APPLIED SYSTEMS ANALYSIS

Engineering Planning and Technology Management

Richard de Neufville
Massachusetts Institute of Technology

McGraw-Hill Publishing Company
New York St. Louis San Francisco Auckland Bogotá Caracas Hamburg
Lisbon London Madrid Mexico Milan Montreal New Delhi Oklahoma City
Paris San Juan São Paulo Singapore Sydney Tokyo Toronto

(*c*) STOMP is offered an overseas franchise in nameplates, which would allow STOMP to sell 15 more cases of nameplates each week. The price of the franchise is \$2250/wk. Should STOMP buy the franchise?

6.14. *Gravel Company*
A gravel company has contracts to supply gravel to four construction projects in Massachusetts:

1. A highway in Berkshire county (1500 tons).
2. A shopping mall in Salem (1000 tons).
3. A runway extension at Logan Airport (1700 tons).
4. An urban development project in Worcester (500 tons).

The company owns three gravel pits in New Hampshire, in New York, and in Connecticut. The marginal cost of producing and shipping a ton of gravel from pit to user is about:

	Pit		
Client	**NH**	**NY**	**CT**
1	\$2.20	\$1.20	\$1.70
2	1.60	2.20	1.90
3	2.50	3.10	2.70
4	2.10	2.00	1.80

No more than 40% of any project's gravel may come from the Connecticut pit, which contains a fair amount of silt. Only 800 tons of gravel are currently available at the New York pit. The objective is to minimize cost.
(*a*) What are the constraint equations?
(*b*) Indicate what sensitivity output or other data from your LP will answer each of the following questions:
  (i) What is the most we should pay for offshore dredging of gravel for the Logan project?
  (ii) How much more would it cost to require that 50 tons of the New York gravel be used on the Salem project?
  (iii) What is the maximum price at which Connecticut gravel would be used for the Berkshire project?
  (iv) How much of the 800 tons of New York gravel will be unused?

CHAPTER 7 DYNAMIC PROGRAMMING

7.1 CONCEPT

Dynamic programming is a powerful method of optimization which overcomes the three major drawbacks of linear programming. It can be applied to problems that

- are nonlinear.
- have nonconvex feasible regions.
- have discontinuous variables.

These substantial advantages are obtained at a significant price, however. Dynamic programming is limited to dealing with problems with

- relatively few constraints.

Dynamic programming complements linear programming. It works well on the nonlinear, nonconvex problems that are impossible for linear programming, but cannot handle the large number of constraints that linear programming deals with routinely. In practical situations, which may actually involve both nonconvex feasible regions and many constraints, analysts will have to make approximations to fit either the linear or dynamic programming approach. They thus need to be familiar with their relative strengths.

Conceptually, dynamic programming represents a completely different approach to optimization than linear programming. Linear programming is based on the use of optimality conditions. As described in Section 5.3, the consequence of linearity is that we can define in advance the nature of the optimum (it is at a corner point, from which all departures worsen the objective function). Linear programming then consists of an organized search through a set of possible solutions until the optimality criterion is met. Dynamic programming, however, has no optimality criterion we can define in advance.

Dynamic programming is based on the concept of enumeration. The basic idea is simple: we systematically enumerate possible solutions, calculate how well each performs and select the best. Since we enumerate the possible solutions, it does not matter if the feasible region is convex or not—or if it is linear. Enumeration is the principle that gives dynamic programming the power to deal with these nonlinear, nonconvex problems that linear programming cannot.

Dynamic programming carries out this approach in a special way, based on some specific characteristics of the problem that it assumes to hold. These conditions, presented in detail in Section 7.2, are crucial because the enumerative approach to optimization is, for any reasonable problem in practice, computationally impossible if we try to list all the possible solutions.

To appreciate the advantages of dynamic programming it is important to understand that a complete evaluation of all the possible designs of a system is simply unthinkable for the foreseeable future. Even the fastest modern computers would be unable to calculate the performance of all the solutions to a relatively simple system.

The difficulty with a complete enumeration is that the number of possible solutions is an exponential function of the number of possible decisions, the number of values these may have, and the number of periods over which the system is considered. The general rule is

$$\text{Maximum number of Possible Designs} = \left[(\text{Number of Values})^{(\text{Number of Variables})}\right]^{(\text{Number of Periods})}$$

For the design of facilities at particular sites, this formula can be stated more specifically as

$$\text{Possible Designs} = \left[(\text{Sizes})^{(\text{Locations})}\right]^{(\text{Periods})}$$

This number is huge for even relatively small problems, and astronomical for really significant problems such as the design of a regional telephone network; the organization of a distribution system for a major company; et cetera. Even supercomputers would require weeks if not months to evaluate individually each of all the possible configurations of such systems (see box).

The special approach used by dynamic programming is *implicit enumeration*. It is said to be implicit because it considers all possible design

Practical Impossibility of Complete Enumeration

Consider first a simple problem: The design of ten facilities, such as warehouses in a distribution system, each of which could be in any one of ten sizes. The total number of possible combinations is thus 10^{10}: The first variable has 10 possible levels, each of which combines with 10 for the second giving 10×10 combinations for these two, then each of these has ten more for the third variable, giving $10 \times 10 \times 10$, and so on.

Suppose that a computer could identify and list these possible designs at the rate of 100,000 per second, which would be a very good speed. The complete enumeration of all the possible designs would then take

$$\text{Time Required} = 10^5\text{s} = \text{More than a day!}$$

If we were interested in designing a substantial system, such as the water distribution system for the Salt Lake City area as described in Section 5.11, the problem would be very much larger. It might thus have 30 possible locations at a minimum, at least 5 possible sizes for each, and be concerned with the evolution over 3 periods. The total number of logical combinations is then

$$\text{Possible designs} = \left[(5)^{30}\right]^3 \sim 10^{60}$$

If we had a super super computer of the future, 100,000 times as fast as the preceding one, thus doing 10^{10} evaluations a second, the time required to enumerate all the possible combinations would be on the order of 10^{40} centuries!

combinations in principle, but not in fact. Based on certain crucial assumptions about a problem (described in Section 7.2), it systematically eliminates many sets of possible combinations because they can be shown to be inferior. In practice, implicit enumeration carries out the elimination of possible solutions in a series of stages. Implicit enumeration is thus a repetitive process of optimization that identifies and eliminates whole sets of inferior solutions before they have to be considered individually in detail.

The following example illustrates the concept of implicit enumeration. Consider an automobile trip a person might take across the United States, from San Francisco to Washington for example. This is too long to be done in a day, it must be done in stages, let us say 6 days. As the country is wide enough, the traveler can choose many different destinations each day, for example 10. The question is, what is the best route to take?

In general, the answer to this problem is far from obvious. The best route, for example the fastest or the least expensive, depends on distance; difficulties along the way, such as mountains or poor roads; weather conditions, such as snow or heat; congestion, which may reduce speed; administrative or legal restraints,

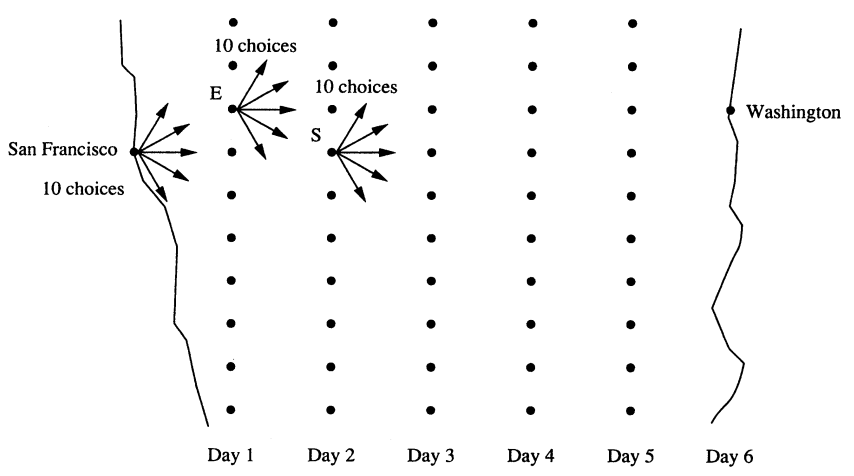


FIGURE 7.1
Example cross-country drive of six days, with 10 possible destinations for each intermediate stage.

such as enforced speed limits; and so on. This example in fact is representative of a large number of practical problems such as those of routing telephone calls or computer messages, or the scheduling of trucks and trains.

Figure 7.1 maps the problem. As shown, the driver has 10 choices each day. There are thus 10^5 possible routes in all: one route to reach any point at the end of the first day, say *E* (Elko, Nevada); thus 10 ways to reach any point at the end of the second day, say *S* (Salt Lake City, Utah)—that through *E* and each of the other nine points for day 1; and so on until Washington.

