CHAPTER 10

Dynamic Programming

Dynamic programming is a useful mathematical technique for making a sequence of interrelated decisions. It provides a systematic procedure for determining the optimal combination of decisions.

In contrast to linear programming, there does not exist a standard mathematical formulation of "the" dynamic programming problem. Rather, dynamic programming is a general type of approach to problem solving, and the particular equations used must be developed to fit each situation. Therefore, a certain degree of ingenuity and insight into the general structure of dynamic programming problems is required to recognize when and how a problem can be solved by dynamic programming procedures. These abilities can best be developed by an exposure to a wide variety of dynamic programming applications and a study of the characteristics that are common to all these situations. A large number of illustrative examples are presented for this purpose.

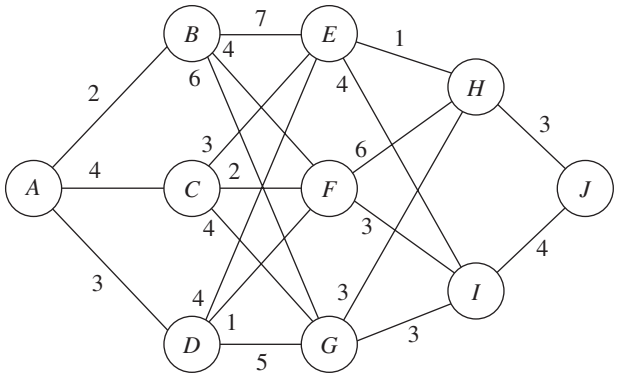
10.1 A PROTOTYPE EXAMPLE FOR DYNAMIC PROGRAMMING

EXAMPLE 1 The Stagecoach Problem

The STAGECOACH PROBLEM is a problem specially constructed[1] to illustrate the features and to introduce the terminology of dynamic programming. It concerns a mythical fortune seeker in Missouri who decided to go west to join the gold rush in California during the mid-19th century. The journey would require traveling by stagecoach through unsettled country where there was serious danger of attack by marauders. Although his starting point and destination were fixed, he had considerable choice as to which states (or territories that subsequently became states) to travel through en route. The possible routes are shown in Fig. 10.1, where each state is represented by a circled letter and the direction of travel is always from left to right in the diagram. Thus, four stages (stagecoach runs) were required to travel from his point of embarkation in state *A* (Missouri) to his destination in state *J* (California).

This fortune seeker was a prudent man who was quite concerned about his safety. After some thought, he came up with a rather clever way of determining the safest route. Life insurance policies were offered to stagecoach passengers. Because the cost of the policy

[1]This problem was developed by Professor Harvey M. Wagner while he was at Stanford University.



■ **FIGURE 10.1**
The road system and costs for the stagecoach problem.

for taking any given stagecoach run was based on a careful evaluation of the safety of that run, the safest route should be the one with the cheapest total life insurance policy.

The cost for the standard policy on the stagecoach run from state i to state j, which will be denoted by c_{ij}, is

	B	C	D
A	2	4	3

	E	F	G
B	7	4	6
C	3	2	4
D	4	1	5

	H	I
E	1	4
F	6	3
G	3	3

	J
H	3
I	4

These costs are also shown in Fig. 10.1.

We shall now focus on the question of which route minimizes the total cost of the policy.

Solving the Problem

First note that the shortsighted approach of selecting the cheapest run offered by each successive stage need not yield an overall optimal decision. Following this strategy would give the route $A \rightarrow B \rightarrow F \rightarrow I \rightarrow J$, at a total cost of 13. However, sacrificing a little on one stage may permit greater savings thereafter. For example, $A \rightarrow D \rightarrow F$ is cheaper overall than $A \rightarrow B \rightarrow F$.

One possible approach to solving this problem is to use trial and error.[2] However, the number of possible routes is large (18), and having to calculate the total cost for each route is not an appealing task.

Fortunately, dynamic programming provides a solution with much less effort than exhaustive enumeration. (The computational savings are enormous for larger versions of this problem.) Dynamic programming starts with a small portion of the original problem and finds the optimal solution for this smaller problem. It then gradually enlarges the problem, finding the current optimal solution from the preceding one, until the original problem is solved in its entirety.

[2]This problem also can be formulated as a *shortest-path problem* (see Sec. 9.3), where *costs* here play the role of *distances* in the shortest-path problem. The algorithm presented in Sec. 9.3 actually uses the philosophy of dynamic programming. However, because the present problem has a fixed number of stages, the dynamic programming approach presented here is even better.

For the stagecoach problem, we start with the smaller problem where the fortune seeker has nearly completed his journey and has only one more stage (stagecoach run) to go. The obvious optimal solution for this smaller problem is to go from his current state (whatever it is) to his ultimate destination (state J). At each subsequent iteration, the problem is enlarged by increasing by 1 the number of stages left to go to complete the journey. For this enlarged problem, the optimal solution for where to go next from each possible state can be found relatively easily from the results obtained at the preceding iteration. The details involved in implementing this approach follow.

Formulation. Let the decision variables x_n ($n = 1, 2, 3, 4$) be the immediate destination on stage n (the nth stagecoach run to be taken). Thus, the route selected is $A \rightarrow x_1 \rightarrow x_2 \rightarrow x_3 \rightarrow x_4$, where $x_4 = J$.

Let $f_n(s, x_n)$ be the total cost of the best overall *policy* for the *remaining* stages, given that the fortune seeker is in state s ready to start stage n, and selects x_n as the immediate destination. Given s and n, let x_n^* denote any value of x_n (not necessarily unique) that minimizes $f_n(s, x_n)$, and let $f_n^*(s)$ be the corresponding minimum value of $f_n(s, x_n)$. Thus,

$$f_n^*(s) = \min_{x_n} f_n(s, x_n) = f_n(s, x_n^*),$$

where

$$\begin{aligned} f_n(s, x_n) &= \text{immediate cost (stage } n) + \text{minimum future cost (stages } n + 1 \text{ onward)} \\ &= c_{sx_n} + f_{n+1}^*(x_n). \end{aligned}$$

The value of c_{sx_n} is given by the preceding tables for c_{ij} by setting $i = s$ (the current state) and $j = x_n$ (the immediate destination). Because the ultimate destination (state J) is reached at the end of stage 4, $f_5^*(J) = 0$.

The objective is to find $f_1^*(A)$ and the corresponding route. Dynamic programming finds it by successively finding $f_4^*(s)$, $f_3^*(s)$, $f_2^*(s)$, for each of the possible states s and then using $f_2^*(s)$ to solve for $f_1^*(A)$.[3]

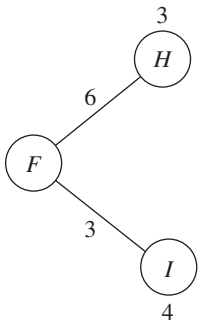
Solution Procedure. When the fortune seeker has only one more stage to go ($n = 4$), his route thereafter is determined entirely by his current state s (either H or I) and his final destination $x_4 = J$, so the route for this final stagecoach run is $s \rightarrow J$. Therefore, since $f_4^*(s) = f_4(s, J) = c_{s,J}$, the immediate solution to the $n = 4$ problem is

$n = 4$:

s	$f_4^*(s)$	x_4^*
H	3	J
I	4	J

When the fortune seeker has two more stages to go ($n = 3$), the solution procedure requires a few calculations. For example, suppose that the fortune seeker is in state F. Then, as depicted below, he must next go to either state H or I at an immediate cost of $c_{F,H} = 6$ or $c_{F,I} = 3$, respectively. If he chooses state H, the minimum additional cost after he reaches there is given in the preceding table as $f_4^*(H) = 3$, as shown above the H node in the diagram. Therefore, the total cost for this decision is $6 + 3 = 9$. If he chooses state I instead, the total cost is $3 + 4 = 7$, which is smaller. Therefore, the optimal choice is this latter one, $x_3^* = I$, because it gives the minimum cost $f_3^*(F) = 7$.

[3]Because this procedure involves moving *backward* stage by stage, some writers also count n backward to denote the number of *remaining stages* to the destination. We use the more natural *forward counting* for greater simplicity.

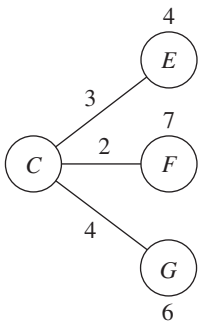


Similar calculations need to be made when you start from the other two possible states $s = E$ and $s = G$ with two stages to go. Try it, proceeding both graphically (Fig. 10.1) and algebraically [combining c_{ij} and $f_4^*(s)$ values], to verify the following complete results for the $n = 3$ problem.

$n = 3$:

s	$f_3(s, x_3) = c_{sx_3} + f_4^*(x_3)$: $x_3 = H$	$x_3 = I$	$f_3^*(s)$	x_3^*
E	4	8	4	H
F	9	7	7	I
G	6	7	6	H

The solution for the second-stage problem ($n = 2$), where there are three stages to go, is obtained in a similar fashion. In this case, $f_2(s, x_2) = c_{sx_2} + f_3^*(x_2)$. For example, suppose that the fortune seeker is in state C, as depicted below.



He must next go to state E, F, or G at an immediate cost of $c_{C,E} = 3$, $c_{C,F} = 2$, or $c_{C,G} = 4$, respectively. After getting there, the minimum additional cost for stage 3 to the end is given by the $n = 3$ table as $f_3^*(E) = 4$, $f_3^*(F) = 7$, or $f_3^*(G) = 6$, respectively, as shown above the E and F nodes and below the G node in the preceding diagram. The resulting calculations for the three alternatives are summarized below.

$x_2 = E$: $\quad f_2(C, E) = c_{C,E} + f_3^*(E) = 3 + 4 = 7.$
$x_2 = F$: $\quad f_2(C, F) = c_{C,F} + f_3^*(F) = 2 + 7 = 9.$
$x_2 = G$: $\quad f_2(C, G) = c_{C,G} + f_3^*(G) = 4 + 6 = 10.$

The minimum of these three numbers is 7, so the minimum total cost from state C to the end is $f_2^*(C) = 7$, and the immediate destination should be $x_2^* = E$.

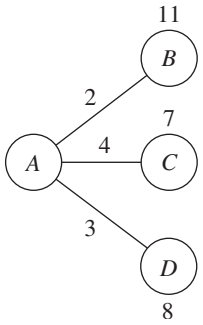
Making similar calculations when you start from state B or D (try it) yields the following results for the $n = 2$ problem:

$n = 2$:

s \ x_2	$f_2(s, x_2) = c_{sx_2} + f_3^*(x_2)$			$f_2^*(s)$	x_2^*
	E	**F**	**G**		
B	11	11	12	11	*E* or *F*
C	7	9	10	7	*E*
D	8	8	11	8	*E* or *F*

In the first and third rows of this table, note that E and F tie as the minimizing value of x_2, so the immediate destination from either state B or D should be $x_2^* = E$ or F.

Moving to the first-stage problem ($n = 1$), with all four stages to go, we see that the calculations are similar to those just shown for the second-stage problem ($n = 2$), except now there is just *one* possible starting state $s = A$, as depicted below.



These calculations are summarized next for the three alternatives for the immediate destination:

$x_1 = B$: $f_1(A, B) = c_{A,B} + f_2^*(B) = 2 + 11 = 13.$
$x_1 = C$: $f_1(A, C) = c_{A,C} + f_2^*(C) = 4 + 7 = 11.$
$x_1 = D$: $f_1(A, D) = c_{A,D} + f_2^*(D) = 3 + 8 = 11.$

Since 11 is the minimum, $f_1^*(A) = 11$ and $x_1^* = C$ or D, as shown in the following table.

$n = 1$:

s \ x_1	$f_1(s, x_1) = c_{sx_1} + f_2^*(x_1)$			$f_1^*(s)$	x_1^*
	B	**C**	**D**		
A	13	11	11	11	*C* or *D*

An optimal solution for the entire problem can now be identified from the four tables. Results for the $n = 1$ problem indicate that the fortune seeker should go initially to either state C or state D. Suppose that he chooses $x_1^* = C$. For $n = 2$, the result for $s = C$ is $x_2^* = E$. This result leads to the $n = 3$ problem, which gives $x_3^* = H$ for $s = E$, and the $n = 4$ problem yields $x_4^* = J$ for $s = H$. Hence, one optimal route is $A \rightarrow C \rightarrow E \rightarrow H \rightarrow J$. Choosing $x_1^* = D$ leads to the other two optimal routes $A \rightarrow D \rightarrow E \rightarrow H \rightarrow J$ and $A \rightarrow D \rightarrow F \rightarrow I \rightarrow J$. They all yield a total cost of $f_1^*(A) = 11$.

These results of the dynamic programming analysis also are summarized in Fig. 10.2. Note how the two arrows for stage 1 come from the first and last columns of the $n = 1$ table and the resulting cost comes from the next-to-last column. Each of the other

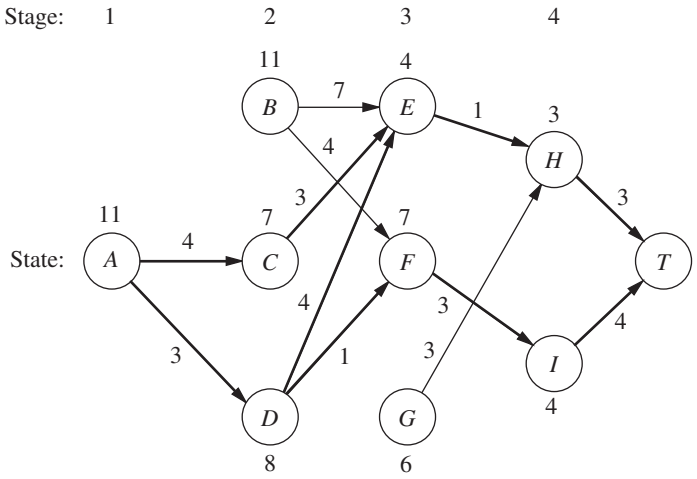


FIGURE 10.2
Graphical display of the dynamic programming solution of the stagecoach problem. Each arrow shows an optimal policy decision (the best immediate destination) from that state, where the number by the state is the resulting cost from there to the end. Following the boldface arrows from *A* to *T* gives the three optimal solutions (the three routes giving the minimum total cost of 11).

arrows (and the resulting cost) comes from one row in one of the other tables in just the same way.

You will see in the next section that the special terms describing the particular context of this problem—*stage, state,* and *policy*—actually are part of the general terminology of dynamic programming with an analogous interpretation in other contexts.

