov decision processes with imprecise transition probabilities, *Operations Research,* 42(4), 739-749, 1994.

[686] White, C.C. and Scherer, W.T., Reward revision and the average Markov decision process, *OR Spectrum,* 9, 203-211, 1987.

[687] White, D.J., Dynamic programming, Markov chains, and the method of successive approximations, *Journal of Mathematical Analysis and Applications,* 6, 373-376, 1963.

[688] White, D.J., *Dynamic Programming,* Holden Day, San Francisco, CA, 1969.

[689] White, D.J., Dynamic programming and probabilistic constraints, *Operations Research,* 22, 654-664, 1974.

[690] White, D.J., *Finite Dynamic Programming,* Wiley, Chichester, UK, 1978.

[691] White, D.J., Multi-objective interactive programming, *Journal of the Operational Research Society,* 31, 517-523, 1980.

[692] White, D.J., Multi-objective infinite-horizon discounted Markov decision processes, *Journal of Mathematical Analysis and Applications,* 89, 639-647, 1982.

[693] White, D.J., The set of efficient solutions for multiple objective shortest path problems, *Computers and Operations Research,* 9, 101-107, 1982.

[694] White, D.J., The determination of approximately optimal policies in Markov decision processes by the use of bounds, *Journal of the Operational Research Society,* 33, 253-259, 1982.

[695] White, D.J., *Optimality and Efficiency,* Wiley, NY, 1982.

[696] White, D.J., Monotone value iteration for discounted finite Markov decision processes, *Journal of Mathematical Analysis and Applications,* 109, 311-324, 1985.

[697] White, D.J., Approximating Markov decision processes using expected state transitions, *Journal of Mathemarical Analysis and Applications,* 107, 167-181, 1985.

[698] White, D.J., Real applications of Markov decision processes, *Interfaces,* 15, 73- 83, 1985.

[699] White, D.J., Mean variance, and probabilistic criteria in finite Markov decision processes: A survey, *Journal of Optimization Theory and Applications,* 56(1), 1-29, 1988.

[700] White, D.J. and Kim, K., Two solution procedures for solving vector criterion Markov decision processes, *Large Scale Systems,* 1, 129-140, 1980.

[701] Whittle, P., *Optimization Over Time, Dynamic Programming and Stochastic Control,* Volume 1, Wiley, NY, 1982.

[702] Wilbaut, C., Hanafi, S., Freville, A. and Balev, S., Tabu search: Global intensification using dynamic programming, *Control and Cybernetics,* 35(3), 579-598, 2006.

[703] Wilson, A.J., Britch, A.L. and Templeman, A.B., The optimal design of drainage systems, *Engineering Optimization,* 1, 111-123, 1974.

[704] Winston, W., A stochastic game model of a weapons development competition, *SIAM Journal of Control and Optimization,* 16(3), 411-419, 1978.

[705] Winston, P.H., *Artificial Intelligence,* (2nd Edition), Addison-Wesley, Reading, MA, 1984 (p. 87,112).

[706] Winston, W.L., *Operations Research Applications and Algorithms,* (4th Edition), Brooks/Cole, Belmont, CA, 2004.

[707] Witsenhausen, H.S., A minimax control problem for sampled linear systems, *IEEE Transactions on Automatic Control,* AC-3(1), 5-21, 1968.

[708] Wolfram, S., *Mathematica,* Addison-Wesley, NY, 1988

[709] Womer, N.K., Gulledge T.R., Jr. and Tarimcilar, M.M., A comparison of several dynamic programming models of the made-to-order production situation, *Operations Research Letters,* 5, 87-92, 1986.

[710] Wong, P.J. and Luenberger, D.G., Reducing the memory requirements of dynamic programming, *Operations Research,* 16, 1115-1125, 1968.

[711] Wong, P.J., A new decomposition procedure for dynamic programming, *Operations Research,* 18, 119-131, 1970.

[712] Wong, P.J., An approach to reducing the computing time for dynamic programming, *Operations Research,* 18, 181-185, 1970.

[713] Yakowitz, S.J., *Mathematics of Adaptive Control Processes,* Elsevier, NY, 1969.

[714] Yakowitz, S.J., Dynamic programming applications in water resources, *Water Resources Research,* 18, 673-696, 1982.

[715] Yakowitz, S.J., Convergence rate analysis of the state increment dynamic programming method, *Automatica,* 19, 53-60, 1983.

[716] Yakowitz, S.J., The stagewise Khun-Tucker condition and differential dynamic programming, *IEEE Transactions on Automatic Control,* 31(1), 25-30, 1986.

[717] Yakowitz, S.J. and Rutherford, B., Computational aspects of discrete-time optimal control, *Applied Mathematics and Computation,* 15, 29-45, 1984.

[718] Yao, F.F., Speed-up in dynamic programming, *SIAM Journal of Algorithms and Discrete Methods,* 3, 532-540, 1982.

[719] Zhan, F.B., Three fastest shortest path algorithms on real road networks: Data structures and procedures, *Journal of Geographic Information and Decision Analysis,* 1(1), 69-82, 1997.

[720] Zietz, J., Dynamic programming: An introduction by example, *The Journal of Economic Education,* 38(2), 165-186, 2007.

[721] Zufryden, F.S., A dynamic programming approach for product selection and supermarket shelf-space allocation, *Journal of the Operational Research Society,* 37, 413-422, 1986.

[722] Zufryden, F.S., New computational results with a dynamic programming approach for product selection and supermarket shelf-space allocation, *Journal of the Operations Research Society,* 38(2), 201-203, 1987.

[723] Zuo, Z.Q. and Wu, C.P., Successive approximation technique for a class of large scale NLP problems and its application to dynamic programming, *Journal of Optimization Theory and Applications,* 62(3), 515-527, 1989.

[724] Zwick, U., Exact and approximate distances in graphs – A survey. *Proceedings of the 9th Annual European Symposium on Algorithms,* (ESA'01), pp. 33-48, London, 2001.

Mathematics

Incorporating a number of the author's recent ideas and examples, **Dynamic Programming: Foundations and Principles, Second Edition** presents a comprehensive and rigorous treatment of dynamic programming. The author emphasizes the crucial role that modeling plays in understanding this area. He also shows how Dijkstra's algorithm is an excellent example of a dynamic programming algorithm, despite the impression given by the computer science literature.

New to the Second Edition

- Expanded discussions of sequential decision models and the role of the state variable in modeling
- A new chapter on forward dynamic programming models
- A new chapter on the Push method that gives a dynamic programming perspective on Dijkstra's algorithm for the shortest path problem
- A new appendix on the Corridor method

Taking into account recent developments in dynamic programming, this edition continues to provide a systematic, formal outline of Bellman's approach to dynamic programming. It looks at dynamic programming as a problem-solving methodology, identifying its constituent components and explaining its theoretical basis for tackling problems.

CRC Press
Taylor & Francis Group
an **informa** business
www.crcpress.com

6000 Broken Sound Parkway, NW
Suite 300, Boca Raton, FL 33487
270 Madison Avenue
New York, NY 10016
2 Park Square, Milton Park
Abingdon, Oxon OX14 4RN, UK

DK2356

ISBN: 978-0-8247-4099-3
90000
9 780824 740993