Implicit enumeration works by doing a partial optimization at each stage. It selects the best way to reach any point at that stage, and discards all the other possibilities. Thus for the second stage of our problem: using implicit enumeration we would look at each of the 10 ways to reach point *S*, and determine which is best—for instance that passing through point *E*. All the other possibilities for getting to *S* (nine in this case) would be dropped. Consequently—and this is where the implicit enumeration arises—so would all the possible routes to Washington whose first two days reach point *S* other than through *E*. We would not need to consider those explicitly since we would already know that they could not be optimal. It is important to realize that in this way we have, at point *S*, not merely discarded 9 possible solutions but $9(10^3)$ since there are 10^3 ways to proceed from *S* to Washington. Since this reasoning applies to each of the 10 points that we might reach by the end of the second stage, the partial optimization at this stage has eliminated $9(10^4)$ or nearly all the 10^5 possibilities. The effort required to do this has, however, been relatively small: the partial optimization at the second

Practicality of Implicit Enumeration

Consider the water distribution problem for the Salt Lake region, the one that had $[(5)^{30}]^3$ possible solutions.

Using implicit enumeration, we would have to consider approximately $(5)(30)(3) = 450$ different possibilities. This is quite feasible, even with a personal computer.

stage actually only considered 10^3 possible paths, 10^2 for each of the 10 points that could be reached at the end of day 2.

Implicit enumeration thus drastically reduces the number of possible solutions that have to be evaluated. Specifically, it transforms the number from the exponential that describes the total number of combinations

$$\text{Maximum Number of Solutions} = (\text{Levels})^{(\text{Stages})}$$

to a simple multiplicative relationship

$$\text{Solutions by Implicit Enumeration} \sim (\text{Levels})(\text{Stages})$$

This is the feature that makes dynamic programming practical (see box).

Dynamic programming can be applied to many different types of problems. Some can be organized into levels and stages in a fairly obvious way, since they describe a process that proceeds sequentially through time or space, as with the preceding example. The way other problems can be formulated in a form suitable for dynamic programming can be fairly subtle, either because they involve complex relationships between different points in time or space or because they do not represent any sequential process. Sections 7.4 and 7.5 present several examples of different kinds of problems on which dynamic programming can be applied.

Semantic caution: There is nothing inherently "dynamic" about dynamic programming. The term originated because this method was originally applied to problems in which time was an essential factor, specifically the launching of missiles or interceptor fighters, which had to reach a critical altitude and speed in a minimum time. As Sections 7.4 and 7.5 demonstrate, dynamic programming applies equally well to many problems that are quite static.

7.2 ASSUMPTIONS

The assumptions necessary for dynamic programming are those that enable us to perform valid partial optimizations along the way. These operations are the means by which dynamic programming cuts the number of evaluations to a feasible size, and they are absolutely crucial to the method.

Independence of Return Functions

Consider the transcontinental trip discussed as an example in Section 7.1. Independence of the return functions would mean that the value of anything we might do on the first day does not depend on what we might do the second or any other day. Concretely, independence implies for example that the cost of going from San Francisco to Elko does not depend on what we do after Salt Lake City, and vice-versa.

In this case, it would seem fairly obvious that the return functions are independent. But this is not necessarily so. For example, the cost of the trip beyond Salt Lake might well depend on how we got there: if we had stayed on the interstate highways and our car were still in good condition, these onward costs might be low; if, however, we had gone cross-country through the desert, we might have severely damaged our engine, making all further segments much more expensive. The return functions would then not be independent.

Consider another case, where the $g_i(X_i)$ represent the value of investments in projects X_i. These return functions might be totally independent, representing, for example, loans by the World Bank to different countries for quite different purposes. They might also be quite dependent, however. This would be the case if two or more facilities competed for the same market. The return from an investment in any one might be quite high, as a monopoly, but could be very much lower if investments are made in any of the competitive projects.

Formally, the assumptions permit us first of all to *decompose* the objective function $G(\mathbf{X})$, that is to break it up into a number of individual functions, $g_i(X_i)$, associated with each variable. Thus we want to be able to write:

$$G(\mathbf{X}) = [g_1(X_1), \ldots, g_n(X_n)]$$

Two conditions are sufficient to permit the decomposition of the objective function, separability and monotonicity. These are the assumptions we must make about a problem if we wish to use dynamic programming to optimize it.

The objective function is *separable* if each of the individual functions $g_i(X_i)$ are *independent* of the values of the other $g_j(X_j)$, $j \neq i$. This means the values of each $g_i(X_i)$ depend uniquely on X_i and are totally unaffected by what has or may happen elsewhere in the system. The separability condition is met quite commonly in practice, but there are many counterexamples and the analyst must be particularly careful about this assumption (see above box).

In the jargon associated with dynamic programming, the $g_i(X_i)$ are said to be *return functions*. They show the return, that is the contribution towards the objective function, associated with each X_i.

Monotonicity of Multiplicative Functions

Consider the multiplicative objective function

$$G(\mathbf{X}) = [g_1(X_1)][g_2(X_2)]$$

If both return functions are positive and real, then it is fairly obvious that increases in either will increase $G(\mathbf{X})$. Thus, if

$$g_1(X_1') = 5 \qquad g_1(X_1'') = 4$$
$$g_2(X_2) = 8$$

we have

$$g_1(X_1') > g_1(X_1'')$$

and

$$[g_1(X_1')][g'(\mathbf{X})] = 5g_2(X_2) = 40$$
$$> [g_1(X_1'')][g'(\mathbf{X})] = 4g_2(X_2) = 32$$

However, if we allow the return functions to be negative or imaginary, we can easily find situations for which the objective function is not monotonic. Thus, if

$$g_1(X_1') = +5 \qquad g_1(X_1'') = -1$$
$$g_2(X_2) = -3$$

we have

$$g_1(X_1') = 5 > g_1(X_1'') = -1$$

but

$$[g_1(X_1')][g'(\mathbf{X})] = (5)(-3) = -15$$
$$< [g_1(X_1'')][g'(\mathbf{X})] = (-1)(-3) = +3$$

The objective function is *monotonic* if improvements in each $g_i(X_i)$ lead to improvements in the objective function. To define this idea formally, consider the objective function separated into two parts, $g_i(X_i)$ and $G'(\mathbf{X})$:

$$G(\mathbf{X}) = [g_i(X_i), G'(\mathbf{X})]$$

The objective function is monotonic if, for all cases where $g_i(X_i') \geq g_i(X_i'')$ for the different levels X_i' and X_i'',

$$[g_i(X_i'), G'(\mathbf{X})] \geq [g_i(X_i''), G'(\mathbf{X})]$$

Additive objective functions:

$$G(\mathbf{X}) = \sum g_i(X_i)$$

are always monotonic. Increases in any $g_i(X_i)$ necessarily make $G(\mathbf{X})$ larger.

Multiplicative objective functions:

$$G(\mathbf{X}) = \prod g_i(X_i)$$

are monotonic only if the return functions are both nonnegative and real (see box on p. 145 for examples). In systems design this restriction never seems to present a practical difficulty.

7.3 SOLUTION STRATEGY

To optimize a system using dynamic programming, we need to deal with four elements:

- The *Organization* of the problem
- The *Formula* to be used in the partial optimizations
- The *Constraints*
- The *Solution* itself

This section discusses each in turn.

It is important to note that dynamic programming is not based on a standard formulation, as linear programming is. Dynamic programming is more an approach or a concept rather than a formula. The analyst wishing to use dynamic programming will often have to use imagination and skill to organize a problem so that it can be solved efficiently. To illustrate how this can be done, Section 7.5 gives examples of different kinds of formulations.

The fact that dynamic programming problems can differ from each other substantially leads to a great practical difficulty: good computer programs for dynamic programming are hard to find. Those that may be available to you may not suit your particular problem. Likewise, companies that develop software are naturally interested in large markets to pay their costs, and have thus not provided a range of dynamic programming programs to meet the various needs.

Organization. The first and essential part of the solution is to organize the problem into levels and stages. Then one must define the corresponding return functions. It is often not obvious how one can do this. Much of the interesting work in dynamic programming has in fact been in the development of new formulations that divide different systems problems into suitable levels and stages.

The stages are generally easier to imagine. The simplest case is when the program we have is *sequential*, that is, it represents a process that has a natural order through time or space. The example problem used in Section 7.1, concerning the transcontinental trip from San Francisco to Washington, was sequential: it embodied a natural progression from west to east. Possible stages in a sequential problem are fairly obvious; they represent some division of the

time or space. For example, each stage in a sequential problem might represent:

- *A division of time*—days, hours, seconds, and so on—as for the example problem in Section 7.1
- *A division of space*—such as blocks of 500 miles in the example problem—which would not correspond to the division of time since it might take more or less than a day to cover the distance, depending on the route chosen
- *A sequence of choices*—as in a game of chess or the sequence of midcourse corrections that can be given to a missile—which may have no intrinsic implication for the amount of space or time required.