10.2 CHARACTERISTICS OF DYNAMIC PROGRAMMING PROBLEMS

The stagecoach problem is a literal prototype of dynamic programming problems. In fact, this example was purposely designed to provide a literal physical interpretation of the rather abstract structure of such problems. Therefore, one way to recognize a situation that can be formulated as a dynamic programming problem is to notice that its basic structure is analogous to the stagecoach problem.

These basic features that characterize dynamic programming problems are presented and discussed here.

1. The problem can be divided into **stages**, with a **policy decision** required at each stage.

The stagecoach problem was literally divided into its four stages (stagecoaches) that correspond to the four legs of the journey. The policy decision at each stage was which life insurance policy to choose (i.e., which destination to select for the next stagecoach ride). Similarly, other dynamic programming problems require making a *sequence of interrelated decisions,* where each decision corresponds to one stage of the problem.

2. Each stage has a number of **states** associated with the beginning of that stage.

The states associated with each stage in the stagecoach problem were the states (or territories) in which the fortune seeker could be located when embarking on that particular leg of the journey. In general, the states are the various *possible conditions* in which the system might be at that stage of the problem. The number of states may be either finite (as in the stagecoach problem) or infinite (as in some subsequent examples).

3. The effect of the policy decision at each stage is to *transform the current state to a state associated with the beginning of the next stage* (possibly according to a probability distribution).

The fortune seeker's decision as to his next destination led him from his current state to the next state on his journey. This procedure suggests that dynamic programming

problems can be interpreted in terms of the *networks* described in Chap. 9. Each *node* would correspond to a *state.* The network would consist of columns of nodes, with each *column* corresponding to a *stage,* so that the flow from a node can go only to a node in the next column to the right. The links from a node to nodes in the next column correspond to the possible policy decisions on which state to go to next. The value assigned to each link usually can be interpreted as the *immediate contribution* to the objective function from making that policy decision. In most cases, the objective corresponds to finding either the *shortest* or the *longest path* through the network.

4. The solution procedure is designed to find an **optimal policy** for the overall problem, i.e., a prescription of the optimal policy decision at each stage for *each* of the possible states.

   For the stagecoach problem, the solution procedure constructed a table for each stage (n) that prescribed the optimal decision (x_n^*) for *each* possible state (s). Thus, in addition to identifying three *optimal solutions* (optimal routes) for the overall problem, the results show the fortune seeker how he should proceed if he gets detoured to a state that is not on an optimal route. For any problem, dynamic programming provides this kind of *policy* prescription of what to do under every possible circumstance (which is why the actual decision made upon reaching a particular state at a given stage is referred to as a *policy* decision). Providing this additional information beyond simply specifying an optimal solution (optimal sequence of decisions) can be helpful in a variety of ways, including sensitivity analysis.

5. Given the current state, an *optimal policy for the remaining stages* is *independent* of the policy decisions adopted in *previous stages.* Therefore, the optimal immediate decision depends on only the current state and not on how you got there. This is the **principle of optimality** for dynamic programming.

   Given the state in which the fortune seeker is currently located, the optimal life insurance policy (and its associated route) from this point onward is independent of how he got there. For dynamic programming problems in general, knowledge of the current state of the system conveys all the information about its previous behavior necessary for determining the optimal policy henceforth. (This property is the *Markovian property,* discussed in Sec. 16.2.) Any problem lacking this property cannot be formulated as a dynamic programming problem.

6. The solution procedure begins by finding the *optimal policy for the last stage.*

   The optimal policy for the last stage prescribes the optimal policy decision for *each* of the possible states at that stage. The solution of this one-stage problem is usually trivial, as it was for the stagecoach problem.

7. A **recursive relationship** that identifies the optimal policy for stage n, given the optimal policy for stage $n + 1$, is available.

   For the stagecoach problem, this recursive relationship was

$$f_n^*(s) = \min_{x_n} \{c_{sx_n} + f_{n+1}^*(x_n)\}.$$

   Therefore, finding the *optimal policy decision* when you start in state s at stage n requires finding the minimizing value of x_n. For this particular problem, the corresponding minimum cost is achieved by using this value of x_n and then following the optimal policy when you start in state x_n at stage $n + 1$.

   The precise form of the recursive relationship differs somewhat among dynamic programming problems. However, notation analogous to that introduced in the preceding section will continue to be used here, as summarized below.

   N = number of stages.

   n = label for current stage ($n = 1, 2, \ldots, N$).

   s_n = current *state* for stage n.

x_n = decision variable for stage n.

x_n^* = optimal value of x_n (given s_n).

$f_n(s_n, x_n)$ = contribution of stages $n, n+1, \ldots, N$ to objective function if system starts in state s_n at stage n, immediate decision is x_n, and optimal decisions are made thereafter.

$f_n^*(s_n) = f_n(s_n, x_n^*)$.

The recursive relationship will always be of the form

$$f_n^*(s_n) = \max_{x_n} \{f_n(s_n, x_n)\} \quad \text{or} \quad f_n^*(s_n) = \min_{x_n} \{f_n(s_n, x_n)\},$$

where $f_n(s_n, x_n)$ would be written in terms of s_n, x_n, $f_{n+1}^*(s_{n+1})$, and probably some measure of the immediate contribution of x_n to the objective function. It is the inclusion of $f_{n+1}^*(s_{n+1})$ on the right-hand side, so that $f_n^*(s_n)$ is defined in terms of $f_{n+1}^*(s_{n+1})$, that makes the expression for $f_n^*(s_n)$ a recursive relationship.

The recursive relationship keeps recurring as we move backward stage by stage. When the current stage number n is decreased by 1, the new $f_n^*(s_n)$ function is derived by using the $f_{n+1}^*(s_{n+1})$ function that was just derived during the preceding iteration, and then this process keeps repeating. This property is emphasized in the next (and final) characteristic of dynamic programming.

8. When we use this recursive relationship, the solution procedure starts at the end and moves *backward* stage by stage—each time finding the optimal policy for that stage—until it finds the optimal policy starting at the *initial* stage. This optimal policy immediately yields an optimal solution for the entire problem, namely, x_1^* for the initial state s_1, then x_2^* for the resulting state s_2, then x_3^* for the resulting state s_3, and so forth to x_N^* for the resulting stage s_N.

This backward movement was demonstrated by the stagecoach problem, where the optimal policy was found successively beginning in each state at stages 4, 3, 2, and 1, respectively.[4] For all dynamic programming problems, a table such as the following would be obtained for each stage ($n = N, N-1, \ldots, 1$).

s_n \ x_n	$f_n(s_n, x_n)$	$f_n^*(s_n)$	x_n^*

When this table is finally obtained for the initial stage ($n = 1$), the problem of interest is solved. Because the initial state is known, the initial decision is specified by x_1^* in this table. The optimal value of the other decision variables is then specified by the other tables in turn according to the state of the system that results from the preceding decisions.

10.3 DETERMINISTIC DYNAMIC PROGRAMMING

This section further elaborates upon the dynamic programming approach to *deterministic* problems, where the *state* at the *next stage* is *completely determined* by the *state* and *policy decision* at the *current stage*. The *probabilistic* case, where there is a probability distribution for what the next state will be, is discussed in the next section.

[4]Actually, for this problem the solution procedure can move *either* backward or forward. However, for many problems (especially when the stages correspond to *time periods*), the solution procedure *must* move backward.

Deterministic dynamic programming can be described diagrammatically as shown in Fig. 10.3. Thus, at stage n the process will be in some state s_n. Making policy decision x_n then moves the process to some state s_{n+1} at stage $n + 1$. The contribution *thereafter* to the objective function under an optimal policy has been previously calculated to be $f^*_{n+1}(s_{n+1})$. The policy decision x_n also makes some contribution to the objective function. Combining these two quantities in an appropriate way provides $f_n(s_n, x_n)$, the contribution of stages n onward to the objective function. Optimizing with respect to x_n then gives $f^*_n(s_n) = f_n(s_n, x^*_n)$. After x^*_n and $f^*_n(s_n)$ are found for each possible value of s_n, the solution procedure is ready to move back one stage.

One way of categorizing deterministic dynamic programming problems is by the *form of the objective function.* For example, the objective might be to minimize the sum of the contributions from the individual stages (as for the stagecoach problem), or to maximize such a sum, or to minimize a product of such terms, and so on. Another categorization is in terms of the nature of the *set of states* for the respective stages. In particular, states s_n might be representable by a *discrete* state variable (as for the stagecoach problem) or by a *continuous* state variable, or perhaps a state *vector* (more than one variable) is required. Similarly, the decision variables $(x_1, x_2, \ldots, x_N)$ also can be either discrete or continuous.

Several examples are presented to illustrate these various possibilities. More importantly, they illustrate that these apparently major differences are actually quite inconsequential (except in terms of computational difficulty) because the underlying basic structure shown in Fig. 10.3 always remains the same.

The first new example arises in a much different context from the stagecoach problem, but it has the same *mathematical formulation* except that the objective is to *maximize* rather than minimize a sum.

EXAMPLE 2 Distributing Medical Teams to Countries

The WORLD HEALTH COUNCIL is devoted to improving health care in the underdeveloped countries of the world. It now has five medical teams available to allocate among three such countries to improve their medical care, health education, and training programs. Therefore, the council needs to determine how many teams (if any) to allocate to each of these countries to maximize the total effectiveness of the five teams. The teams must be kept intact, so the number allocated to each country must be an integer.

The measure of performance being used is *additional person-years of life.* (For a particular country, this measure equals the *increased life expectancy* in years times the country's population.) Table 10.1 gives the estimated additional person-years of life (in multiples of 1,000) for each country for each possible allocation of medical teams.

Which allocation maximizes the measure of performance?

Formulation. This problem requires making three *interrelated decisions,* namely, how many medical teams to allocate to each of the three countries. Therefore, even though there is no fixed sequence, these three countries can be considered as the three stages in

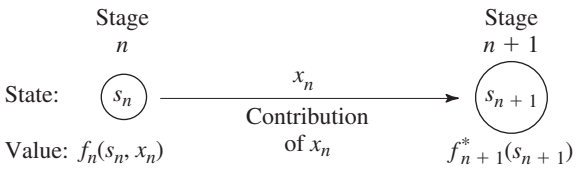


FIGURE 10.3
The basic structure for deterministic dynamic programming.

An Application Vignette

Six days after Saddam Hussein ordered his Iraqi military forces to invade Kuwait on August 2, 1990, the United States began the long process of deploying many of its own military units and cargo to the region. After developing a coalition force from 35 nations led by the United States, the military operation called **Operation Desert Storm** was launched on January 17, 1991, to expel the Iraqi troops from Kuwait. This led to a decisive victory for the coalition forces, which liberated Kuwait and penetrated Iraq.

The logistical challenge involved in quickly transporting the needed troops and cargo to the war zone was a daunting one. A typical airlift mission carrying troops and cargo from the United States to the Persian Gulf required a three-day round-trip, visited seven or more different airfields, burned almost one million pounds of fuel, and cost $280,000. During Operation Desert Storm, the Military Airlift Command (MAC) averaged more than 100 such missions daily as it managed the largest airlift in history.

To meet this challenge, operations research was applied to develop the decision support systems needed to schedule and route each airlift mission. The OR technique used to drive this process was *dynamic programming*. The stages in the dynamic programming formulation correspond to the airfields in the network of flight legs relevant to the mission. For a given airfield, the states are characterized by the departure time from the airfield and the remaining available duty for the current crew. The objective function to be minimized is a weighted sum of several measures of performance: the lateness of deliveries, the flying time of the mission, the ground time, and the number of crew changes. The constraints include a lower bound on the load carried by the mission and upper bounds on the availability of crew and ground-support resources at airfields.

This application of dynamic programming had a dramatic impact on the ability to deliver the necessary cargo and personnel to the Persian gulf quickly to support Operation Desert Storm. For example, when speaking to the developers of this approach, MAC's deputy chief of staff for operations and transportation is quoted as saying, "I guarantee you that we could not have done that (the deployment to the Persian Gulf) without your help and the contributions you made to (the decision support systems)—we absolutely could not have done that."

Source: M. C. Hilliard, R. S. Solanki, C. Liu, I. K. Busch, G. Harrison, and R. D. Kraemer: "Scheduling the Operation Desert Storm Airlift: An Advanced Automated Scheduling Support System," *Interfaces*, **22**(1): 131–146, Jan.–Feb. 1992.

TABLE 10.1 Data for the World Health Council problem

Medical Teams	Thousands of Additional Person-Years of Life		
	Country		
	1	2	3
0	0	0	0
1	45	20	50
2	70	45	70
3	90	75	80
4	105	110	100
5	120	150	130

a dynamic programming formulation. The decision variables x_n ($n = 1, 2, 3$) are the number of teams to allocate to stage (country) n.

The identification of the states may not be readily apparent. To determine the states, we ask questions such as the following. What is it that changes from one stage to the next? Given that the decisions have been made at the previous stages, how can the status of the situation at the current stage be described? What information about the current state of affairs is necessary to determine the optimal policy hereafter? On these bases, an appropriate choice for the "state of the system" is

s_n = number of medical teams still available for allocation to remaining countries (n, . . . , 3).

Thus, at stage 1 (country 1), where all three countries remain under consideration for allocations, $s_1 = 5$. However, at stage 2 or 3 (country 2 or 3), s_n is just 5 minus the number of teams allocated at preceding stages, so that the sequence of states is

$$s_1 = 5, \qquad s_2 = 5 - x_1, \qquad s_3 = s_2 - x_2.$$

With the dynamic programming procedure of solving backward stage by stage, when we are solving at stage 2 or 3, we shall not yet have solved for the allocations at the preceding stages. Therefore, we shall consider every possible state we could be in at stage 2 or 3, namely, $s_n = 0, 1, 2, 3, 4$, or 5.

