10. DYNAMIC PROGRAMMING

10.1 INTRODUCTION

A basic characteristic of simplex-based optimization methods--linear programming and its extensions, such as integer programming and piecewise linearization--is that all the necessary constraints are combined into a single system of equations which are solved simultaneously. Dynamic Programming (DP) is a very different approach to optimization. It has as its basic strategy the decomposition of a problem into a series of series of sub-problems that are solved sequentially--a technique that has more in common with simulation than with LP.

The principal advantage of DP over LP is that non-linearity in either objective or constraints causes no difficulty and, in fact, even poorly behaved functions with discontinuities can be solved.

The principal disadvantages of DP include:

1. The "curse of dimensionality" which causes the computational burden to become unmanageable as the number of state variables increases to more than two or three.

2. Most DP problems have to be defined in terms of discrete steps rather than continuous variables, thereby requiring a large number of steps (time, space, flow rates, etc.) to adequately model the real system. Hence, DP reservoir operating models will define storage in a reservoir (the state of the system) in terms of an array of discrete ranges.

3. Finally, a disadvantage to the DP modeler is that each new problem must be expressed in terms of a recursive equation that defines the connection between the sub-problems into which the system is being decomposed. In effect, this requires the derivation of a special algorithm for most types of problems rather than obtaining a solution by use of existing software (such as the simplex algorithm that will solve any LP problem). (There have, however, been some attempts to develop generalized software for a particular problem type. For example, Labadie (1989) has developed DP software for solving resource allocation problems (including reservoir operation) called CSUDP, which is now available for microcomputers.)

10.2 DP CONCEPTS

The principal advantage of DP over techniques that simulate a system with exhaustive search type approaches derive from the principle of optimality developed by Richard Bellman:

> *No matter in what state of what stage one may be, in order for a policy to be optimal, one must proceed from that state and stage in an optimal manner.*

A corollary that follows from this supposedly simple idea is:

No matter in what state or stage one might be, in order for a policy to be optimal one had to arrive in that state and stage in an optimal manner.

A corollary which follows from this supposedly simple idea is that no matter in what state or stage one might be, in order for a policy to be optimal one had to arrive in that state and stage in an optimal manner.

Dynamic programming requires that a problem be defined in terms of state variables, stages within a state (the basis for decomposition), and a recursive equation which formally expresses the objective function in a manner that defines the interaction between state and stage. The most intuitive way to decompose problem into stages is on the basis of time. For example, consider a reservoir that is modeled with monthly time steps. A recursive equation could be derived from a mass balance equation which calculates the state of the system (storage) at stage t+1 as a function of stage, inflow data, and release (decision variable) during stage t. The form of this DP function will appear to be quite different from an LP type mass balance function, however, because it must also include the objective function. Most recursive equations will consist of two terms: the benefit or cost associated with the decision to be made in the current stage, $g_j(x_j)$, and a term which includes the total of all benefits or costs from optimal decisions at previous stages if one is proceeding forward $[f_{j-1}(s_j - x_j)]$ or all future stages if proceeding backwards $[f_{j+1}(s_j - x_j)]$ along stage index j (where s_j is the state of the system at stage j). The complete recursive equation might then be:

$$f_j(s_j) = \text{maximum } [g_j(x_j) + f_{j+1}(s_j - x_j)] \quad \text{(if proceeding backward)} \qquad ...[10.1]$$

$$f_j(s_j) = \text{maximum } [g_j(x_j) + f_{j-1}(s_j - x_j)] \quad \text{(if proceeding forward)} \qquad ...[10.2]$$

for which the decision x_j at any stage must be selected from a feasible set of possible x_j for each state s_j. Dynamic programming can best be learned by studying examples of various types of problems. A simple example is presented in the following section.

10.3 A HYDROPOWER DESIGN PROBLEM

Consider a hydropower problem in which three independent sites for dams are possible. A total budget of 3 units of money are available for capital investment at any or all of the sites. At each site 4 decisions are possible: either build nothing or select from 3 possible scales of project. The estimated benefits at each site and size of project are shown in Table 10.1. The objective is to select the optimal size of dam at each site that maximizes return subject to the budget limitation.

Note that in this problem the stage is not time but rather the dam site j. The state of the system will be defined as the budget remaining (s_j) for possible use at stage j. The decision variable x_j will be the scale of project selected at site j.

Table 10.1: Hydropower Benefits of Alternative Dam Sites

Benefit Function	State (Site)		
	1	2	3
$g_j(0)$	0	0	0
$g_j(1)$	2	1	3
$g_j(2)$	4	5	5
$g_j(3)$	6	6	6

The recursive equation (proceeding backwards) can be defined as:

$$f_j(s) = \max [g_j(x_j) + f_{j+1}(s_j - x_j)] \quad \text{...[10.3]}$$

The solution can be displayed in tabular form as shown, stage-by-stage, in Tables 10.2 through 10.4:

Table 10.2: Stage 3 Tableau

Stage 3	$\max [g_3(x_3)]$			
State s_3	g_3	x_3	max g_3	comment
0	0	0	0	
1	3	1	3	$x_3^* = 1$
2	5	2	5	
3	6	3	6	

Table 10.3: Stage 2 Tableau

Stage 2	$\max [g_2(x_2) + f_3(s_2 - x_2)]$						
	x2						
State s_2	0	1	2	3	max g_2	x_2	comment
0	0				0	0	
1	3	1			3	0	
2	5	1 + 3	5		5	0, 2	
3	6	1 + 5	5 + 3	6	8	2	$x_2^* = 2$

Table 10.4: Stage 1 Tableau

Stage 1	$\max [g_1(x_1) + f_2(s_1 - x_1)]$						
	x_1						
State s_1	0	1	2	3	max g_1	x_1	comment
3	8	2 + 5	4 + 3	6	8	0	$x_1^* = 0$

The optimal solution is x = (0,2,1).

10.4 PROBLEMS

1. Solve the canal problem described in Loucks' Exercise 2-13.

2. Solve the investment timing problem presented in Chapter 7 by using DP.