The stages of a nonsequential problem are often simply the several parts of a system. For instance:

- The different possible investments in a portfolio of loans or shares in companies
- The various redundant components or machines in a system

For nonsequential problems the order we assign to the different stages will be absolutely arbitrary and, for all practical purposes, have no consequences for how fast or well the solution is obtained.

Defining the levels to be used in dynamic programming can be difficult. The simplest case is when the stage represents an investment in a particular project: the level is then the degree of commitment to that project. Thus, if Stage i represents the construction of an electric plant at a specific site, X_i will be the level of investment in that plant.

The "levels" may also be nominal: they may represent situations or names rather than degrees or levels of anything. In the example of the transcontinental trip of Section 7.1, the levels were cities along the way (Elko, Salt Lake City) that did not need to have a particular relation to each other latitudinally or otherwise.

In general, it is more accurate to refer to the *states* of a system at each stage rather than "levels." The state is the description of situation attained as the result of a transition from one stage to another. For instance, a state might be:

- The speed and altitude of a missile as a result of the number of stages of its guidance systems
- The position on the chess board
- The wealth of a gambler resulting from the probabilistic outcomes of a roulette wheel

The examples in the next two sections illustrate this concept.

In defining the levels or states of each stage, the analyst cannot use continuous functions. This is because the cost of a dynamic program is a direct proportion to the number of levels considered, and a continuous function implies an infinite number of levels—and infinite costs. The analyst must define the levels in steps, as a discontinuous function.

This necessity is generally an advantage. Dynamic programming deals easily with situations that are naturally discontinuous—as linear programming does not. For example, if we are determining the optimal number of aircraft in a fleet, we recognize that the number of aircraft must be an integer. We may have 52 or 53, for example, but not 52.4. In linear programming, which uses continuous functions, it is difficult or expensive to restrict the number of aircraft to integer values; we may have to solve a problem and then round out the result somehow. Dynamic programming, however, can define its feasible levels in any convenient way, specifically by using integer numbers. It can therefore determine an accurate answer where linear programming cannot.

The analyst must consequently be careful in defining the specific levels. The levels should preferably relate to discrete levels to the extent these are part of the actual problem. Their number should also represent a compromise between the desire for greater precision and the need for keeping computational costs down.

Finally, the definition of the return functions for each stage, $g_i(X_i)$, can be most complex. The simplest case again occurs when each stage is nonsequential and represents an investment of level X_i in a particular project, for example the construction of an electric plant. The return function $g_i(X_i)$ could then represent the profits or the amount of energy obtained from this investment. This kind of return function can typically be expressed as a simple formula or curve.

The return function for a sequential problem represents the change in state from one stage to the next. It can sometimes be expressed as a formula. Just as commonly it will be defined by a table providing each $g_i(X_i)$ in terms of states in the previous stage, $i - 1$, and the state X_i of stage i. Thus, in the transcontinental problem: the return function for stage 2 is the 10 by 10 matrix giving the cost (or mileage or travel time) from each city in stage 1 to each city in stage 2. The cost from Elko to Salt Lake City would be one entry in this matrix.

When return functions have to be described by tables rather than formulas, all subsequent calculations become more cumbersome. Additionally, it will take much more time to describe a problem. On the other hand, if certain transitions between stages are not feasible, they may simply be omitted from the $g_i(X_i)$ matrix. This possibility makes it easy to describe restrictions that would otherwise be very difficult to define with constraints, as required by linear programming.

Formula. The solution itself consists of two parts:

- Partial optimization at each stage.
- Repetition of this process through all stages.

The object of the partial optimization is to define the best result that can be obtained at any level or state at the end of a specific number of stages. This quantity is given by the *cumulative return function*, by analogy with the return function that relates to only one stage. The cumulative return function is denoted by $f_s(K)$, which designates the effect of being in state K, having passed through or examined the first s stages (see box).

Cumulative Return Function

Again consider the transcontinental automobile trip. Suppose that the traveler wishes to minimize cost.

The return functions for each stage would be the cost of getting to each city in that stage from the previous stage. Thus $g_1(X_1)$ would give the costs from the starting point, San Francisco, to each city in stage 1, such as Elko. Also, $g_2(X_2)$ would give the costs that could be incurred in the second day's trip, including that from Elko to Salt Lake City.

The cumulative return function, however, indicates the best result over previous stages. Thus:

$$f_2(\text{Salt Lake City})$$

is the least cost of getting to Salt Lake City in the first two stages. It is the cost of the San Francisco-Elko-Salt Lake City route, if that is best.

The cumulative return function is built up iteratively, from stage to stage. It is defined in terms of the return function of the current stage and the cumulative return function for previous stages:

$$f_i(K) = [g_i(X_i), f_{i-1}(K)]$$

Sometimes the cumulative return function can be defined by a closed-form equation. Its formula is then known as a *recurrence function*, so called because it repeats the definition of the function for each stage. Often, however, the cumulative return function will simply be defined by examination of the various possibilities as entered in a table (on paper or in computer memory). The examples in the next two sections illustrate how this can be done.

Special conditions exist for the first stage. The cumulative return function, $f_1(K)$, is then equal to the return function for that stage, $g_1(X_1)$. This follows from the fact that $f_0(K)$, the best that can be done over no stages, has no meaning.

Constraints. Constraints in dynamic programming are dealt with quite differently than they are in linear programming. They are almost never written as equations. Indeed, a dynamic programming problem may not be described in terms of equations.

Most constraints in dynamic programming are embedded directly in the organization of the problem. For example, if there is a restriction on the maximum number of levels or states for any stage, only that number is provided. For our transcontinental drive for instance, if a snowstorm blocked five cities in stage 2, these could simply be dropped from the description of the problem. Similarly, if certain transitions between stages are not feasible, they are simply not entered into the description of the problem. Thus, if the route from Elko to Salt Lake City is closed for repairs, that route is either dropped from the description of the $g_2(X_2)$—or given an infinite cost, which eliminates it from consideration.

Constraints that limit several X_i at once are difficult to enter into the program. Here again, what is easy for linear programming is difficult for dynamic programming. The general rule is that only one such constraint can in practice be incorporated with dynamic programming. This is done by means of the recurrence formula, as the example in Section 7.4 shows.

Solution. The optimal solution is provided by the cumulative return function over all the stages. This function is constructed so as to define the best solution over the preceding s stages and so must, when we have considered all stages, define the best value of the objective function.

In addition we need to know $\mathbf{X}^*$, the optimal value of the X_i variables. In the language associated with dynamic programming, the optimal set of X_i is known as the *optimal policy*. To obtain the optimal policy we must determine how we reached the optimal solution. This is in fact quite easy since the cumulative return function is always defined in terms of the return function for the last stage:

$$f_i(K) = [g_i(X_i), f_{i-1}(K)]$$

Knowing the optimal solution, we then know the transition from the previous stage that brought us to that point; from the cumulative return function for the previous stage we can trace back one more stage, and so on. In practice, since we do not know which path eventually leads to the optimum, the dynamic program must keep track of the optimal path to each level of each stage.

Dynamic programming provides a limited amount of sensitivity information as part of the solution. We are pretty much limited to asking how the optimal solution would change if we wished to end up at a different state at the end of all stages. This is most useful when we have a budget on investments and wish to define the shadow price on that constraint. The example in Section 7.4 illustrates this point.

7.4 EXAMPLE

Suppose we wish to maximize the hydroelectric power produced by building dams on three different river basins, subject to a budget of \$3 million.

The problem is suitable for dynamic programming provided the return on each investment, that is, the hydroelectric power from each dam, is independent of the others. As a practical matter these projects will be independent so long as the river basins do not interconnect. If they did, a dam on one river might change the water available to another dam and thus change its return function.

The problem is nonsequential since there is no necessary order in which investments have to be made. We may eventually decide to build the facilities in some order, but we are free to do so as we wish.

Organization. The stages are the individual projects. Since there is no necessary order, we can label any project we want as stage 1, another as stage 2, and the third as stage 3.

The levels are the amounts we can invest in each project. For simplicity, suppose these are 0, 1, 2, or 3 million dollars. We could have more levels if we wish, to obtain greater accuracy. The method would not change, however. Note that the increments between each level are the same. Organizing the problem this way makes it much easier to write the recurrence formula and to execute the program. It is not absolutely necessary, but strongly recommended if at all possible.