Figure 10.4 shows the states to be considered at each stage. The links (line segments) show the possible transitions in states from one stage to the next from making a feasible allocation of medical teams to the country involved. The numbers shown next to the links are the corresponding contributions to the measure of performance, where these numbers

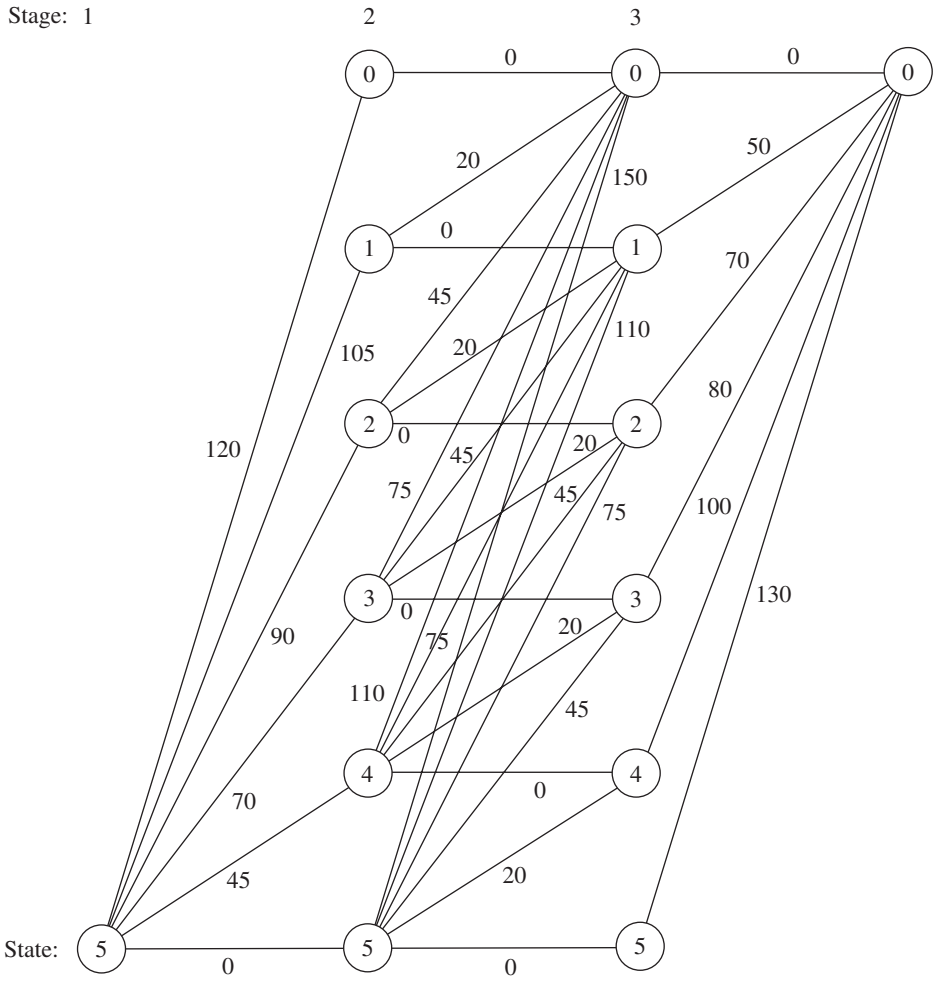


FIGURE 10.4 Graphical display of the World Health Council problem, showing the possible states at each stage, the possible transitions in states, and the corresponding contributions to the measure of performance.

come from Table 10.1. From the perspective of this figure, the overall problem is to find the path from the initial state 5 (beginning stage 1) to the final state 0 (after stage 3) that maximizes the sum of the numbers along the path.

To state the overall problem mathematically, let $p_i(x_i)$ be the measure of performance from allocating x_i medical teams to country i, as given in Table 10.1. Thus, the objective is to choose x_1, x_2, x_3 so as to

$$\text{Maximize} \quad \sum_{i=1}^{3} p_i(x_i),$$

subject to

$$\sum_{i=1}^{3} x_i = 5,$$

and

x_i are nonnegative integers.

Using the notation presented in Sec. 10.2, we see that $f_n(s_n, x_n)$ is

$$f_n(s_n, x_n) = p_n(x_n) + \max \sum_{i=n+1}^{3} p_i(x_i),$$

where the maximum is taken over $x_{n+1}, \ldots, x_3$ such that

$$\sum_{i=n}^{3} x_i = s_n$$

and the x_i are nonnegative integers, for $n = 1, 2, 3$. In addition,

$$f_n^*(s_n) = \max_{x_n = 0,1,\ldots,s_n} f_n(s_n, x_n)$$

Therefore,

$$f_n(s_n, x_n) = p_n(x_n) + f_{n+1}^*(s_n - x_n)$$

(with f_4^* defined to be zero). These basic relationships are summarized in Fig. 10.5.

Consequently, the *recursive relationship* relating functions f_1^*, f_2^*, and f_3^* for this problem is

$$f_n^*(s_n) = \max_{x_n = 0,1,\ldots,s_n} \{p_n(x_n) + f_{n+1}^*(s_n - x_n)\}, \qquad \text{for } n = 1, 2.$$

For the last stage ($n = 3$),

$$f_3^*(s_3) = \max_{x_3 = 0,1,\ldots,s_3} p_3(x_3).$$

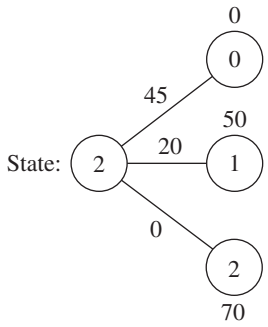
The resulting dynamic programming calculations are given next.

Solution Procedure. Beginning with the last stage ($n = 3$), we note that the values of $p_3(x_3)$ are given in the last column of Table 10.1 and these values keep increasing as we move down the column. Therefore, with s_3 medical teams still available for allocation to country 3, the maximum of $p_3(x_3)$ is automatically achieved by allocating all s_3 teams; so $x_3^* = s_3$ and $f_3^*(s_3) = p_3(s_3)$, as shown in the following table.

n = 3:

s_3	$f_3^*(s_3)$	x_3^*
0	0	0
1	50	1
2	70	2
3	80	3
4	100	4
5	130	5

We now move backward to start from the next-to-last stage ($n = 2$). Here, finding x_2^* requires calculating and comparing $f_2(s_2, x_2)$ for the alternative values of x_2, namely, $x_2 = 0, 1, \ldots, s_2$. To illustrate, we depict this situation when $s_2 = 2$ graphically:

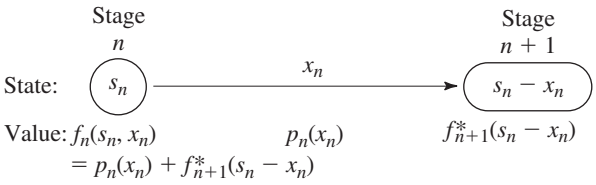


This diagram corresponds to Fig. 10.5 except that all three possible states at stage 3 are shown. Thus, if $x_2 = 0$, the resulting state at stage 3 will be $s_2 - x_2 = 2 - 0 = 2$, whereas $x_2 = 1$ leads to state 1 and $x_2 = 2$ leads to state 0. The corresponding values of $p_2(x_2)$ from the country 2 column of Table 10.1 are shown along the links, and the values of $f_3^*(s_2 - x_2)$ from the $n = 3$ table are given next to the stage 3 nodes. The required calculations for this case of $s_2 = 2$ are summarized below.

Formula: $f_2(2, x_2) = p_2(x_2) + f_3^*(2 - x_2)$.
$p_2(x_2)$ is given in the country 2 column of Table 10.1.
$f_3^*(2 - x_2)$ is given in the $n = 3$ table above.

$x_2 = 0$: $f_2(2, 0) = p_2(0) + f_3^*(2) = 0 + 70 = 70.$
$x_2 = 1$: $f_2(2, 1) = p_2(1) + f_3^*(1) = 20 + 50 = 70.$
$x_2 = 2$: $f_2(2, 2) = p_2(2) + f_3^*(0) = 45 + 0 = 45.$

Because the objective is *maximization,* $x_2^* = 0$ or 1 with $f_2^*(2) = 70$.



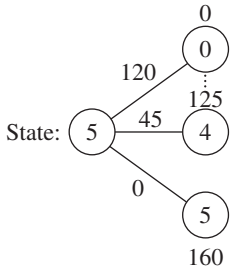
■ **FIGURE 10.5**
The basic structure for the World Health Council problem.

Proceeding in a similar way with the other possible values of s_2 (try it) yields the following table.

$n = 2$:

s_2 \ x_2	$f_2(s_2, x_2) = p_2(x_2) + f_3^*(s_2 - x_2)$						$f_2^*(s_2)$	x_2^*
	0	**1**	**2**	**3**	**4**	**5**		
0	0						0	0
1	50	20					50	0
2	70	70	45				70	0 or 1
3	80	90	95	75			95	2
4	100	100	115	125	110		125	3
5	130	120	125	145	160	150	160	4

We now are ready to move backward to solve the original problem where we are starting from stage 1 ($n = 1$). In this case, the only state to be considered is the starting state of $s_1 = 5$, as depicted below.



Since allocating x_1 medical teams to country 1 leads to a state of $5 - x_1$ at stage 2, a choice of $x_1 = 0$ leads to the bottom node on the right, $x_1 = 1$ leads to the next node up, and so forth up to the top node with $x_1 = 5$. The corresponding $p_1(x_1)$ values from Table 10.1 are shown next to the links. The numbers next to the nodes are obtained from the $f_2^*(s_2)$ column of the $n = 2$ table. As with $n = 2$, the calculation needed for each alternative value of the decision variable involves adding the corresponding link value and node value, as summarized below.

Formula: $f_1(5, x_1) = p_1(x_1) + f_2^*(5 - x_1)$.
$p_1(x_1)$ is given in the country 1 column of Table 10.1.
$f_2^*(5 - x_1)$ is given in the $n = 2$ table.

$x_1 = 0$: $f_1(5, 0) = p_1(0) + f_2^*(5) = \quad 0 + 160 = 160.$
$x_1 = 1$: $f_1(5, 1) = p_1(1) + f_2^*(4) = \quad 45 + 125 = 170.$
$\vdots$
$x_1 = 5$: $f_1(5, 5) = p_1(5) + f_2^*(0) = 120 + \quad 0 = 120.$

The similar calculations for $x_1 = 2, 3, 4$ (try it) verify that $x_1^* = 1$ with $f_1^*(5) = 170$, as shown in the following table.

$n = 1$:

s_1 \ x_2	$f_1(s_1, x_1) = p_1(x_1) + f_2^*(s_1 - x_1)$						$f_1^*(s_1)$	x_1^*
	0	**1**	**2**	**3**	**4**	**5**		
5	160	170	165	160	155	120	170	1

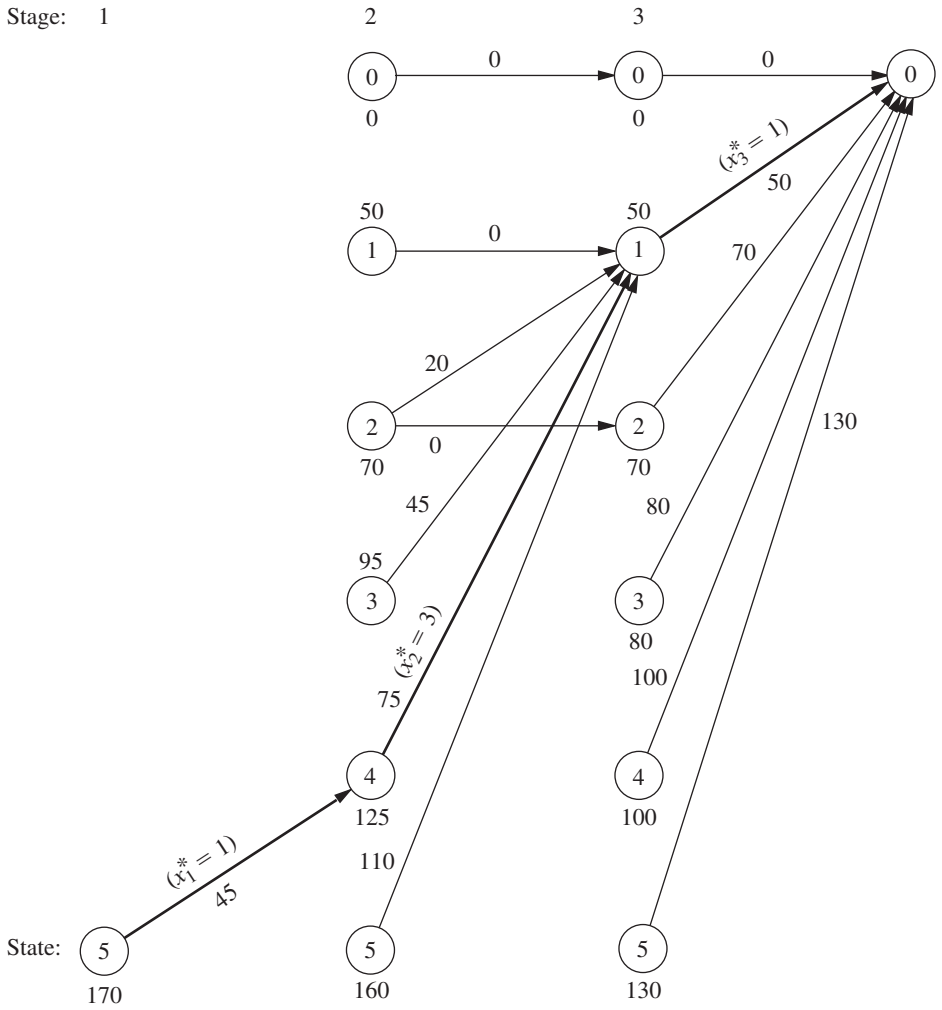


FIGURE 10.6
Graphical display of the dynamic programming solution of the World Health Council problem. An arrow from state s_n to state s_{n+1} indicates that an optimal policy decision from state s_n is to allocate $(s_n - s_{n+1})$ medical teams to country n. Allocating the medical teams in this way when following the boldfaced arrows from the initial state to the final state gives the optimal solution.

Thus, the optimal solution has $x_1^* = 1$, which makes $s_2 = 5 - 1 = 4$, so $x_2^* = 3$, which makes $s_3 = 4 - 3 = 1$, so $x_3^* = 1$. Since $f_1^*(5) = 170$, this (1, 3, 1) allocation of medical teams to the three countries will yield an estimated total of 170,000 additional person-years of life, which is at least 5,000 more than for any other allocation.

These results of the dynamic programming analysis also are summarized in Fig. 10.6.

A Prevalent Problem Type—The Distribution of Effort Problem

The preceding example illustrates a particularly common type of dynamic programming problem called the *distribution of effort problem.* For this type of problem, there is just one kind of *resource* that is to be allocated to a number of *activities.* The objective is to determine how to distribute the effort (the resource) among the activities most effectively. For the World Health Council example, the resource involved is the medical teams, and the three activities are the health care work in the three countries.

Assumptions. This interpretation of allocating resources to activities should ring a bell for you, because it is the typical interpretation for linear programming problems given at

the beginning of Chap. 3. However, there also are some key differences between the distribution of effort problem and linear programming that help illuminate the general distinctions between dynamic programming and other areas of mathematical programming.