The return functions, $g_i(X_i)$, are the amounts of power produced by each dam at stage i (that is, site i) for an investment of level X_i at that site. Table 7.1 defines the return functions for this problem. As can be seen from their plots in Figure 7.2, the feasible region is both nonconvex, due to $g_2(X_2)$, and nonlinear. The objective function is additive since it represents the sum of the power produced:

$$\text{Power} = \sum g_i(X_i)$$

Constraints. The budget constraint implies that

$$\sum X_i \leq 3$$

This restriction will be embedded into the organization of the problem by limiting the number of levels at each stage. These investments can in no case be greater than the budget. (They can, however, be less.) Additionally, the investments cannot be negative, $X_i \geq 0$. This constraint will be embedded in the detailed description of the problem, as shown by the subsection on network representation that follows.

Recurrence formula. The recurrence formula for additive objective functions such as this one is

$$f_i(K) = \text{Max}[g_i(X_i) + f_{i-1}(K - X_i)]$$

This equation is to be understood as the maximum return that can be obtained by

TABLE 7.1
Return functions for the investment problem

Level of investment,	**Return function, $g_i(X_i)$**		
X_i	**$i = 1$**	**$i = 2$**	**$i = 3$**
0	0	0	0
1	2	1	3
2	4	5	5
3	6	6	6

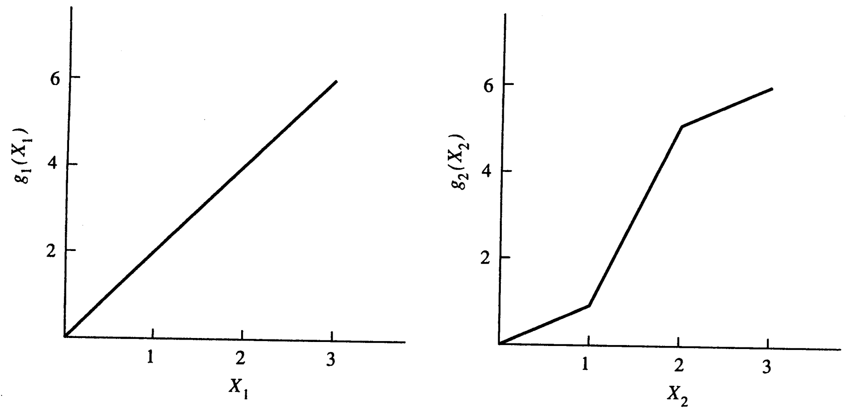


FIGURE 7.2
Return functions $g_i(X_i)$, showing the value of one, two, or three units of investment in a dam for each river basin site of the example problem.

investing some level of money K over s stages is equal to the maximum of all the combinations of investing X_i in stage i and the balance, $(K - X_i)$ in the previous stages. [Note: Max[x, y] is to be read as the maximum of all the combinations of x and y, related by addition or multiplication as defined within the brackets.]

Network representation. The problem can be represented by a network similar to the one for the illustrative situation in Section 7.1. Figure 7.3 does this.

By analogy to the transcontinental drive, we can express the investment problem as starting from point A (no investments in any project) to point M (a budget of 3 invested over 3 projects). We want to find the optimal path to do this, that is, the best transition from stage to stage. These transitions represent the investments in the next stage, $g_i(X_i)$, as labeled in Figure 7.3.

This network differs from the map of the transcontinental drive in two respects. First of all, the paths from left to right are all level or downward sloping: this is because all the investments in any project are greater or equal to zero, $X_i \geq 0$. This means that the number of combinations in a complete enumeration in this case is less than the maximum given in Section 7.1. The network also illustrates how infeasible transitions are dropped from the network of logical possibilities.

The second difference is that we have included several endpoints for this network, points J, K, and L in addition to M. These points define the maximum return we could obtain for lesser amounts than the budget. They are thus useful in sensitivity analysis.

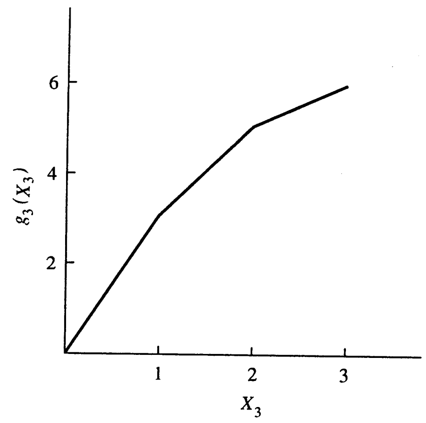


FIGURE 7.2 *(continued)*
Return functions $g_i(X_i)$, showing the value of one, two, or three units of investment in a dam for each river basin site of the example problem.

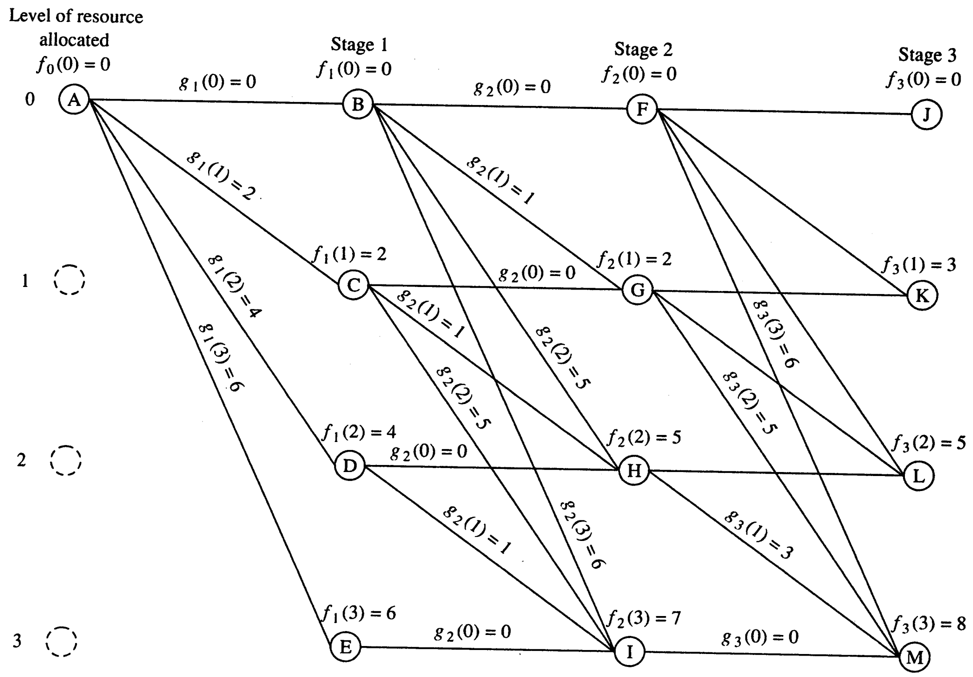


FIGURE 7.3
Nodal representation of example problem illustrating the many ways in which the four levels of investment (for zero, one, two, or three units) can be spent on three projects or stages.

Solution. We want to find the best way to spend the budget ($3 million) over all 3 projects: $f_3(\$3\text{M})$. We build this cumulative return function from $f_1(K)$, using the recurrence formula.

The cumulative return function for the first stage is equal to the return function of that stage, as indicated in Section 7.3. Thus:

$$f_1(K) = g_1(K)$$

$$f_1(0) = 0 \qquad f_1(1) = 2 \qquad f_1(2) = 4 \qquad f_1(3) = 6$$

To determine the cumulative return function for the second stage we must consider all the combinations for investing 0, 1, 2, and 3 million dollars over two stages, as defined by

$$f_2(K) = \text{Max}[g_2(X_2) + f_1(K - X2)]$$

To find $f_2(0)$ is easy. There is only one way to do this, by spending nothing on both projects 1 and 2:

$$f_2(0) = \text{Max}[g_2(0) + f_1(0)] = 0$$

To find $f_2(1)$ becomes more complicated. There are now two ways to invest; either $X_1 = 1$ or $X_2 = 1$. Thus:

$$\begin{aligned} f_2(1) &= \text{Max}\{[g_2(1) + f_1(0)] \text{ or } [g_2(0) + f_1(1)]\} \\ &= \text{Max}[(1 + 0) \text{ or } (0 + 2)] = 2 \end{aligned}$$

Similarly, $f_2(2)$ requires three combinations, and $f_2(3)$, four. We find for each:

$$f_2(2) = 5 \qquad f_2(3) = 7$$

as Figure 7.3 indicates.

Finally, for the third stage we proceed just as previously. If we are only interested in the optimum, we need only calculate:

$$\begin{aligned} f_3(3) &= \text{Max}[g_3(X_3) + f_2(3 - X_3)] \\ &= \text{Max}[(0+7) \text{ or } (3+5) \text{ or } (5+2) \text{ or } (6+0)] = 8 \end{aligned}$$

The optimum policy is determined by looking at how we obtained the optimum. In this case we can see that

$$f_3(3) = 8 = g_3(1) + f_2(2)$$

So to obtain this optimum we must set $X_3 = 1$ and invest $2 million on the previous two projects. Looking further we find that

$$f_2(2) = 5 = g_2(2) + f_1(0)$$

which implies $X_2 = 2$ and $X_1 = 0$. The optimal policy is thus

$$\mathbf{X}^* = (0, 2, 1)$$

This is path *ABHM* in the network in Figure 7.3.