One key difference is that the distribution of effort problem involves only *one resource* (one functional constraint), whereas linear programming can deal with thousands of resources. (In principle, dynamic programming can handle slightly more than one resource, as we shall illustrate in Example 5 by solving the three-resource Wyndor Glass Co. problem, but it quickly becomes very inefficient when the number of resources is increased.)

On the other hand, the distribution of effort problem is far more general than linear programming in other ways. Consider the four assumptions of linear programming presented in Sec. 3.3: proportionality, additivity, divisibility, and certainty. *Proportionality* is routinely violated by nearly all dynamic programming problems, including distribution of effort problems (e.g., Table 10.1 violates proportionality). *Divisibility* also is often violated, as in Example 2, where the decision variables must be integers. In fact, dynamic programming calculations become more complex when divisibility does hold (as in Examples 4 and 5). Although we shall consider the distribution of effort problem only under the assumption of *certainty,* this is not necessary, and many other dynamic programming problems violate this assumption as well (as described in Sec. 10.4).

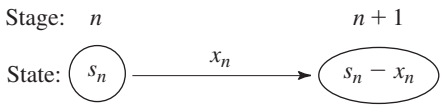
Of the four assumptions of linear programming, the *only* one needed by the distribution of effort problem (or other dynamic programming problems) is *additivity* (or its analog for functions involving a *product* of terms). This assumption is needed to satisfy the *principle of optimality* for dynamic programming (characteristic 5 in Sec. 10.2).

Formulation. Because they always involve allocating one kind of resource to a number of activities, distribution of effort problems always have the following dynamic programming formulation (where the ordering of the activities is arbitrary):

Stage n = activity n ($n = 1, 2, \ldots, N$).
x_n = amount of resource allocated to activity n.
State s_n = amount of resource still available for allocation to remaining activities $(n, \ldots, N)$.

The reason for defining state s_n in this way is that the amount of the resource still available for allocation is precisely the information about the current state of affairs (entering stage n) that is needed for making the allocation decisions for the remaining activities.

When the system starts at stage n in state s_n, the choice of x_n results in the next state at stage $n + 1$ being $s_{n+1} = s_n - x_n$, as depicted below:[5]



Note how the structure of this diagram corresponds to the one shown in Fig. 10.5 for the World Health Council example of a distribution of effort problem. What will differ from one such example to the next is the *rest* of what is shown in Fig. 10.5, namely, the relationship between $f_n(s_n, x_n)$ and $f^*_{n+1}(s_n - x_n)$, and then the resulting *recursive relationship* between the f^*_n and f^*_{n+1} functions. These relationships depend on the particular objective function for the overall problem.

[5]This statement assumes that x_n and s_n are expressed in the same units. If it is more convenient to define x_n as some other quantity such that the amount of the resource allocated to activity n is $a_n x_n$, then $s_{n+1} = s_n - a_n x_n$.

The structure of the next example is similar to the one for the World Health Council because it, too, is a distribution of effort problem. However, its recursive relationship differs in that its objective is to minimize a product of terms for the respective stages.

At first glance, this example may appear *not* to be a deterministic dynamic programming problem because probabilities are involved. However, it does indeed fit our definition because the state at the next stage is completely determined by the state and policy decision at the current stage.

EXAMPLE 3 Distributing Scientists to Research Teams

A government space project is conducting research on a certain engineering problem that must be solved before people can fly safely to Mars. Three research teams are currently trying three different approaches for solving this problem. The estimate has been made that, under present circumstances, the probability that the respective teams—call them 1, 2, and 3—will not succeed is 0.40, 0.60, and 0.80, respectively. Thus, the current probability that all three teams will fail is $(0.40)(0.60)(0.80) = 0.192$. Because the objective is to minimize the probability of failure, two more top scientists have been assigned to the project.

Table 10.2 gives the estimated probability that the respective teams will fail when 0, 1, or 2 additional scientists are added to that team. Only integer numbers of scientists are considered because each new scientist will need to devote full attention to one team. The problem is to determine how to allocate the two additional scientists to minimize the probability that all three teams will fail.

Formulation. Because both Examples 2 and 3 are distribution of effort problems, their underlying structure is actually very similar. In this case, scientists replace medical teams as the kind of resource involved, and research teams replace countries as the activities. Therefore, instead of medical teams being allocated to countries, scientists are being allocated to research teams. The only basic difference between the two problems is in their objective functions.

With so few scientists and teams involved, this problem could be solved very easily by a process of exhaustive enumeration. However, the dynamic programming solution is presented for illustrative purposes.

In this case, stage n ($n = 1, 2, 3$) corresponds to research team n, and the state s_n is the number of new scientists *still available* for allocation to the remaining teams. The decision variables x_n ($n = 1, 2, 3$) are the number of additional scientists allocated to team n.

Let $p_i(x_i)$ denote the probability of failure for team i if it is assigned x_i additional scientists, as given by Table 10.2. If we let Π denote multiplication, the government's objective is to choose x_1, x_2, x_3 so as to

$$\text{Minimize} \quad \prod_{i=1}^{3} p_i(x_i) = p_1(x_1)p_2(x_2)p_3(x_3),$$

TABLE 10.2 Data for the Government Space Project problem

New Scientists	Probability of Failure		
	Team		
	1	2	3
0	0.40	0.60	0.80
1	0.20	0.40	0.50
2	0.15	0.20	0.30

subject to

$$\sum_{i=1}^{3} x_i = 2$$

and

x_i are nonnegative integers.

Consequently, $f_n(s_n, x_n)$ for this problem is

$$f_n(s_n, x_n) = p_n(x_n) \cdot \min \prod_{i=n+1}^{3} p_i(x_i),$$

where the minimum is taken over $x_{n+1}, \ldots, x_3$ such that

$$\sum_{i=n}^{3} x_i = s_n$$

and

x_i are nonnegative integers,

for $n = 1, 2, 3$. Thus,

$$f_n^*(s_n) = \min_{x_n = 0,1,\ldots,s_n} f_n(s_n, x_n),$$

where

$$f_n(s_n, x_n) = p_n(x_n) \cdot f_{n+1}^*(s_n - x_n)$$

(with f_4^* defined to be 1). Figure 10.7 summarizes these basic relationships.

Thus, the *recursive relationship* relating the f_1^*, f_2^*, and f_3^* functions in this case is

$$f_n^*(s_n) = \min_{x_n = 0,1,\ldots,s_n} \{p_n(x_n) \cdot f_{n+1}^*(s_n - x_n)\}, \qquad \text{for } n = 1, 2,$$

and, when $n = 3$,

$$f_3^*(s_3) = \min_{x_3 = 0,1,\ldots,s_3} p_3(x_3).$$

Solution Procedure. The resulting dynamic programming calculations are as follows:

$n = 3$:

s_3	$f_3^*(s_3)$	x_3^*
0	0.80	0
1	0.50	1
2	0.30	2

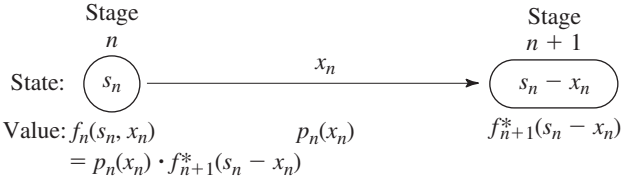


FIGURE 10.7
The basic structure for the government space project problem.

$n = 2$:

s_2 \ x_2	$f_2(s_2, x_2) = p_2(x_2) \cdot f_3^*(s_2 - x_2)$			$f_2^*(s_2)$	x_2^*
	0	1	2		
0	0.48			0.48	0
1	0.30	0.32		0.30	0
2	0.18	0.20	0.16	0.16	2

$n = 1$:

s_1 \ x_1	$f_1(s_1, x_1) = p_1(x_1) \cdot f_2^*(s_1 - x_1)$			$f_1^*(s_1)$	x_1^*
	0	1	2		
2	0.064	0.060	0.072	0.060	1

Therefore, the optimal solution must have $x_1^* = 1$, which makes $s_2 = 2 - 1 = 1$, so that $x_2^* = 0$, which makes $s_3 = 1 - 0 = 1$, so that $x_3^* = 1$. Thus, teams 1 and 3 should each receive one additional scientist. The new probability that all three teams will fail would then be 0.060.

All the examples thus far have had a *discrete* state variable s_n at each stage. Furthermore, they all have been *reversible* in the sense that the solution procedure actually could have moved *either* backward or forward stage by stage. (The latter alternative amounts to renumbering the stages in reverse order and then applying the procedure in the standard way.) This reversibility is a general characteristic of distribution of effort problems such as Examples 2 and 3, since the activities (stages) can be ordered in any desired manner.

The next example is different in both respects. Rather than being restricted to integer values, its state variable s_n at stage n is a *continuous* variable that can take on *any* value over certain intervals. Since s_n now has an infinite number of values, it is no longer possible to consider each of its feasible values individually. Rather, the solution for $f_n^*(s_n)$ and x_n^* must be expressed as *functions* of s_n. Furthermore, this example is *not* reversible because its stages correspond to *time periods,* so the solution procedure *must* proceed backward.

Before proceeding directly to the rather involved example presented next, you might find it helpful at this point to look at the **two additional examples** of deterministic dynamic programming presented in the Worked Examples section of the book's website. The first one involves production and inventory planning over a number of time periods. Like the examples thus far, both the state variable and the decision variable at each stage are discrete. However, this example is not reversible since the stages correspond to time periods. It also is not a distribution of effort problem. The second example is a nonlinear programming problem with two variables and a single constraint. Therefore, even though it is reversible, its state and decision variables are continuous. However, in contrast to the following example (which has four continuous variables and thus four stages), it has only two stages, so it can be solved relatively quickly with dynamic programming and a bit of calculus.

EXAMPLE 4 Scheduling Employment Levels

The workload for the LOCAL JOB SHOP is subject to considerable seasonal fluctuation. However, machine operators are difficult to hire and costly to train, so the manager is reluctant to lay off workers during the slack seasons. He is likewise reluctant to maintain his peak season payroll when it is not required. Furthermore, he is definitely opposed to

overtime work on a regular basis. Since all work is done to custom orders, it is not possible to build up inventories during slack seasons. Therefore, the manager is in a dilemma as to what his policy should be regarding employment levels.

The following estimates are given for the minimum employment requirements during the four seasons of the year for the foreseeable future:

Season	Spring	Summer	Autumn	Winter	Spring
Requirements	255	220	240	200	255

Employment will not be permitted to fall below these levels. Any employment above these levels is wasted at an approximate cost of $2,000 per person per season. It is estimated that the hiring and firing costs are such that the total cost of changing the level of employment from one season to the next is $200 times the square of the difference in employment levels. Fractional levels of employment are possible because of a few part-time employees, and the cost data also apply on a fractional basis.

Formulation. On the basis of the data available, it is not worthwhile to have the employment level go above the peak season requirements of 255. Therefore, spring employment should be at 255, and the problem is reduced to finding the employment level for the other three seasons.

For a dynamic programming formulation, the seasons should be the stages. There are actually an indefinite number of stages because the problem extends into the indefinite future. However, each year begins an identical cycle, and because spring employment is known, it is possible to consider only one cycle of four seasons ending with the spring season, as summarized below.

Stage 1 = summer,
Stage 2 = autumn,
Stage 3 = winter,
Stage 4 = spring.
x_n = employment level for stage n ($n = 1, 2, 3, 4$).
($x_4 = 255$.)

It is necessary that the spring season be the last stage because the optimal value of the decision variable for each state at the last stage must be either known or obtainable without considering other stages. For every other season, the solution for the optimal employment level must consider the effect on costs in the following season.

Let

r_n = minimum employment requirement for stage n,

where these requirements were given earlier as $r_1 = 220$, $r_2 = 240$, $r_3 = 200$, and $r_4 = 255$. Thus, the only feasible values for x_n are

$$r_n \leq x_n \leq 255.$$

Referring to the cost data given in the problem statement, we have

$$\text{Cost for stage } n = 200(x_n - x_{n-1})^2 + 2{,}000(x_n - r_n).$$

Note that the cost at the current stage depends upon only the current decision x_n and the employment in the preceding season x_{n-1}. Thus, the preceding employment level is all the information about the current state of affairs that we need to determine the optimal policy henceforth. Therefore, the state s_n for stage n is

$$\text{State } s_n = x_{n-1}.$$

When $n = 1$, $s_1 = x_0 = x_4 = 255$.

For your ease of reference while working through the problem, a summary of the data is given in Table 10.3 for each of the four stages.

The objective for the problem is to choose x_1, x_2, x_3 (with $x_0 = x_4 = 255$) so as to

$$\text{Minimize} \quad \sum_{i=1}^{4} [200(x_i - x_{i-1})^2 + 2{,}000(x_i - r_i)],$$

subject to

$$r_i \le x_i \le 255, \qquad \text{for } i = 1, 2, 3, 4.$$

Thus, for stage n onward ($n = 1, 2, 3, 4$), since $s_n = x_{n-1}$

$$f_n(s_n, x_n) = 200(x_n - s_n)^2 + 2{,}000(x_n - r_n) + \min_{r_i \le x_i \le 255} \sum_{i=n+1}^{4} [200(x_i - x_{i-1})^2 + 2{,}000(x_i - r_i)],$$

where this summation equals zero when $n = 4$ (because it has no terms). Also,

$$f_n^*(s_n) = \min_{r_n \le x_n \le 255} f_n(s_n, x_n).$$

Hence,

$$f_n(s_n, x_n) = 200(x_n - s_n)^2 + 2{,}000(x_n - r_n) + f_{n+1}^*(x_n)$$

(with f_5^* defined to be zero because costs after stage 4 are irrelevant to the analysis). A summary of these basic relationships is given in Fig. 10.8.

Consequently, the recursive relationship relating the f_n^* functions is

$$f_n^*(s_n) = \min_{r_n \le x_n \le 255} \{200(x_n - s_n)^2 + 2{,}000(x_n - r_n) + f_{n+1}^*(x_n)\}.$$

The dynamic programming approach uses this relationship to identify successively these functions—$f_4^*(s_4)$, $f_3^*(s_3)$, $f_2^*(s_2)$, $f_1^*(255)$—and the corresponding minimizing x_n.