Sensitivity analysis. The sensitivity to changes in the budget constraint can be obtained simply by calculating $f_3(K)$ where K is less than the budget. We can thus find $f_3(2) = 5$ as shown by point L in Figure 7.3. The shadow price on the budget for the first $1 million decrease (from 3 to 2 million dollars) is thus 3.

Note that, because the feasible region is not convex in this case, the shadow price does not change monotonically as the constraint tightens, as we would expect for linear programming. Thus, for the second $1 million decrease in budget (from 2 to 1 million dollars) the shadow price is

$$f_3(2) - f_3(1) = 5 - 3 = 2$$

Yet the shadow prices for the third $1 million decrease in budget (from 1 million dollars to zero) is

$$f_3(1) - f_3(0) = 3 - 0 = 3$$

In this case the shadow price both increases and decreases as we tighten the constraint from a budget of $3 million.

Comparison with marginal analysis. If one attempted to apply marginal analysis to this problem, one would not get to the right answer. This is because, by marginal analysis, we would invest each increment of money where it immediately seemed most productive, and we would miss opportunities for increasing returns to scale.

Referring to either Table 7.1 or Figure 7.2, we can see that, by marginal analysis, the best way to invest our first million dollars is in project 3, to obtain $g_3(1) = 3$. The best way to invest our second and third million is project 1 or 3. Thus by marginal analysis our apparent—but erroneous—best policy is (2, 0, 1) or (1, 0, 2) for a total of 7. By looking myopically at the marginal returns, project 2 consistently looks unattractive although it eventually has the highest returns and should be part of the optimal policy. The example demonstrates the usefulness of dynamic programming for situations involving nonconvex feasible regions.

Tree representation. Another way to look at the proliferation of possible designs when we do complete enumeration is by means of a "tree." This shows the possible choices at any stage and, by the time we reach the last stage, all the possible combinations. The "tree" for our problem appears in Figure 7.4. It is an extensive form of the network of Figure 7.3.

The advantage of the tree representation is that it provides a convenient way of showing how the partial optimization of each stage eliminates classes of possible designs. In the operations research jargon associated with these trees, we say that we "prune" the tree at each stage. (Keeping with the botanical allusion, we also say that, as the number of possible combinations proliferates, the tree becomes a "messy bush.")

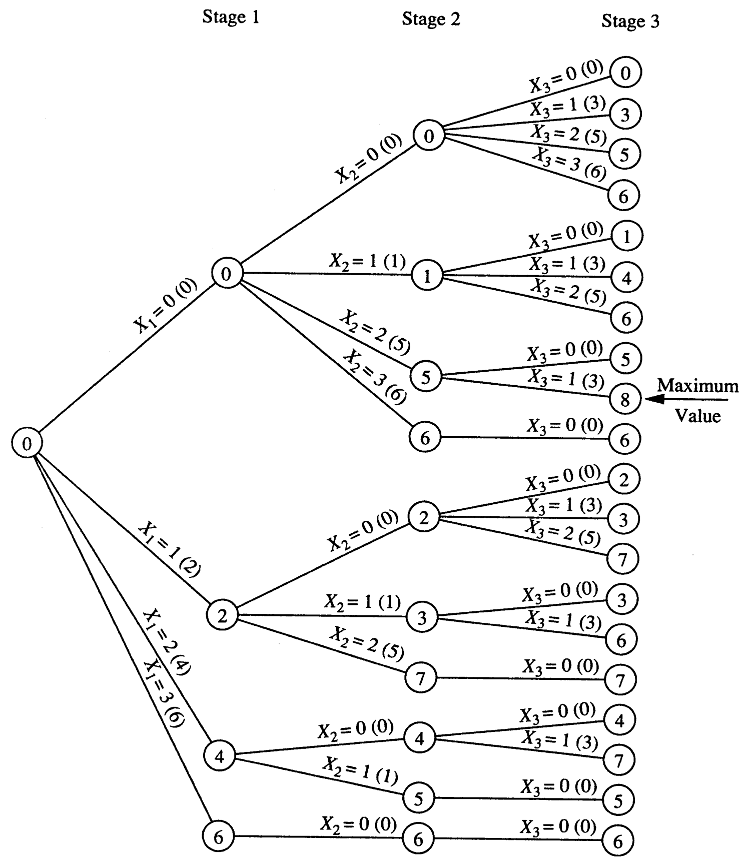


FIGURE 7.4
The "tree" of all combinations of three units or less over three projects in the example problem, showing the value of each combination of allocations. Note: Read $X_1 = 1(2)$ as the return associated with setting the first decision variable X_1 at 1 is 2. The number in the circle gives the total value or return of all preceding allocations.

Consider the number of ways of allocating $1 million over the first two projects. The two possibilities appear in Figure 7.5*a*. One of them is worse and is pruned—along with its three possibilities for stage 3 shown in Figure 7.4. Similarly, when we consider how to allocate $2 million over two projects, we drop some more possibilities, together with their sequels (Figure 7.5*b*). Altogether, when we have finished with the partial optimization at the second stage, we are left with only four alternatives—one for each level (Figure 7.6).

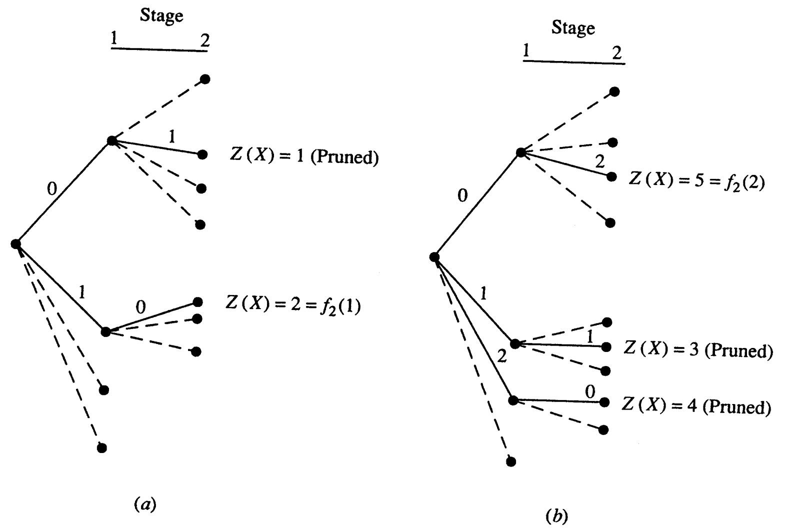


FIGURE 7.5
Pruning of suboptimal combinations at the second stage of the example problem, shown as a schematic from Fig. 5.5. (*a*) One unit over two stages; (*b*) two units over two stages.

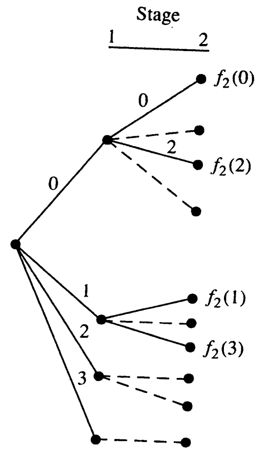


FIGURE 7.6
Optimal combinations at each level for the example problem, shown as a schematic of Fig. 7.4. All but four combinations, one for each level, are pruned away at each stage.

7.5 FURTHER FORMULATIONS

This section shows how dynamic programming can be applied to three important classes of problems: inventory planning, replacement scheduling, and reliability analysis. The optimization of inventories over time is a sequential problem, similar to the introductory example in Section 7.3. The difference is that the development is through time rather than over space. The question of scheduling the replacement or renovation of equipment is another kind of sequential problem. The reliability analysis is a nonsequential problem like the example in Section 7.4. The reliability analysis requires a multiplicative recurrence function, moreover. These three examples thus cover the range of situations for which dynamic programming is most useful.

Inventory analysis. The purpose of an inventory is to keep a supply on hand to meet demands when needed. If one runs out of inventory, and cannot meet a requirement, one generally incurs a loss: this might be the profit foregone on a sale that was not made, or the cost of halting production until the item needed is available.

Maintaining an inventory is expensive. One must both buy the supply in advance (perhaps borrowing money to do so), and pay for its storage, maintenance, and security. This encourages one to keep inventory as low as possible.

There are two reasons to keep an inventory fairly large, however. The bigger it is, the less likely one is to run out due to sudden surges in demand. Further, by ordering in large lots, one may be able to benefit from volume discounts and economies of scale. Optimizing an inventory requires a careful balance between the advantages and disadvantages of size. The objective is to determine the least expensive size of inventory in any period, that is, in any stage.

The typical inventory problem involves the following elements:

- The *beginning inventory*, I_0, the amount on hand for the beginning or first period of the analysis
- The *ending inventory*, I_N, the amount desired for the last period of the analysis
- The *requirement* or use, R_i, for each period
- The *production*, Y_i, or amount of resupply for each period
- The *costs* of both holding inventory in any period, C, and of obtaining the resupply, $C(Y)$

The optimal policy will specify the amount to be produced in each period. The Y_i are the decision variables; all the other parameters are considered set or beyond the control of the analyst. The ultimate objective is to find the least cost, over all N periods, of achieving the desired final inventory: $f_N(I_N)$. The return function $g_i(X_i)$ for each period represents the financial implications of producing Y_i. This involves both the cost of production itself and of holding the inventory this production leads to. This is the inventory of the previous period, plus production, minus use:

$$I_i = I_{i-1} + Y_i - R_i$$

Store Inventory

Consider a store with a starting inventory of 2 in October, monthly demands of 3 through December, and a desire to end up with no inventory in January:

$$I_0 = 2 \qquad R_i = 3 \qquad I_4 = 0$$

Assume that costs of holding inventory are 10 per unit per period and that the costs of production of supply has economies of scale up to 5 units/month:

Y_i:	1	2	3	4	5
$C(Y_i)$:	80	140	180	200	210

Observe that, since demand is 3, the most we can have on hand in December is 3 if we are to eliminate stock by January. Note also that the largest increase in inventory from one period to the next is 2, since this is the maximum supply less demand.

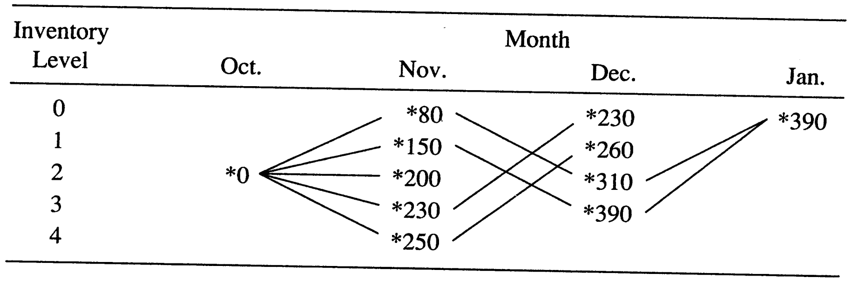
Table 7.2 presents the results. The optimal costs for November, stage 1, are simply the $g_i(Y_i)$. Thus the cost of having an inventory of 2 is the cost of producing 3 units, to meet the demand, plus the cost of holding the inventory:

$$f_1(2) = C(Y_i = 3) + C(2 + 3 - 3) = 180 + 20 = 200$$

The optimal values for December, stage 2, derive from the recursion formula. Thus to have zero inventory, one can start with zero in November and produce 3, start with 1 and produce 2, et cetera. So:

$$f_2(0) = \text{Min}[(80 + 180) \text{ or } (150 + 140) \text{ or } (200 + 80) \text{ or } (230 + 0)]$$
$$= 230$$

TABLE 7.2
Analysis of inventory problem with optimal cost, $f_i(X_i)$, and optimal path for each point



Inventory Level	Month: Oct.	Nov.	Dec.	Jan.
0		*80	*230	*390
1		*150	*260	
2	*0	*200	*310	
3		*230	*390	
4		*250		

Therefore:

$$g_i(Y_i) = C(Y_i) + C(I_{i-1} + Y_i - R_i)$$

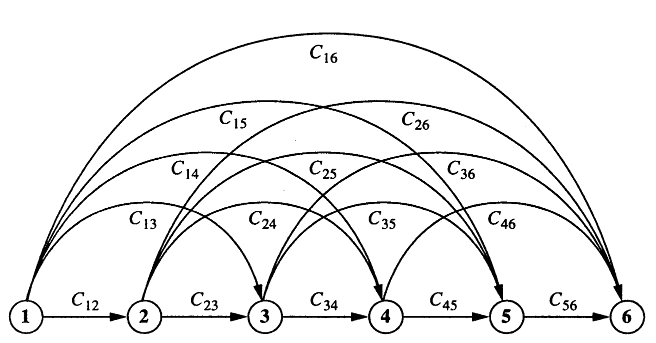
The recursion formula for the cumulative return function seeks the minimum of the additional costs and the previous inventory:

$$f_i(I_i) = \text{Min}[g_i(Y_i) + f_{i-1}(I_i - Y_i + R_i)]$$

Note that, to be in state I_i in period i, we had to be in state $I_{i-1} = (I_i - Y_i + R_i)$ in period $i - 1$. See box (page 159) for an example of this analysis.

Replacement scheduling. The components of a system normally degrade over time and have to be replaced. When this should happen is rarely obvious. Any machine can usually be maintained, and it is generally cheaper to pay the maintenance bill in any period than to buy a new machine. Yet the replacement should cut maintenance costs for many years. When should one take advantage of this possibility? For example, when should you trade in a car for a newer one?

Dynamic programming is an effective way to determine an optimal replacement policy. To do so, one needs to have the costs or benefits of buying the possible range of newer components, ones that will last for one or more periods. In this kind of problem, the stages are conveniently the beginning of each period, and the states describe the useful life of the equipment. The analysis seeks to find the least cost policy of buying and maintaining equipment over a given total number of periods. See box for example.



C_{12}	100	C_{23}	105	C_{34}	95	C_{45}	90	C_{56}	140
C_{13}	190	C_{24}	210	C_{35}	200	C_{46}	195		
C_{14}	310	C_{25}	290	C_{36}	295				
C_{15}	395	C_{26}	410						
C_{16}	495								

FIGURE 7.7
Replacement schedule data.

Replacement Schedule

A company has a contract to produce insulated pipe for the construction of an Arctic pipeline over five years . It must thus acquire and maintain a machine to wrap tubes over this total period.

The company could equip itself in many ways. For example, it could buy a new machine every year, selling the old one for scrap or reuse elsewhere. It could also buy a single machine for the whole five years. Figure 7.7 illustrates the various transitions that could occur from stage 1, the start of year 1, to stage 6, the end of the fifth year, that is the start of year 6.

The table in Figure 7.7 indicates the costs C_{ij} of acquiring in stage i a machine that will be replaced in stage j. Thus, $C_{12} = 100$ means that buying a machine at the beginning of the first year that will only serve until the beginning of year 2 costs 100. These costs are the $g_i(X_i)$.

Table 7.3 presents the results. The cumulative return function for the beginning of the first year, $f_i(X_i)$, is just the $g_i(X_i)$. The optimal cost for having a machine that will be replaced at the beginning of year 3, that is, one that will be replaced one year from stage 2, is

$$\begin{aligned} f_2(1) &= \text{Min}[C_{13} \text{ or } (C_{12} + C_{23})] \\ &= \text{Min}[190 \text{ or } (100 + 105)] = 190 \end{aligned}$$

Ultimately we want, at the end of the fifth year which is the beginning of year 6, to have zero life in our equipment. For this problem, the optimal result is: $f_6 = 480$. The optimal policy is first to buy a machine for two years, then for one year, and finally for 2 years again.

TABLE 7.3
Analysis of replacement schedule with optimal cost and optimal path for each point

Years of Machine-Life	Beginning of Year 1	2	3	4	5	6
0						
1	100	190	285	375	480	
2	190	310	390	480		
3	310	390	485			
4	395	495				
5	495					

Reliability analysis. A system can be made more reliable in two ways: by increasing the reliability of any component, and by providing additional, redundant components. These components, placed in parallel with others, increase reliability because they can function when the others fail.

The object of a reliability analysis is to determine how to maximize the probability that a system will function. The idea is to allocate resources optimally to the possible components. In this regard the analysis is identical to the investment example in Section 7.4.

Reliability analysis can only be done by dynamic programming if the probability of failure of each component is independent of that of the others. This is a basic requirement for dynamic programming (see Section 7.2). The method cannot be used when the failure of the parts is all due to a common cause, such as a fire that affects all parts concurrently.

The dynamic programming version of reliability analysis differs from the investment problem in that its cumulative return function is multiplicative. This is because the probability that several independent components function simultaneously is a factor of the probabilities of individual components. The point of this example is to show how to deal with such multiplicative functions.

In practice the analysis focuses on minimizing the probability of failure of a system. Of course, this achieves the same result as maximizing success since the two are complementary:

$$P\text{ (Success)} + P\text{ (Failure)} = 1.0$$

The reason for focusing on the failure of the system is that its probability is much easier to calculate. Given the assumption of independence, it is the multiplication of the probability that all parallel components fail simultaneously:

$$P\text{ (Failure of System)} = \prod{}_i P\text{ (Failure of Each Component)}$$

The probability of success is much more complicated to calculate: it is the sum of all the combinations of ways in which the system could operate.