TABLE 10.3 Data for the Local Job Shop problem

n	r_n	Feasible x_n	Possible $s_n = x_{n-1}$	Cost
1	220	$220 \le x_1 \le 255$	$s_1 = 255$	$200(x_1 - 255)^2 + 2{,}000(x_1 - 220)$
2	240	$240 \le x_2 \le 255$	$220 \le s_2 \le 255$	$200(x_2 - x_1)^2 + 2{,}000(x_2 - 240)$
3	200	$200 \le x_3 \le 255$	$240 \le s_3 \le 255$	$200(x_3 - x_2)^2 + 2{,}000(x_3 - 200)$
4	255	$x_4 = 255$	$200 \le s_4 \le 255$	$200(255 - x_3)^2$

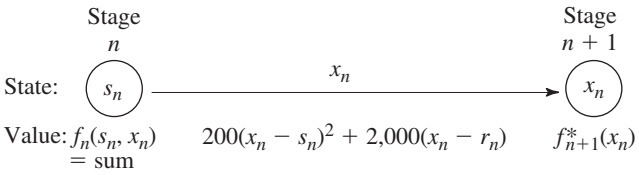


FIGURE 10.8 The basic structure for the Local Job Shop problem.

Solution Procedure. *Stage 4:* Beginning at the last stage ($n = 4$), we already know that $x_4^* = 255$, so the necessary results are

$n = 4$:

s_4	$f_4^*(s_4)$	x_4^*
$200 \le s_4 \le 255$	$200(255 - s_4)^2$	255

Stage 3: For the problem consisting of just the last *two* stages ($n = 3$), the recursive relationship reduces to

$$f_3^*(s_3) = \min_{200 \le x_3 \le 255} \{200(x_3 - s_3)^2 + 2{,}000(x_3 - 200) + f_4^*(x_3)\}$$
$$= \min_{200 \le x_3 \le 255} \{200(x_3 - s_3)^2 + 2{,}000(x_3 - 200) + 200(255 - x_3)^2\},$$

where the possible values of s_3 are $240 \le s_3 \le 255$.

One way to solve for the value of x_3 that minimizes $f_3(s_3, x_3)$ for any particular value of s_3 is the graphical approach illustrated in Fig. 10.9.

However, a faster way is to use *calculus*. We want to solve for the minimizing x_3 in terms of s_3 by considering s_3 to have some fixed (but unknown) value. Therefore, set the first (partial) derivative of $f_3(s_3, x_3)$ with respect to x_3 equal to zero:

$$\frac{\partial}{\partial x_3} f_3(s_3, x_3) = 400(x_3 - s_3) + 2{,}000 - 400(255 - x_3)$$
$$= 400(2x_3 - s_3 - 250)$$
$$= 0,$$

which yields

$$x_3^* = \frac{s_3 + 250}{2}.$$

Because the second derivative is positive, and because this solution lies in the feasible interval for x_3 ($200 \le x_3 \le 255$) for all possible s_3 ($240 \le s_3 \le 255$), it is indeed the desired minimum.

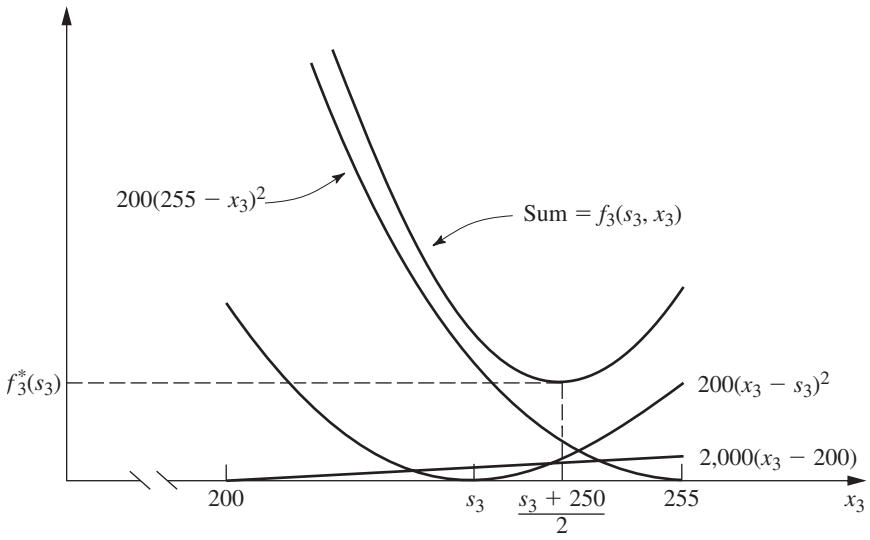


FIGURE 10.9
Graphical solution for $f_3^*(s_3)$ for the Local Job Shop problem.

Note a key difference between the nature of this solution and those obtained for the preceding examples where there were only a few possible states to consider. We now have an *infinite* number of possible states ($240 \le s_3 \le 255$), so it is no longer feasible to solve separately for x_3^* for each possible value of s_3. Therefore, we instead have solved for x_3^* as a *function* of the unknown s_3.

Using

$$f_3^*(s_3) = f_3(s_3, x_3^*) = 200\left(\frac{s_3 + 250}{2} - s_3\right)^2 + 200\left(255 - \frac{s_3 + 250}{2}\right)^2 + 2{,}000\left(\frac{s_3 + 250}{2} - 200\right)$$

and reducing this expression algebraically complete the required results for the third-stage problem, summarized as follows.

$n = 3$:

s_3	$f_3^*(s_3)$	x_3^*
$240 \le s_3 \le 255$	$50(250 - s_3)^2 + 50(260 - s_3)^2 + 1{,}000(s_3 - 150)$	$\frac{s_3 + 250}{2}$

Stage 2: The second-stage ($n = 2$) and first-stage problems ($n = 1$) are solved in a similar fashion. Thus, for $n = 2$,

$$\begin{aligned} f_2(s_2, x_2) &= 200(x_2 - s_2)^2 + 2{,}000(x_2 - r_2) + f_3^*(x_2) \\ &= 200(x_2 - s_2)^2 + 2{,}000(x_2 - 240) \\ &\quad + 50(250 - x_2)^2 + 50(260 - x_2)^2 + 1{,}000(x_2 - 150). \end{aligned}$$

The possible values of s_2 are $220 \le s_2 \le 255$, and the feasible region for x_2 is $240 \le x_2 \le 255$. The problem is to find the minimizing value of x_2 in this region, so that

$$f_2^*(s_2) = \min_{240 \le x_2 \le 255} f_2(s_2, x_2).$$

Setting to zero the partial derivative with respect to x_2:

$$\begin{aligned} \frac{\partial}{\partial x_2} f_2(s_2, x_2) &= 400(x_2 - s_2) + 2{,}000 - 100(250 - x_2) - 100(260 - x_2) + 1{,}000 \\ &= 200(3x_2 - 2s_2 - 240) \\ &= 0 \end{aligned}$$

yields

$$x_2 = \frac{2s_2 + 240}{3}.$$

Because

$$\frac{\partial^2}{\partial x_2^2} f_2(s_2, x_2) = 600 > 0,$$

this value of x_2 is the desired minimizing value *if* it is *feasible* ($240 \le x_2 \le 255$). Over the possible s_2 values ($220 \le s_2 \le 255$), this solution actually is feasible only if $240 \le s_2 \le 255$.

Therefore, we still need to solve for the feasible value of x_2 that minimizes $f_2(s_2, x_2)$ when $220 \le s_2 < 240$. The key to analyzing the behavior of $f_2(s_2, x_2)$ over the feasible region for x_2 again is the partial derivative of $f_2(s_2, x_2)$. When $s_2 < 240$,

$$\frac{\partial}{\partial x_2} f_2(s_2, x_2) > 0, \qquad \text{for } 240 \le x_2 \le 255,$$

so that $x_2 = 240$ is the desired minimizing value.

The next step is to plug these values of x_2 into $f_2(s_2, x_2)$ to obtain $f_2^*(s_2)$ for $s_2 \ge 240$ and $s_2 < 240$. This yields

$n = 2$:

s_2	$f_2^*(s_2)$	x_2^*
$220 \le s_2 \le 240$	$200(240 - s_2)^2 + 115{,}000$	240
$240 \le s_2 \le 255$	$\frac{200}{9}[(240 - s_2)^2 + (255 - s_2)^2 + (270 - s_2)^2] + 2{,}000(s_2 - 195)$	$\frac{2s_2 + 240}{3}$

Stage 1: For the first-stage problem ($n = 1$),

$$f_1(s_1, x_1) = 200(x_1 - s_1)^2 + 2{,}000(x_1 - r_1) + f_2^*(x_1).$$

Because $r_1 = 220$, the feasible region for x_1 is $220 \le x_1 \le 255$. The expression for $f_2^*(x_1)$ will differ in the two portions $220 \le x_1 \le 240$ and $240 \le x_1 \le 255$ of this region. Therefore,

$$f_1(s_1, x_1) = \begin{cases} 200(x_1 - s_1)^2 + 2{,}000(x_1 - 220) + 200(240 - x_1)^2 + 115{,}000, & \text{if } 220 \le x_1 \le 240 \\ 200(x_1 - s_1)^2 + 2{,}000(x_1 - 220) + \frac{200}{9}[(240 - x_1)^2 + (255 - x_1)^2 + (270 - x_1)^2] & \\ \quad + 2{,}000(x_1 - 195), & \text{if } 240 \le x_1 \le 255. \end{cases}$$

Considering first the case where $220 \le x_1 \le 240$, we have

$$\begin{aligned} \frac{\partial}{\partial x_1} f_1(s_1, x_1) &= 400(x_1 - s_1) + 2{,}000 - 400(240 - x_1) \\ &= 400(2x_1 - s_1 - 235). \end{aligned}$$

It is known that $s_1 = 255$ (spring employment), so that

$$\frac{\partial}{\partial x_1} f_1(s_1, x_1) = 800(x_1 - 245) < 0$$

for all $x_1 \le 240$. Therefore, $x_1 = 240$ is the minimizing value of $f_1(s_1, x_1)$ over the region $220 \le x_1 \le 240$.

When $240 \le x_1 \le 255$,

$$\begin{aligned} \frac{\partial}{\partial x_1} f_1(s_1, x_1) &= 400(x_1 - s_1) + 2{,}000 \\ &\quad - \frac{400}{9}[(240 - x_1) + (255 - x_1) + (270 - x_1)] + 2{,}000 \\ &= \frac{400}{3}(4x_1 - 3s_1 - 225). \end{aligned}$$

Because

$$\frac{\partial^2}{\partial x_1^2} f_1(s_1, x_1) > 0 \qquad \text{for all } x_1,$$

set

$$\frac{\partial}{\partial x_1} f_1(s_1, x_1) = 0,$$

which yields

$$x_1 = \frac{3s_1 + 225}{4}.$$

Because $s_1 = 255$, it follows that $x_1 = 247.5$ minimizes $f_1(s_1, x_1)$ over the region $240 \le x_1 \le 255$.

Note that this region ($240 \le x_1 \le 255$) includes $x_1 = 240$, so that $f_1(s_1, 240) > f_1(s_1, 247.5)$. In the next-to-last paragraph, we found that $x_1 = 240$ minimizes $f_1(s_1, x_1)$ over the region $220 \le x_1 \le 240$. Consequently, we now can conclude that $x_1 = 247.5$ also minimizes $f_1(s_1, x_1)$ over the *entire* feasible region $220 \le x_1 \le 255$.

Our final calculation is to find $f_1^*(s_1)$ for $s_1 = 255$ by plugging $x_1 = 247.5$ into the expression for $f_1(255, x_1)$ that holds for $240 \le x_1 \le 255$. Hence,

$$\begin{aligned} f_1^*(255) &= 200(247.5 - 255)^2 + 2{,}000(247.5 - 220) \\ &\quad + \frac{200}{9}\left[2(250 - 247.5)^2 + (265 - 247.5)^2 + 30(742.5 - 575)\right] \\ &= 185{,}000. \end{aligned}$$

These results are summarized as follows:

$n = 1$:

s_1	$f_1^*(s_1)$	x_1^*
255	185,000	247.5

Therefore, by tracing back through the tables for $n = 2$, $n = 3$, and $n = 4$, respectively, and setting $s_n = x_{n-1}^*$ each time, the resulting optimal solution is $x_1^* = 247.5$, $x_2^* = 245$, $x_3^* = 247.5$, $x_4^* = 255$, with a total estimated cost per cycle of \$185,000.

To conclude our illustrations of deterministic dynamic programming, we give one example that requires *more than one* variable to describe the state at each stage.

EXAMPLE 5 Wyndor Glass Company Problem

Consider the following linear programming problem:

Maximize $\quad Z = 3x_1 + 5x_2,$

subject to

$$\begin{aligned} x_1 &\le 4 \\ 2x_2 &\le 12 \\ 3x_1 + 2x_2 &\le 18 \end{aligned}$$

and

$$x_1 \geq 0, \qquad x_2 \geq 0.$$

(You might recognize this as being the model for the Wyndor Glass Co. problem—introduced in Sec. 3.1.) One way of solving small linear (or nonlinear) programming problems like this one is by dynamic programming, which is illustrated below.

Formulation. This problem requires making two interrelated decisions, namely, the level of activity 1, denoted by x_1, and the level of activity 2, denoted by x_2. Therefore, these two activities can be interpreted as the two stages in a dynamic programming formulation. Although they can be taken in either order, let stage n = activity n ($n = 1, 2$). Thus, x_n is the decision variable at stage n.

What are the states? In other words, given that the decision had been made at prior stages (if any), what information is needed about the current state of affairs before the decision can be made at stage n? Reflection might suggest that the required information is the *amount of slack* left in the functional constraints. Interpret the right-hand side of these constraints (4, 12, and 18) as the total available amount of resources 1, 2, and 3, respectively (as described in Sec. 3.1). Then state s_n can be defined as

State s_n = amount of respective resources still available for allocation to remaining activities.

(Note that the definition of the state is analogous to that for distribution of effort problems, including Examples 2 and 3, except that there are now three resources to be allocated instead of just one.) Thus,

$$s_n = (R_1, R_2, R_3),$$

where R_i is the amount of resource i remaining to be allocated ($i = 1, 2, 3$). Therefore,

$$s_1 = (4, 12, 18),$$
$$s_2 = (4 - x_1, 12, 18 - 3x_1).$$

However, when we begin by solving for stage 2, we do not yet know the value of x_1, and so we use $s_2 = (R_1, R_2, R_3)$ at that point.

Therefore, in contrast to the preceding examples, this problem has *three* state variables (i.e., a *state vector* with three components) at each stage rather than one. From a theoretical standpoint, this difference is not particularly serious. It only means that, instead of considering all possible values of the one state variable, we must consider all possible *combinations* of values of the several state variables. However, from the standpoint of computational efficiency, this difference tends to be a very serious complication. Because the number of combinations, in general, can be as large as the *product* of the number of possible values of the respective variables, the number of required calculations tends to "blow up" rapidly when additional state variables are introduced. This phenomenon has been given the apt name of the **curse of dimensionality.**

Each of the three state variables is *continuous.* Therefore, rather than consider each possible combination of values separately, we must use the approach introduced in Example 4 of solving for the required information as a *function* of the state of the system.

Despite these complications, this problem is small enough that it can still be solved without great difficulty. To solve it, we need to introduce the usual dynamic programming notation. Thus,

$$f_2(R_1, R_2, R_3, x_2) = \text{contribution of activity 2 to } Z \text{ if system starts in state } (R_1, R_2, R_3) \text{ at stage 2 and decision is } x_2$$
$$= 5x_2,$$

$$f_1(4, 12, 18, x_1) = \text{contribution of activities 1 and 2 to } Z \text{ if system starts in state (4, 12, 18) at stage 1, immediate decision is } x_1, \text{ and then optimal decision is made at stage 2,}$$
$$= 3x_1 + \max_{\substack{2x_2 \le 12 \\ 2x_2 \le 18 - 3x_1 \\ x_2 \ge 0}} \{5x_2\}.$$

Similarly, for $n = 1, 2$,

$$f_n^*(R_1, R_2, R_3) = \max_{x_n} \quad f_n(R_1, R_2, R_3, x_n),$$

where this maximum is taken over the feasible values of x_n. Consequently, using the relevant portions of the constraints of the problem gives

$$(1) \qquad f_2^*(R_1, R_2, R_3) = \max_{\substack{2x_2 \le R_2 \\ 2x_2 \le R_3 \\ x_2 \ge 0}} \{5x_2\},$$
$$(2) \qquad f_1(4, 12, 18, x_1) = 3x_1 + f_2^*(4 - x_1, 12, 18 - 3x_1),$$
$$(3) \qquad f_1^*(4, 12, 18) = \max_{\substack{x_1 \le 4 \\ 3x_1 \le 18 \\ x_1 \ge 0}} \{3x_1 + f_2^*(4 - x_1, 12, 18 - 3x_1)\}.$$

Equation (1) will be used to solve the stage 2 problem. Equation (2) shows the basic dynamic programming structure for the overall problem, also depicted in Fig. 10.10. Equation (3) gives the *recursive relationship* between f_1^* and f_2^* that will be used to solve the stage 1 problem.

Solution Procedure. *Stage 2:* To solve at the last stage ($n = 2$), Eq. (1) indicates that x_2^* must be the largest value of x_2 that *simultaneously* satisfies $2x_2 \le R_2$, $2x_2 \le R_3$, and $x_2 \ge 0$. Assuming that $R_2 \ge 0$ and $R_3 \ge 0$, so that feasible solutions exist, this largest value is the smaller of $R_2/2$ and $R_3/2$. Thus, the solution is

$n = 2$:

(R_1, R_2, R_3)	$f_2^*(R_1, R_2, R_3)$	x_2^*
$R_2 \ge 0,\ R_3 \ge 0$	$5 \min\left\{\frac{R_2}{2}, \frac{R_3}{2}\right\}$	$\min\left\{\frac{R_2}{2}, \frac{R_3}{2}\right\}$

Stage 1: To solve the two-stage problem ($n = 1$), we plug the solution just obtained for $f_2^*(R_1, R_2, R_3)$ into Eq. (3). For stage 2,

$$(R_1, R_2, R_3) = (4 - x_1, 12, 18 - 3x_1),$$

so that

$$f_2^*(4 - x_1, 12, 18 - 3x_1) = 5 \min\left\{\frac{R_2}{2}, \frac{R_3}{2}\right\} = 5 \min\left\{\frac{12}{2}, \frac{18 - 3x_1}{2}\right\}$$

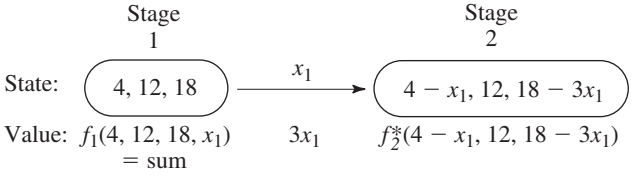


FIGURE 10.10
The basic structure for the Wyndor Glass Co. linear programming problem.

is the specific solution plugged into Eq. (3). After we combine its constraints on x_1, Eq. (3) then becomes

$$f_1^*(4, 12, 18) = \max_{0 \le x_1 \le 4} \left\{ 3x_1 + 5 \min \left\{ \frac{12}{2}, \frac{18 - 3x_1}{2} \right\} \right\}.$$

Over the feasible interval $0 \le x_1 \le 4$, notice that

$$\min \left\{ \frac{12}{2}, \frac{18 - 3x_1}{2} \right\} = \begin{cases} 6 & \text{if } 0 \le x_1 \le 2 \\ 9 - \frac{3}{2} x_1 & \text{if } 2 \le x_1 \le 4, \end{cases}$$

so that

$$3x_1 + 5 \min \left\{ \frac{12}{2}, \frac{18 - 3x_1}{2} \right\} = \begin{cases} 3x_1 + 30 & \text{if } 0 \le x_1 \le 2 \\ 45 - \frac{9}{2} x_1 & \text{if } 2 \le x_1 \le 4. \end{cases}$$

Because both

$$\max_{0 \le x_1 \le 2} \{3x_1 + 30\} \quad \text{and} \quad \max_{2 \le x_1 \le 4} \left\{ 45 - \frac{9}{2} x_1 \right\}$$

achieve their maximum at $x_1 = 2$, it follows that $x_1^* = 2$ and that this maximum is 36, as given in the following table.

$n = 1$:

(R_1, R_2, R_3)	$f_1^*(R_1, R_2, R_3)$	x_1^*
(4, 12, 18)	36	2

Because $x_1^* = 2$ leads to

$$R_1 = 4 - 2 = 2, \qquad R_2 = 12, \qquad R_3 = 18 - 3(2) = 12$$

for stage 2, the $n = 2$ table yields $x_2^* = 6$. Consequently, $x_1^* = 2$, $x_2^* = 6$ is the optimal solution for this problem (as originally found in Sec. 3.1), and the $n = 1$ table shows that the resulting value of Z is 36.

You now have seen a variety of applications of dynamic programming, with more to come in the next section. However, these examples only scratch the surface. For example, Chapter 2 of Selected Reference 4 describes 47 types of problems to which dynamic programming can be applied. (This reference also presents a software tool that can be used to solve all these problem types.) The one common theme that runs through all these applications of dynamic programming is the need to make a series of interrelated decisions and the efficient way dynamic programming provides for finding an optimal combination of decisions.

10.4 PROBABILISTIC DYNAMIC PROGRAMMING

Probabilistic dynamic programming differs from deterministic dynamic programming in that the state at the next stage is *not* completely determined by the state and policy decision at the current stage. Rather, there is a *probability distribution* for what the next state will be. However, this probability distribution still is completely determined by the state

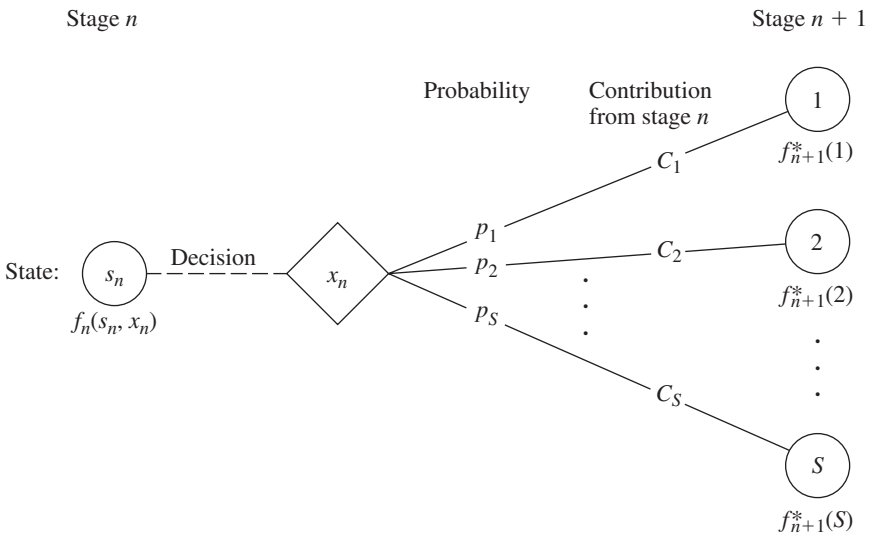


FIGURE 10.11
The basic structure for probabilistic dynamic programming.

and policy decision at the current stage. The resulting basic structure for probabilistic dynamic programming is described diagrammatically in Fig. 10.11.

For the purposes of this diagram, we let S denote the number of possible states at stage $n + 1$ and label these states on the right side as 1, 2, . . . , S. The system goes to state i with probability p_i ($i = 1, 2, \ldots, S$) given state s_n and decision x_n at stage n. If the system goes to state i, C_i is the contribution of stage n to the objective function.

When Fig. 10.11 is expanded to include all the possible states and decisions at all the stages, it is sometimes referred to as a **decision tree.** If the decision tree is not too large, it provides a useful way of summarizing the various possibilities.

Because of the probabilistic structure, the relationship between $f_n(s_n, x_n)$ and the $f^*_{n+1}(s_{n+1})$ necessarily is somewhat more complicated than that for deterministic dynamic programming. The precise form of this relationship will depend upon the form of the overall objective function.

To illustrate, suppose that the objective is to *minimize* the *expected sum* of the contributions from the individual stages. In this case, $f_n(s_n, x_n)$ represents the minimum expected sum from stage n onward, *given* that the state and policy decision at stage n are s_n and x_n, respectively. Consequently,

$$f_n(s_n, x_n) = \sum_{i=1}^{S} p_i[C_i + f^*_{n+1}(i)],$$

with

$$f^*_{n+1}(i) = \min_{x_{n+1}} f_{n+1}(i, x_{n+1}),$$

where this minimization is taken over the *feasible* values of x_{n+1}.

Example 6 has this same form. Example 7 will illustrate another form.

EXAMPLE 6 Determining Reject Allowances

The HIT-AND-MISS MANUFACTURING COMPANY has received an order to supply one item of a particular type. However, the customer has specified such stringent quality requirements that the manufacturer may have to produce more than one item to obtain an

item that is acceptable. The number of *extra* items produced in a production run is called the *reject allowance*. Including a reject allowance is common practice when producing for a custom order, and it seems advisable in this case.

The manufacturer estimates that each item of this type that is produced will be *acceptable* with probability $\frac{1}{2}$ and *defective* (without possibility for rework) with probability $\frac{1}{2}$. Thus, the number of acceptable items produced in a lot of size L will have a *binomial distribution;* i.e., the probability of producing no acceptable items in such a lot is $(\frac{1}{2})^L$.

Marginal production costs for this product are estimated to be \$100 per item (even if defective), and excess items are worthless. In addition, a setup cost of \$300 must be incurred whenever the production process is set up for this product, and a completely new setup at this same cost is required for each subsequent production run if a lengthy inspection procedure reveals that a completed lot has not yielded an acceptable item. The manufacturer has time to make no more than three production runs. If an acceptable item has not been obtained by the end of the third production run, the cost to the manufacturer in lost sales income and penalty costs will be \$1,600.

The objective is to determine the policy regarding the lot size (1 + reject allowance) for the required production run(s) that minimizes total expected cost for the manufacturer.

Formulation. A dynamic programming formulation for this problem is

Stage n = production run n ($n = 1, 2, 3$),
x_n = lot size for stage n,
State s_n = number of acceptable items still needed (1 or 0) at beginning of stage n.

Thus, at stage 1, state $s_1 = 1$. If at least one acceptable item is obtained subsequently, the state changes to $s_n = 0$, after which no additional costs need to be incurred.

Because of the stated objective for the problem,

$f_n(s_n, x_n)$ = total expected cost for stages $n, \ldots, 3$ if system starts in state s_n at stage n, immediate decision is x_n, and optimal decisions are made thereafter,

$$f_n^*(s_n) = \min_{x_n = 0, 1, \ldots} f_n(s_n, x_n),$$

where $f_n^*(0) = 0$. Using \$100 as the unit of money, the contribution to cost from stage n is $[K(x_n) + x_n]$ regardless of the next state, where $K(x_n)$ is a function of x_n such that

$$K(x_n) = \begin{cases} 0, & \text{if } x_n = 0 \\ 3, & \text{if } x_n > 0. \end{cases}$$

Therefore, for $s_n = 1$

$$f_n(1, x_n) = K(x_n) + x_n + \left(\frac{1}{2}\right)^{x_n} f_{n+1}^*(1) + \left[1 - \left(\frac{1}{2}\right)^{x_n}\right] f_{n+1}^*(0)$$

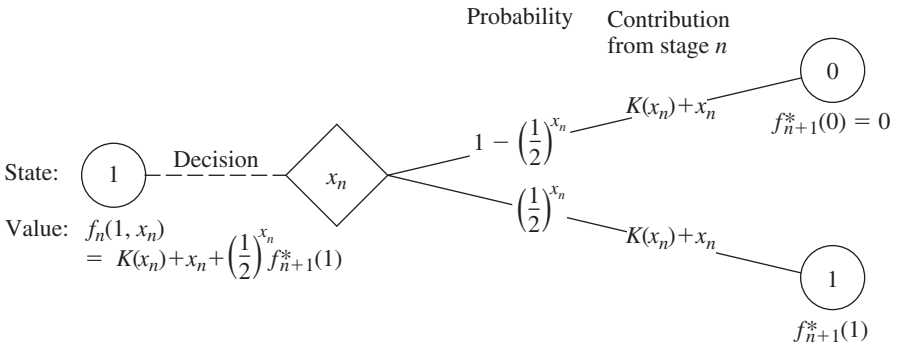
$$= K(x_n) + x_n + \left(\frac{1}{2}\right)^{x_n} f_{n+1}^*(1)$$

[where $f_4^*(1)$ is defined to be 16, the terminal cost if no acceptable items have been obtained]. A summary of these basic relationships is given in Fig. 10.12.

Consequently, the recursive relationship for the dynamic programming calculations is

$$f_n^*(1) = \min_{x_n = 0, 1, \ldots} \left\{ K(x_n) + x_n + \left(\frac{1}{2}\right)^{x_n} f_{n+1}^*(1) \right\}$$

for $n = 1, 2, 3$.