The first step of the analysis is thus to convert all the data on the probability of success of components into their complementary probability of failure. The step is necessary because, for understandable psychological reasons, manufacturers like to accentuate the positive high performance of their parts, rather than draw attention to the fact that these may also fail.

As with the investment problem, the return functions $g_i(X_i)$ for greater investment X_i in any part is simply the capability achieved, that is, the probability of failure of a part. Note that this is a decreasing function: more expense should reduce the probability of failure. The recursion function is similar to that for the investment, except that it is a minimization and a multiplication:

$$f_i(X_i) = \text{Min}\,\{[g_i(X_i)][f_{i-1}(K - X_i)]\}$$

See box for an example of reliability analysis.

Reliability Analysis

Consider a system that can have up to three parallel components to provide redundancy. The probability of failure or nonperformance of each depends in the investment made in the part, as indicated in Table 7.4. The question is: how should the system be designed to maximize the reliability, that is, to minimize the probability of failure?

The results of the analysis appear in Table 7.5, organized as Table 7.2. As usual, the optimal performance for the first stage is simply $g_1(X_1)$. The optimal values for the next two stages derive from the recursion formula. The best way to spend 2 units on the first two components, for example, is the minimum of the probability of failure achieved by no investment in the first part and 2 units in the second, $\{f_1(0)\}\{g_2(2)\}$, 1 unit of investment in each, and so on:

$$f_2(2) = \text{Min }[(1.0)(0.4) \text{ or } (0.4)(0.5) \text{ or } (0.3)(1.0)] = 0.20$$

The optimal design in this case is triple redundancy, achieved by placing one unit of investment in each part:

$$f_3(3) = (0.4)(0.5)(0.6) = 0.12$$

This solution is not intuitively obvious. Indeed, part 1 is far superior to the other two for any level of investment; as Table 7.4 shows. Yet this kind of diversified solution is characteristic of reliability problems: because probabilities for parallel components multiply, better performance is often achieved by redundancy than single superior components.

TABLE 7.4
Return functions for reliability analysis

Investment level, X_i	Return function $g_i(X_i)$ $L = 1$	$L = 2$	$L = 3$
0	1.0	1.0	1.0
1	0.4	0.5	0.6
2	0.3	0.4	0.5
3	0.2	0.3	0.4

TABLE 7.5
Analysis of reliability problem with optimal cost, $f_i(X_i)$, and optimal path to each point

Investment Level	Component 1	Component 2	Component 3
0	*1.0	*1.0	
1	*0.4	*0.4	
2	*0.3	*0.2	
3	*0.2	*0.15	*0.12

REFERENCES

Wagner, H. M., (1975). "Introduction to Dynamic Optimization Models," Chapter 8 *Principles of Operations Research*, 2nd ed., Prentice-Hall, Englewood Cliffs, NJ.
Winston, W. L., (1987). "Deterministic Dynamic Programming," Chapter 18, *Operations Research: Applications and Algorithms*, PWS Publishers, New York.

PROBLEMS

7.1. *Engulf and Devour*
The safety engineer for Engulf and Devour's Lakeside factory has estimated the number of employee disabilities (measured in sick days/year) that can be avoided by various measures: covering dangerous machinery (CDM), providing protective clothing (PPC), improving ventilation (IV), and/or lowering noise levels (LNL). The information is summarized in the following table.

	Measures			
Investment (10^3\$)	**CDM**	**PPC**	**IV**	**LNL**
0	0	0	0	0
10	5	15	2	5
20	10	22	12	7
30	15	25	15	15
40	20	25	17	25
50	22	30	25	26

(*a*) Use dynamic programming to determine how a \$70,000 budget could be most effectively employed to reduce disabilities.
(*b*) How much of a difference would it make if management cut back the safety budget to \$60,000? How much money would be saved for each additional sick day the budget cut would cause?
(*c*) Could this problem be solved by linear programming? Why or why not?

7.2. *Stereo*
Felicia has decided to subscribe to a service, offered by Bob's Stereo Outlet, that will optimize her stereo system purchase. Having decided to buy a tape deck, a turntable, a pair of speakers, and a receiver, Felicia fills out the following form, indicating the value to her of each component:

Model	Price(\$)	Tape	Turntable	Speakers	Receiver
A	200	0.1	0.2	0.1	0.3
B	400	0.3	0.3	0.2	0.4
C	600	0.5	0.5	0.5	0.5
D	800	0.9	0.8	0.9	0.8
E	1000	0.9	0.9	1.0	0.9

What would the BSO recommend Felicia buy for \$1000? For \$2000?

7.3. *Exam Cram*
Howie McKim has a maximum of six hours of study time before the Systems Analysis midterm. He knows what material will be covered on the test, how well prepared he is in each area, and how much an additional hour of study will help him in each area:

	Areas			
Hours of study	**MA**	**LP**	**SA**	**DP**
0	5	15	2	5
1	10	22	12	7
2	15	25	15	15
3	20	25	17	25
4	22	25	25	25

MA = Marginal Analysis
LP = Linear Programming
SA = Sensitivity Analysis
DP = Dynamic Programming

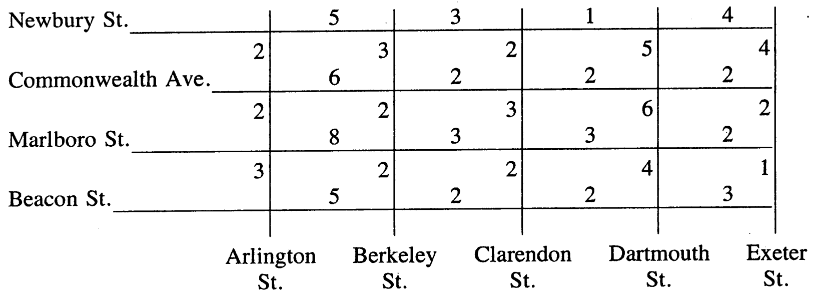
(*a*) Use dynamic programming to allocate Howie's study time to maximize his point grade on the midterm.
(*b*) How many fewer points will he get if he takes an hour off to work for the presidential candidate of his preference?
(*c*) Could this problem be solved by linear programming? Why or why not?

7.4. *Fire House*
City planners must determine the "optimal" allocation of fire stations to three districts. Zero to three stations may be located in a district. The table below shows the relationship between the number of stations and the annual expected property damage due to fires, based on statistical data. Differences among districts are due to population, construction materials, and so on. A budget constraint restricts the total number of stations to five.

	Number of stations per district			
District	**0**	**1**	**2**	**3**
1	2.0	0.9	0.3	0.2
2	0.5	0.3	0.2	0.1
3	1.5	1.0	0.7	0.3

(*a*) Determine the optimal allocation by dynamic programming.
(*b*) Could the problem be solved by linear programming? Why or why not?
(*c*) Write an equation for the recurrence formula.

7.5. *Rat Patrol*
A Boston rat crawls out of its hole at Beacon and Arlington Streets and decides to raid its favorite garbage can behind a store at Newbury and Exeter Streets. Based on previous experience with cats, traffic, and lighting conditions, the rat estimates travel time (in minutes) for each block as shown:



(*a*) Find the rat's minimum route to the garbage can.
(*b*) How long will the trip take by the shortest path?
(*c*) If, instead, the rat decided to chew on a telephone pole at Marlboro & Dartmouth, how long would that trip take from its hole?

7.6. *Winter Inventory*
A company manufacturing large industrial equipment projects sales of 3 units/month in January, February, and March. Beginning inventory in January (ending inventory in December) is 2 units. The company desires that ending inventory in March also be 2 units. Demand for any month may be satisfied from beginning inventory or that month's production. Ending inventory in any month is limited to 5 units, and production during any month is limited to 5 units. There is a $10,000 holding cost for any unit left in inventory at the end of a month. Production cost depends on the number of units manufactured in a month as follows:

Units/Month	0	1	2	3	4	5
Cost ($000)	0	80	100	120	130	140

(*a*) Define the objective, decision variables, objective function, constraints, and the return functions for each stage.
(*b*) Determine the optimal production schedule for January, February, and March.

7.7. *Ad Campaign*
Muncho wants to start a TV ad campaign for its new line of desserts. Market studies tell it how many potential customers may be attracted by ads on the three major TV networks (see following table). Suppose advertising time costs the same on all networks and that Muncho's budget limits the ad campaign to five hours.
Millions of customers who will see the ad on different networks:

	Network		
Number of hours	**ABC**	**CBS**	**NBC**
1	3	1	2
2	5	2	4
3	6	5	6
4	6	6	6

(*a*) How should Muncho buy ad time to attract the most customers?
(*b*) Write the recursion formula for the problem.
(*c*) Could Muncho use linear programming to solve this problem? Why or why not?