■ **FIGURE 10.12**
The basic structure for the Hit-and-Miss Manufacturing Co. problem.

Solution Procedure. The calculations using this recursive relationship are summarized as follows.

$n = 3$:

s_3 \ x_3	0	1	2	3	4	5	$f^*_3(s_3)$	x^*_3
	$f_3(1, x_3) = K(x_3) + x_3 + 16\left(\frac{1}{2}\right)^{x_3}$							
0	0						0	0
1	16	12	9	8	8	$8\frac{1}{2}$	8	3 or 4

$n = 2$:

s_2 \ x_2	0	1	2	3	4	$f^*_2(s_2)$	x^*_2
	$f_2(1, x_2) = K(x_2) + x_2 + \left(\frac{1}{2}\right)^{x_2} f^*_3(1)$						
0	0					0	0
1	8	8	7	7	$7\frac{1}{2}$	7	2 or 3

$n = 1$:

s_1 \ x_1	0	1	2	3	4	$f^*_1(s_1)$	x^*_1
	$f_1(1, x_1) = K(x_1) + x_1 + \left(\frac{1}{2}\right)^{x_1} f^*_2(1)$						
1	7	$7\frac{1}{2}$	$6\frac{3}{4}$	$6\frac{7}{8}$	$7\frac{7}{16}$	$6\frac{3}{4}$	2

Thus, the optimal policy is to produce two items on the first production run; if none is acceptable, then produce either two or three items on the second production run; if none is acceptable, then produce either three or four items on the third production run. The total expected cost for this policy is $675.

EXAMPLE 7 Winning in Las Vegas

An enterprising young statistician believes that she has developed a system for winning a popular Las Vegas game. Her colleagues do not believe that her system works, so they have made a large bet with her that if she starts with three chips, she will not have at least five chips after three plays of the game. Each play of the game involves betting any desired number of available chips and then either winning or losing this number of chips. The statistician believes that her system will give her a probability of $\frac{2}{3}$ of winning a given play of the game.

Assuming the statistician is correct, we now use dynamic programming to determine her optimal policy regarding how many chips to bet (if any) at each of the three plays of the game. The decision at each play should take into account the results of earlier plays. The objective is to maximize the probability of winning her bet with her colleagues.

Formulation. The dynamic programming formulation for this problem is

$$\begin{aligned}\text{Stage } n &= n\text{th play of game } (n = 1, 2, 3),\\ x_n &= \text{number of chips to bet at stage } n,\\ \text{State } s_n &= \text{number of chips in hand to begin stage } n.\end{aligned}$$

This definition of the state is chosen because it provides the needed information about the current situation for making an optimal decision on how many chips to bet next.

Because the objective is to maximize the probability that the statistician will win her bet, the objective function to be maximized at each stage must be the probability of finishing the three plays with at least five chips. (Note that the value of ending with more than five chips is just the same as ending with exactly five, since the bet is won either way.) Therefore,

$$\begin{aligned}f_n(s_n, x_n) &= \text{probability of finishing three plays with at least five chips, given that the statistician starts stage } n \text{ in state } s_n, \text{ makes immediate decision } x_n, \text{ and makes optimal decisions thereafter,}\\ f_n^*(s_n) &= \max_{x_n = 0, 1, \ldots, s_n} f_n(s_n, x_n).\end{aligned}$$

The expression for $f_n(s_n, x_n)$ must reflect the fact that it may still be possible to accumulate five chips eventually even if the statistician should lose the next play. If she loses, the state at the next stage will be $s_n - x_n$, and the probability of finishing with at least five chips will then be $f_{n+1}^*(s_n - x_n)$. If she wins the next play instead, the state will become $s_n + x_n$, and the corresponding probability will be $f_{n+1}^*(s_n + x_n)$. Because the assumed probability of winning a given play is $\frac{2}{3}$, it now follows that

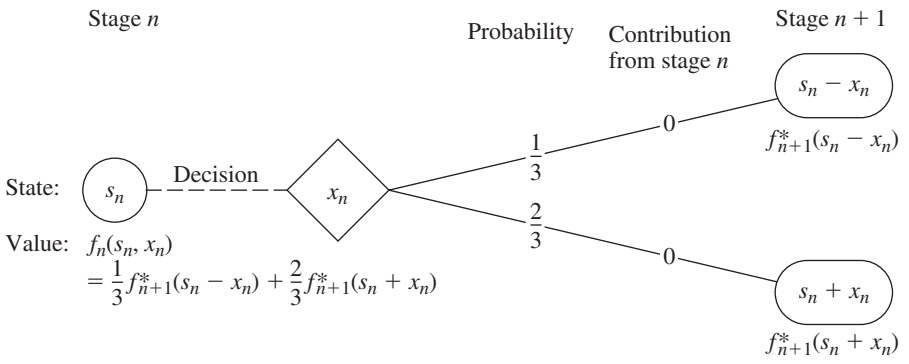
$$f_n(s_n, x_n) = \frac{1}{3} f_{n+1}^*(s_n - x_n) + \frac{2}{3} f_{n+1}^*(s_n + x_n)$$

[where $f_4^*(s_4)$ is defined to be 0 for $s_4 < 5$ and 1 for $s_4 \geq 5$]. Thus, there is no direct contribution to the objective function from stage n other than the effect of then being in the next state. These basic relationships are summarized in Fig. 10.13.

Therefore, the recursive relationship for this problem is

$$f_n^*(s_n) = \max_{x_n = 0, 1, \ldots, s_n} \left\{ \frac{1}{3} f_{n+1}^*(s_n - x_n) + \frac{2}{3} f_{n+1}^*(s_n + x_n) \right\},$$

for $n = 1, 2, 3$, with $f_4^*(s_4)$ as just defined.



■ **FIGURE 10.13**
The basic structure for the Las Vegas problem.

Solution Procedure. This recursive relationship leads to the following computational results.

***n* = 3:**

s_3	$f_3^*(s_3)$	x_3^*
0	0	—
1	0	—
2	0	—
3	$\frac{2}{3}$	2 (or more)
4	$\frac{2}{3}$	1 (or more)
≥5	1	0 (or ≤ $s_3 - 5$)

***n* = 2:**

s_2 \ x_2	$f_2(s_2, x_2) = \frac{1}{3}f_3^*(s_2 - x_2) + \frac{2}{3}f_3^*(s_2 + x_2)$					$f_2^*(s_2)$	x_2^*
	0	**1**	**2**	**3**	**4**		
0	0					0	—
1	0	0				0	—
2	0	$\frac{4}{9}$	$\frac{4}{9}$			$\frac{4}{9}$	1 or 2
3	$\frac{2}{3}$	$\frac{4}{9}$	$\frac{2}{3}$	$\frac{2}{3}$		$\frac{2}{3}$	0, 2, or 3
4	$\frac{2}{3}$	$\frac{8}{9}$	$\frac{2}{3}$	$\frac{2}{3}$	$\frac{2}{3}$	$\frac{8}{9}$	1
≥5	1					1	0 (or ≤ $s_2 - 5$)

***n* = 1:**

s_1 \ x_1	$f_1(s_1, x_1) = \frac{1}{3}f_2^*(s_1 - x_1) + \frac{2}{3}f_2^*(s_1 + x_1)$				$f_1^*(s_1)$	x_1^*
	0	**1**	**2**	**3**		
3	$\frac{2}{3}$	$\frac{20}{27}$	$\frac{2}{3}$	$\frac{2}{3}$	$\frac{20}{27}$	1

Therefore, the optimal policy is

$$x_1^* = 1 \begin{cases} \text{if win,} & x_2^* = 1 \begin{cases} \text{if win,} & x_3^* = 0 \\ \text{if lose,} & x_3^* = 2 \text{ or } 3. \end{cases} \\ \text{if lose,} & x_2^* = 1 \text{ or } 2 \begin{cases} \text{if win,} & x_3^* = \begin{cases} 2 \text{ or } 3 & (\text{for } x_2^* = 1) \\ 1, 2, 3, \text{ or } 4 & (\text{for } x_2^* = 2) \end{cases} \\ \text{if lose,} & \text{bet is lost} \end{cases} \end{cases}$$

This policy gives the statistician a probability of $\frac{20}{27}$ of winning her bet with her colleagues.

10.5 CONCLUSIONS

Dynamic programming is a very useful technique for making a *sequence of interrelated decisions.* It requires formulating an appropriate *recursive relationship* for each individual problem. However, it provides a great computational savings over using exhaustive enumeration to find the best combination of decisions, especially for large problems. For example, if a problem has 10 stages with 10 states and 10 possible decisions at each stage, then exhaustive enumeration must consider up to 10 billion combinations, whereas dynamic programming need make no more than a thousand calculations (10 for each state at each stage).

This chapter has considered only dynamic programming with a *finite* number of stages. Chapter 19 is devoted to a general kind of model for probabilistic dynamic programming where the stages continue to recur indefinitely, namely, Markov decision processes.

SELECTED REFERENCES

1. Bertsekas, D. P.: *Dynamic Programming: Deterministic and Stochastic Models,* Prentice-Hall, Englewood Cliffs, NJ, 1987.
2. Denardo, E. V.: *Dynamic Programming Theory and Applications,* Prentice-Hall, Englewood Cliffs, NJ, 1982.
3. Howard, R. A.: "Dynamic Programming," *Management Science,* **12:** 317–345, 1966.
4. Lew, A., and H. Mauch: *Dynamic Programming: A Computational Tool*, Springer, New York, 2007.
5. Smith, D. K.: *Dynamic Programming: A Practical Introduction,* Ellis Horwood, London, 1991.
6. Sniedovich, M.: *Dynamic Programming,* Marcel Dekker, New York, 1991.

LEARNING AIDS FOR THIS CHAPTER ON OUR WEBSITE (www.mhhe.com/hillier)

Worked Examples:

Examples for Chapter 10

"Ch. 10—Dynamic Programming" LINGO File

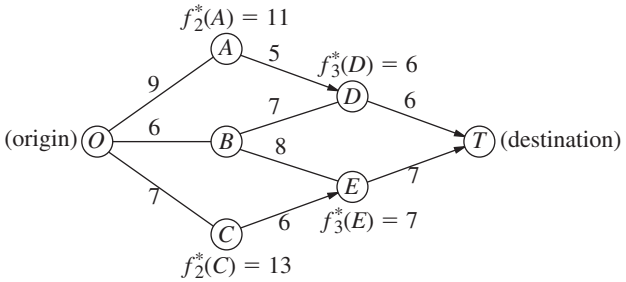
Glossary for Chapter 10

See Appendix 1 for documentation of the software.

PROBLEMS

An asterisk on the problem number indicates that at least a partial answer is given in the back of the book.

10.2-1. Consider the following network, where each number along a link represents the actual distance between the pair of nodes connected by that link. The objective is to find the shortest path from the origin to the destination.



(a) What are the stages and states for the dynamic programming formulation of this problem?

(b) Use dynamic programming to solve this problem. However, instead of using the usual tables, show your work graphically (similar to Fig. 10.2). In particular, start with the given network, where the answers already are given for $f_n^*(s_n)$ for four of the nodes; then solve for and fill in $f_2^*(B)$ and $f_1^*(O)$. Draw an arrowhead that shows the optimal link to traverse out of each of the latter two nodes. Finally, identify the optimal path by following the arrows from node O onward to node T.

(c) Use dynamic programming to solve this problem by manually constructing the usual tables for $n = 3$, $n = 2$, and $n = 1$.

(d) Use the shortest-path algorithm presented in Sec. 9.3 to solve this problem. Compare and contrast this approach with the one in parts (*b*) and (*c*).

10.2-2. The sales manager for a publisher of college textbooks has six traveling salespeople to assign to three different regions of the country. She has decided that each region should be assigned at least one salesperson and that each individual salesperson should be restricted to one of the regions, but now she wants to determine how many salespeople should be assigned to the respective regions in order to maximize sales.

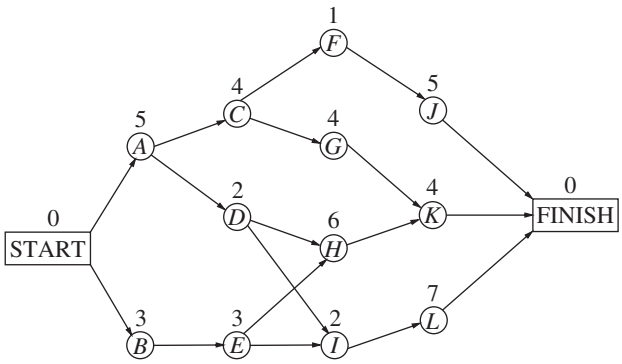
The next table gives the estimated increase in sales (in appropriate units) in each region if it were allocated various numbers of salespeople:

	Region		
Salespersons	**1**	**2**	**3**
1	40	24	32
2	54	47	46
3	78	63	70
4	99	78	84

(a) Use dynamic programming to solve this problem. Instead of using the usual tables, show your work graphically by constructing and filling in a network such as the one shown for Prob. 10.2-1. Proceed as in Prob. 10.2-1*b* by solving for $f_n^*(s_n)$ for each node (except the terminal node) and writing its value by the node. Draw an arrowhead to show the optimal link (or links in case of a tie) to take out of each node. Finally, identify the resulting optimal path (or paths) through the network and the corresponding optimal solution (or solutions).

(b) Use dynamic programming to solve this problem by constructing the usual tables for $n = 3$, $n = 2$, and $n = 1$.

10.2-3. Consider the following project network (as described in Sec. 9.8), where the number over each node is the time required for the corresponding activity. Consider the problem of finding the *longest path* (the largest total time) through this network from start to finish, since the longest path is the critical path.



(a) What are the stages and states for the dynamic programming formulation of this problem?

(b) Use dynamic programming to solve this problem. However, instead of using the usual tables, show your work graphically. In particular, fill in the values of the various $f_n^*(s_n)$ under the corresponding nodes, and show the resulting optimal arc to traverse out of each node by drawing an arrowhead near the beginning of the arc. Then identify the optimal path (the longest path) by following these arrowheads from the Start node to the Finish node. If there is more than one optimal path, identify them all.

(c) Use dynamic programming to solve this problem by constructing the usual tables for $n = 4$, $n = 3$, $n = 2$, and $n = 1$.

10.2-4. Consider the following statements about solving dynamic programming problems. Label each statement as true or false, and then justify your answer by referring to specific statements in the chapter.