7.8. *Cheryl Consultant*
Cheryl Consultant has an opportunity to invest in one or more of four proposal writing projects A, B, C, or D. Any investment in a project must be made in $1000 increments. Moreover, there are limits to the amounts Cheryl might invest effectively in any one project: $7000 for A; $5000 for B and D; and $6000 for C. Cheryl is also willing not to invest at all in a project if her money might be invested more profitably in the other projects. Cheryl estimates her returns for investments as follows:

	Level of investment (in $1000)						
Project	**1**	**2**	**3**	**4**	**5**	**6**	**7**
A	5	10	15	25	35	50	55
B	3	6	12	18	30	30	30
C	20	35	45	55	60	65	65
D	9	16	29	37	45	45	45

(*a*) If Cheryl has $8000, what is her optimal strategy?
(*b*) What will her returns from this strategy be?
(*c*) Suppose Cheryl's friend Jill offers her an extra $1000 if Cheryl agrees to repay Jill $3000. Should Cheryl accept the offer? Explain your answer.
(*d*) If Cheryl only has $7000, how should her investment plan change?

7.9. *Unit Fund*
The Unit Fund wants to assign 10 volunteers to solicit contributions from the companies in four office buildings. The director estimates the contributions (in kilo bucks) as follows:

	Number of volunteers							
Building	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
1	0	5	10	15	25	35	50	55
2	0	3	6	12	18	30		
3	0	20	35	45	55	60	65	
4	0	9	16	29	37	45		

No additional pledges would be received by sending more than seven volunteers to building 1, more than five to building 2, more than six to building 3, and five to building 4.
(*a*) How should volunteers be assigned, and how much will they collect?
(*b*) How does the solution change if there are nine volunteers?

7.10. *V. Erner Brawn*
Colonel V. Erner Brawn, coordinator for the Advanced Space Shuttle, has to determine the allocation of NASA R&D funds that will maximize the probability that three technological breakthroughs (A, B, & C) will take place in time to incorporate them into the advanced shuttle. Specialists estimate that the probability that any breakthrough will occur in time is a function of project funding as follows:

	Funding (in 10^7 dollars)			
Project	**0**	**2**	**4**	**6**
A	0.1	0.2	0.3	0.4
B	0.1	0.3	0.4	0.6
C	0.1	0.1	0.2	0.5

Assuming that the above probabilities are independent, the likelihood that all three occur in time is the product of the individual probabilities.
(*a*) If NASA funding is \$100,000,000 [10 times 10^7], what probability of success should Colonel Brawn report to her superiors?
(*b*) How should she allocate her funds?
(*c*) How much money is needed to triple the probability of success?
(*d*) How should funds then be allocated?

7.11. *Stagecoach*
When Mark Off decides to seek his fortune in San Francisco, the stagecoach is the only real means of transportation from the East, where he lives, to the West.

The entire trip requires four stops, regardless of routing. Worried about the dangers involved in such a trip, Mark reasons that the insurance premiums between stops measures the inherent danger of each link.

	Insurance premiums (\$ × 10^3)								
From stop / **To stop:**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**SF**
1	3	3	6						
2				7	4	8			
3				4	4	5			
4				6	3	3			
5							6	9	
6							7	7	
7							8	5	
8									5
9									4

(*a*) State Mark's objective function for going by the safest route.
(*b*) How is his problem characterized and analyzed in terms of stages?
(*c*) What route should he take to go to San Francisco?
(*d*) Mark's father, Pop Off, lives in stop 8. Find an optimal routing from the East to the West that goes through stop 8.

7.12. *Mark Off Again*
Mark Off receives an urgent request from the Tsar to find a route to Rostov from Vladivostok. The expenses associated with each leg of the journey are for protection against attacks by Cossacks and local tribesmen. Find an optimal routing.

	Expenses (rubles × 10^3)										
From stop / **To stop:**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**R**
V	5	6										
2			4	7	11							
3			8	6	9							
4						9	12	13	14			
5						8	15	12	8			
6						10	9	8	7			
7										12	11	
8										9	8	
9										7	5	
10										9	9	
11												2
12												3

7.13. *Truck Loading*
Consider a truck whose maximum loading capacity is 20 tons. Suppose that there are four different items, various quantities of which are to make up a load to be carried to a remote geographical area. Suppose we wish to maximize the value of what the truck carries to the inhabitants, given the following weights and values of the four items.

Item	Weight (tons)	Value
1	3	5
2	4	7
3	5	8
4	6	10

(*a*) Find the optimal solution by dynamic programming.
(*b*) Write the recurrence formula for this problem.
(*c*) Define the assumptions of dynamic programming.

7.14. *Amplifier*
Consider a three-stage amplifier, where the gain in each stage is a function of the expenditure on that stage, and the total gain equals the product of the gains of the three stages.

	Stage		
Cost ($)	1	2	3
0	1	1	1
1	2	1.5	2.5
2	3	2	3
3	4	5	6

For $0, you can put in wire. Spending more than $3 per stage produces no additional gain. Determine the maximum gain attainable for $0, $1, $3, $4, $5. How would each maximum be achieved?

7.15. *Burn Out*
Three independent systems can be used to detect overheating of a large and expensive machine. The probability that any one of these systems tends to operate is inversely related to its quality and cost.

Cost	Probability of failure of system		
($, 10^3)	A	B	C
0	1	1	1
1	0.9	0.8	0.9
2	0.8	0.7	0.7
3	0.6	0.6	0.6
4	0.5	0.5	0.5

(*a*) Use DP to select the best design(s), given a $4,000 budget.
(*b*) Write a recursion formula for this problem.

7.16. *B, D, and M*
Two companies, B and D, and university M are independently trying to solve a robotics problem by different methods. Their probability of success is 0.6, 0.4, and 0.2, respectively. A consultant has determined that these probabilities would improve if more money were invested in the projects according to the following schedule. The government, wishing to maximize the probability that the problem will be solved, is prepared to support the research with $6 million.

Support (10^6)	Company B	Company D	Univ. M
None	0.60	0.40	0.20
1	0.75	0.55	0.40
2	0.85	0.65	0.55
3	0.90	0.75	0.65
4	0.93	0.82	0.73
5	0.95	0.87	0.80
6	0.96	0.91	0.86

(*a*) How should the government allocate its money? Assume that, with respect to this research, the government does not think in units of less than $1 million.
(*b*) How does the solution obtained by dynamic programming compare with the strategy of backing the most advanced company completely? Are you prepared to justify backing anyone else?

7.17. *Circuit Reliability*
An electronic circuit consists of four parts connected in series as shown:

Input — Part 1 — Part 2 — Part 3 — Part 4 — Output

The circuit functions properly only if *all four* parts function properly. Each part of the circuit is built by connecting a number of identical components in parallel. Each component in each part functions or fails independently of other components in the circuit. A given circuit part functions properly if *at least one* of the components in this part functions properly.
For convenience, we define

P_i = probability that a particular component in part i will function
C_i = unit cost of each component in part i

Suppose that we are given:

Part	P_i	$C_i(\$)$
1	0.9	2
2	0.8	4
3	0.7	3
4	0.6	1

To calculate the reliability of the circuit, we note:

(*a*) P(Circuit functions) $= \prod_i$ P (part i functions)

(*b*) P(Part i functions) $= 1 -$ P (part i fails)

$= 1 -$ P (all components in i fail)

$= 1 - [$ P(one component in i fails)$]^n$

$= 1 - [1 - P_i]^n$

Assume a budget of \$16 and use dynamic programming to design the most reliable circuit.

CHAPTER 8 MULTIOBJECTIVE OPTIMIZATION

8.1 THE PROBLEM

Most systems either produce several different outputs or serve a variety of purposes at once. A factory almost always makes distinct products: an automobile factory may make cars and trucks; a refinery can turn crude petroleum into a range of distillates. Service facilities likewise typically accomplish several goals: schools, for instance, both teach children and provide custodial care so parents can work outside the home; a dam on a river may both control floods and provide some recreational benefits associated with its reservoir.

The preceding chapters have all dealt with optimization over only one objective, benefit or cost, in contrast to the reality of the multiple outputs for any system. This focus is based on two reasons. The first is that it is often realistic to assume that multiple outputs can all be translated into a single overriding objective. The managers of a factory producing cars and trucks may, for example, really be concerned with profits rather than types of vehicles; they may thus be quite prepared to express each different output in terms of its profitability. The second reason for the focus on single objective optimization is that it is virtually impossible to achieve an optimum over many objectives at once.

The essential difficulty with multiobjective optimization is that the meaning of the optimum is not defined so long as we deal with multiple objectives that are truly different. For example, suppose that we are trying to determine the best design of a system of dams on a river, with the objectives of promoting "national income," reducing "deaths by flooding," and increasing "employment." Some designs will be more profitable, but less effective at reducing deaths. How can we state which is better when the objectives are so different, and measured in such