(a) The solution procedure uses a recursive relationship that enables solving for the optimal policy for stage $(n + 1)$ given the optimal policy for stage n.
(b) After completing the solution procedure, if a nonoptimal decision is made by mistake at some stage, the solution procedure will need to be reapplied to determine the new optimal decisions (given this nonoptimal decision) at the subsequent stages.
(c) Once an optimal policy has been found for the overall problem, the information needed to specify the optimal decision at a particular stage is the state at that stage and the decisions made at preceding stages.

10.3-1. Read the referenced article that fully describes the OR study summarized in the application vignette presented in Sec. 10.3. Briefly describe how dynamic programming was applied in this study. Then list the various financial and nonfinancial benefits that resulted from this study.

10.3-2.* The owner of a chain of three grocery stores has purchased five crates of fresh strawberries. The estimated probability distribution of potential sales of the strawberries before spoilage differs among the three stores. Therefore, the owner wants to know how to allocate five crates to the three stores to maximize expected profit.

For administrative reasons, the owner does not wish to split crates between stores. However, he is willing to distribute no crates to any of his stores.

The following table gives the estimated expected profit at each store when it is allocated various numbers of crates:

	Store		
Crates	**1**	**2**	**3**
0	0	0	0
1	5	6	4
2	9	11	9
3	14	15	13
4	17	19	18
5	21	22	20

Use dynamic programming to determine how many of the five crates should be assigned to each of the three stores to maximize the total expected profit.

10.3-3. A college student has 7 days remaining before final examinations begin in her four courses, and she wants to allocate this study time as effectively as possible. She needs at least 1 day on each course, and she likes to concentrate on just one course each day, so she wants to allocate 1, 2, 3, or 4 days to each course. Having recently taken an OR course, she decides to use dynamic programming to make these allocations to maximize the total grade points to be obtained from the four courses. She estimates that the alternative allocations for each course would yield the number of grade points shown in the following table:

	Estimated Grade Points			
	Course			
Study Days	**1**	**2**	**3**	**4**
1	1	5	4	4
2	3	6	6	4
3	6	8	7	5
4	8	8	9	8

Solve this problem by dynamic programming.

10.3-4. A political campaign is entering its final stage, and polls indicate a very close election. One of the candidates has enough funds left to purchase TV time for a total of five prime-time commercials on TV stations located in four different areas. Based on polling information, an estimate has been made of the number of additional votes that can be won in the different broadcasting areas depending upon the number of commercials run. These estimates are given in the following table in thousands of votes:

	Area			
Commercials	**1**	**2**	**3**	**4**
0	0	0	0	0
1	4	6	5	3
2	7	8	9	7
3	9	10	11	12
4	12	11	10	14
5	15	12	9	16

Use dynamic programming to determine how the five commercials should be distributed among the four areas in order to maximize the estimated number of votes won.

10.3-5. A county chairwoman of a certain political party is making plans for an upcoming presidential election. She has received the services of six volunteer workers for precinct work, and she wants to assign them to four precincts in such a way as to maximize their effectiveness. She feels that it would be inefficient to assign a worker to more than one precinct, but she is willing to assign no workers to any one of the precincts if they can accomplish more in other precincts.

The following table gives the estimated increase in the number of votes for the party's candidate in each precinct if it were allocated various numbers of workers:

	Precinct			
Workers	**1**	**2**	**3**	**4**
0	0	0	0	0
1	4	7	5	6
2	9	11	10	11
3	15	16	15	14
4	18	18	18	16
5	22	20	21	17
6	24	21	22	18

This problem has several optimal solutions for how many of the six workers should be assigned to each of the four precincts to maximize the total estimated increase in the plurality of the party's candidate. Use dynamic programming to find all of them so the chairwoman can make the final selection based on other factors.

10.3-6. Use dynamic programming to solve the Northern Airplane Co. production scheduling problem presented in Sec. 8.1 (see Table 8.7). Assume that production quantities must be integer multiples of 5.

10.3-7.* A company will soon be introducing a new product into a very competitive market and is currently planning its marketing strategy. The decision has been made to introduce the product in three phases. Phase 1 will feature making a special introductory offer of the product to the public at a greatly reduced price to attract first-time buyers. Phase 2 will involve an intensive advertising campaign to persuade these first-time buyers to continue purchasing the product at a regular price. It is known that another company will be introducing a new competitive product at about the time that phase 2 will end. Therefore, phase 3 will involve a follow-up advertising and promotion campaign to try to keep the regular purchasers from switching to the competitive product.

A total of $4 million has been budgeted for this marketing campaign. The problem now is to determine how to allocate this money most effectively to the three phases. Let m denote the initial share of the market (expressed as a percentage) attained in phase 1, f_2 the fraction of this market share that is retained in phase 2, and f_3 the fraction of the remaining market share that is retained in phase 3. Use dynamic programming to determine how to allocate the $4 million to maximize the final share of the market for the new product, i.e., to maximize mf_2f_3.

(a) Assume that the money must be spent in integer multiples of $1 million in each phase, where the minimum permissible multiple is 1 for phase 1 and 0 for phases 2 and 3. The following table gives the estimated effect of expenditures in each phase:

	Effect on Market Share		
Millions of Dollars Expended	m	f_2	f_3
0	—	0.2	0.3
1	20	0.4	0.5
2	30	0.5	0.6
3	40	0.6	0.7
4	50	—	—

(b) Now assume that *any* amount within the total budget can be spent in each phase, where the estimated effect of spending an amount x_i (in units of *millions* of dollars) in phase i ($i = 1, 2, 3$) is

$m = 10x_1 - x_1^2$
$f_2 = 0.40 + 0.10x_2$
$f_3 = 0.60 + 0.07x_3$.

[*Hint:* After solving for the $f_2^*(s)$ and $f_3^*(s)$ functions analytically, solve for x_1^* graphically.]

10.3-8. Consider an electronic system consisting of four components, each of which must work for the system to function. The reliability of the system can be improved by installing several parallel units in one or more of the components. The following table gives the probability that the respective components (labeled as Comp. 1, 2, 3, and 4) will function if they consist of one, two, or three parallel units:

	Probability of Functioning			
Parallel Units	**Comp. 1**	**Comp. 2**	**Comp. 3**	**Comp. 4**
1	0.5	0.6	0.7	0.5
2	0.6	0.7	0.8	0.7
3	0.8	0.8	0.9	0.9

The probability that the system will function is the product of the probabilities that the respective components will function.

The cost (in hundreds of dollars) of installing one, two, or three parallel units in the respective components (labeled as Comp. 1, 2, 3, and 4) is given by the following table:

	Cost			
Parallel Units	**Comp. 1**	**Comp. 2**	**Comp. 3**	**Comp. 4**
1	1	2	1	2
2	2	4	3	3
3	3	5	4	4

Because of budget limitations, a maximum of \$1,000 can be expended.

Use dynamic programming to determine how many parallel units should be installed in each of the four components to maximize the probability that the system will function.

10.3-9. Consider the following integer nonlinear programming problem.

$$\text{Maximize} \quad Z = 3x_1^2 - x_1^3 + 5x_2^2 - x_2^3,$$

subject to

$$x_1 + 2x_2 \leq 4$$

and

$$x_1 \geq 0, \quad x_2 \geq 0$$
x_1, x_2 are integers.

Use dynamic programming to solve this problem.

10.3-10. Consider the following integer nonlinear programming problem.

$$\text{Maximize} \quad Z = 32x_1 - 2x_1^2 + 30x_2 + 20x_3,$$

subject to

$$3x_1 + 7x_2 + 5x_3 \leq 20$$

and

x_1, x_2, x_3 are nonnegative integers.

Use dynamic programming to solve this problem.

10.3-11.* Consider the following nonlinear programming problem.

$$\text{Maximize} \quad Z = 36x_1 + 9x_1^2 - 6x_1^3 + 36x_2 - 3x_2^3,$$

subject to

$$x_1 + x_2 \leq 3$$

and

$$x_1 \geq 0, \quad x_2 \geq 0.$$

Use dynamic programming to solve this problem.

10.3-12. Re-solve the Local Job Shop employment scheduling problem (Example 4) when the total cost of changing the level of employment from one season to the next is changed to \$100 times the square of the difference in employment levels.

10.3-13. Consider the following nonlinear programming problem.

$$\text{Maximize} \quad Z = 2x_1^2 + 2x_2 + 4x_3 - x_3^2$$

subject to

$$2x_1 + x_2 + x_3 \leq 4$$

and

$$x_1 \geq 0, \quad x_2 \geq 0, \quad x_3 \geq 0.$$

Use dynamic programming to solve this problem.

10.3-14. Consider the following nonlinear programming problem.

$$\text{Minimize} \quad Z = x_1^4 + 2x_2^2$$

subject to

$$x_1^2 + x_2^2 \geq 2.$$

(There are no nonnegativity constraints.) Use dynamic programming to solve this problem.

10.3-15. Consider the following nonlinear programming problem.

$$\text{Maximize} \quad Z = x_1^3 + 4x_2^2 + 16x_3,$$

subject to

$$x_1x_2x_3 = 4$$

and

$$x_1 \geq 1, \quad x_2 \geq 1, \quad x_3 \geq 1.$$

(a) Solve by dynamic programming when, in addition to the given constraints, all three variables also are required to be integer.
(b) Use dynamic programming to solve the problem as given (continuous variables).

10.3-16. Consider the following nonlinear programming problem.

$$\text{Maximize} \quad Z = x_1(1 - x_2)x_3,$$

subject to

$$x_1 - x_2 + x_3 \leq 1$$

and

$$x_1 \geq 0, \quad x_2 \geq 0, \quad x_3 \geq 0.$$

Use dynamic programming to solve this problem.

10.3-17. Consider the following linear programming problem.

$$\text{Maximize} \quad Z = 15x_1 + 10x_2,$$

subject to

$$\begin{aligned} x_1 + 2x_2 &\leq 6 \\ 3x_1 + x_2 &\leq 8 \end{aligned}$$

and

$$x_1 \geq 0, \quad x_2 \geq 0.$$

Use dynamic programming to solve this problem.

10.3-18. Consider the following "fixed-charge" problem.

$$\text{Maximize} \quad Z = 3x_1 + 7x_2 + 6f(x_3),$$

subject to

$$x_1 + 3x_2 + 2x_3 \leq 6$$
$$x_1 + x_2 \leq 5$$

and

$$x_1 \geq 0, \quad x_2 \geq 0, \quad x_3 \geq 0,$$

where

$$f(x_3) = \begin{cases} 0 & \text{if } x_3 = 0 \\ -1 + x_3 & \text{if } x_3 > 0. \end{cases}$$

Use dynamic programming to solve this problem.

10.4-1. A backgammon player will be playing three consecutive matches with friends tonight. For each match, he will have the opportunity to place an even bet that he will win; the amount bet can be *any* quantity of his choice between zero and the amount of money he still has left after the bets on the preceding matches. For each match, the probability is $\frac{1}{2}$ that he will win the match and thus win the amount bet, whereas the probability is $\frac{1}{2}$ that he will lose the match and thus lose the amount bet. He will begin with \$75, and his goal is to have \$100 at the end. (Because these are friendly matches, he does not want to end up with more than \$100.) Therefore, he wants to find the optimal betting policy (including all ties) that maximizes the probability that he will have exactly \$100 after the three matches.

Use dynamic programming to solve this problem.

10.4-2. Imagine that you have \$10,000 to invest and that you will have an opportunity to invest that amount in either of two investments (*A* or *B*) at the beginning of each of the next 3 years. Both investments have uncertain returns. For investment *A* you will either lose your money entirely or (with higher probability) get back \$20,000 (a profit of \$10,000) at the end of the year. For investment *B* you will get back either just your \$10,000 or (with low probability) \$20,000 at the end of the year. The probabilities for these events are as follows:

Investment	Amount Returned (\$)	Probability
A	0	0.25
	20,000	0.75
B	10,000	0.9
	20,000	0.1

You are allowed to make only (at most) *one* investment each year, and you can invest only \$10,000 each time. (Any additional money accumulated is left idle.)

(a) Use dynamic programming to find the investment policy that maximizes the expected amount of money you will have after 3 years.

(b) Use dynamic programming to find the investment policy that maximizes the probability that you will have at least \$20,000 after 3 years.

10.4-3.* Suppose that the situation for the Hit-and-Miss Manufacturing Co. problem (Example 6) has changed somewhat. After a more careful analysis, you now estimate that each item produced will be acceptable with probability $\frac{2}{3}$, rather than $\frac{1}{2}$, so that the probability of producing *zero* acceptable items in a lot of size L is $(\frac{1}{3})^L$. Furthermore, there now is only enough time available to make two production runs. Use dynamic programming to determine the new optimal policy for this problem.

10.4-4. Reconsider Example 7. Suppose that the bet is changed as follows: "Starting with two chips, she will not have at least five chips after five plays of the game." By referring to the previous computational results, make additional calculations to determine the new optimal policy for the enterprising young statistician.

10.4-5. The Profit & Gambit Co. has a major product that has been losing money recently because of declining sales. In fact, during the current quarter of the year, sales will be 4 million units below the break-even point. Because the marginal revenue for each unit sold exceeds the marginal cost by \$5, this amounts to a loss of \$20 million for the quarter. Therefore, management must take action quickly to rectify this situation. Two alternative courses of action are being considered. One is to abandon the product immediately, incurring a cost of \$20 million for shutting down. The other alternative is to undertake an intensive advertising campaign to increase sales and then abandon the product (at the cost of \$20 million) only if the campaign is not sufficiently successful. Tentative plans for this advertising campaign have been developed and analyzed. It would extend over the next three quarters (subject to early cancellation), and the cost would be \$30 million in each of the three quarters. It is estimated that the increase in sales would be approximately 3 million units in the first quarter, another 2 million units in the second quarter, and another 1 million units in the third quarter. However, because of a number of unpredictable market variables, there is considerable uncertainty as to what impact the advertising actually would have; and careful analysis indicates that the estimates for each quarter could turn out to be off by as much as 2 million units in either direction. (To quantify this uncertainty, assume that the additional increases in sales

in the three quarters are independent random variables having a uniform distribution with a range from 1 to 5 million, from 0 to 4 million, and from −1 to 3 million, respectively.) If the actual increases are too small, the advertising campaign can be discontinued and the product abandoned at the end of either of the next two quarters.

If the intensive advertising campaign were initiated and continued to its completion, it is estimated that the sales for some time thereafter would continue to be at about the same level as in the third (last) quarter of the campaign. Therefore, if the sales in that quarter still were below the break-even point, the product would be abandoned. Otherwise, it is estimated that the expected discounted profit thereafter would be $40 for each unit sold over the break-even point in the third quarter.

Use dynamic programming to determine the optimal policy maximizing the expected profit.