Dynamic Programming

11

Dynamic programming is an optimization approach that transforms a complex problem into a sequence of simpler problems; its essential characteristic is the multistage nature of the optimization procedure. More so than the optimization techniques described previously, dynamic programming provides a general framework for analyzing many problem types. Within this framework a variety of optimization techniques can be employed to solve particular aspects of a more general formulation. Usually creativity is required before we can recognize that a particular problem can be cast effectively as a dynamic program; and often subtle insights are necessary to restructure the formulation so that it can be solved effectively.

We begin by providing a general insight into the dynamic programming approach by treating a simple example in some detail. We then give a formal characterization of dynamic programming under certainty, followed by an in-depth example dealing with optimal capacity expansion. Other topics covered in the chapter include the discounting of future returns, the relationship between dynamic-programming problems and shortest paths in networks, an example of a continuous-state-space problem, and an introduction to dynamic programming under uncertainty.

11.1 AN ELEMENTARY EXAMPLE

In order to introduce the dynamic-programming approach to solving multistage problems, in this section we analyze a simple example. Figure 11.1 represents a street map connecting homes and downtown parking lots for a group of commuters in a model city. The arcs correspond to streets and the nodes correspond to intersections. The network has been designed in a diamond pattern so that every commuter must traverse five streets in driving from home to downtown. The design characteristics and traffic pattern are such that the total time spent by any commuter between intersections is independent of the route taken. However, substantial delays, are experienced by the commuters in the intersections. The lengths of these delays in minutes, are indicated by the numbers within the nodes. We would like to minimize the total delay any commuter can incur in the intersections while driving from his home to downtown. Figure 11.2 provides a compact tabular representation for the problem that is convenient for discussing its solution by dynamic programming. In this figure, boxes correspond to intersections in the network. In going from home to downtown, any commuter must move from left to right through this diagram, moving at each stage only to an adjacent box in the next column to the right. We will refer to the ‘‘stages to go," meaning the number of intersections left to traverse, not counting the intersection that the commuter is currently in.

The most naive approach to solving the problem would be to enumerate all 150 paths through the diagram, selecting the path that gives the smallest delay. Dynamic programming reduces the number of computations by moving systematically from one side to the other, building the best solution as it goes.

Suppose that we move backward through the diagram from right to left. If we are in any intersection (box) with no further intersections to go, we have no decision to make and simply incur the delay corresponding to that intersection. The last column in Fig. 11.2 summarizes the delays with no (zero) intersections to go.

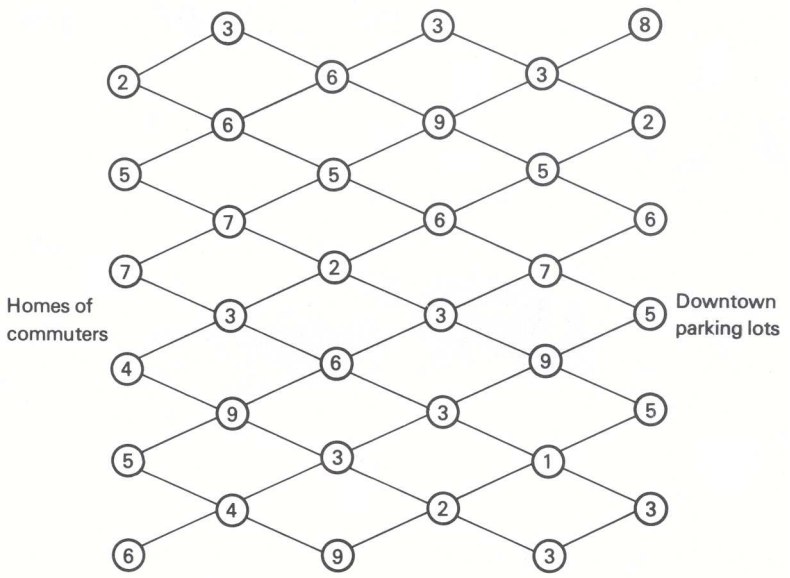


Figure 11.1 Street map with intersection delays.

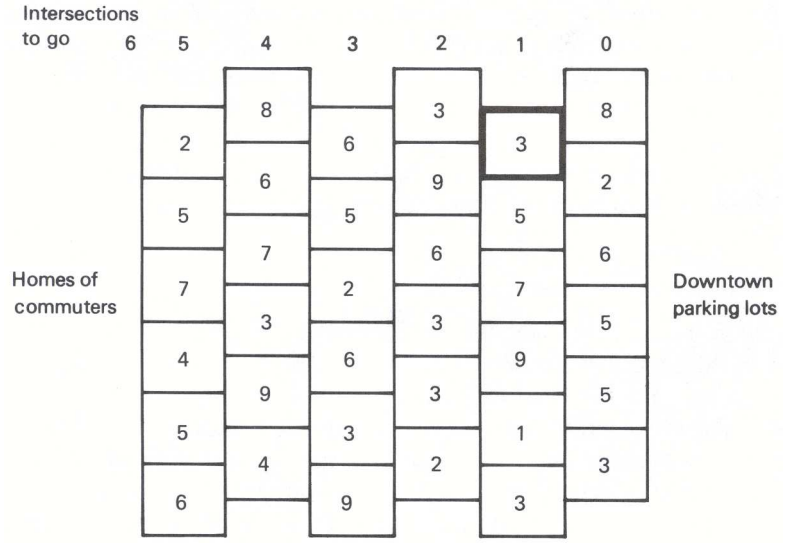


Figure 11.2 Compact representation of the network.

Our first decision (from right to left) occurs with one stage, or intersection, left to go. If for example, we are in the intersection corresponding to the highlighted box in Fig. 11.2, we incur a delay of three minutes in this intersection and a delay of either *eight* or *two* minutes in the last intersection, depending upon whether we move up or down. Therefore, the smallest possible delay, or optimal solution, in this intersection is $3+2=5$ minutes. Similarly, we can consider each intersection (box) in this column in turn and compute the smallest total delay as a result of being in each intersection. The solution is given by the bold-faced numbers in Fig. 11.3. The arrows indicate the optimal decision, up or down, in any intersection with one stage, or one intersection, to go.

Note that the numbers in bold-faced type in Fig. 11.3 completely summarize, for decision-making purposes, the total delays over the last two columns. Although the original numbers in the last two columns have been used to determine the bold-faced numbers, whenever we are making decisions to the left of these columns we need only know the bold-faced numbers. In an intersection, say the topmost with one stage to go, we know that our (optimal) remaining delay, including the delay in this intersection, is five minutes. The bold-faced numbers summarize all delays from this point on. For decision-making to the left of the bold-faced numbers, the last column can be ignored.

With this in mind, let us back up one more column, or stage, and compute the optimal solution in each intersection with two intersections to go. For example, in the bottom-most intersection, which is highlighted in Fig. 11.3, we incur a delay of two minutes in the intersection, plus *four* or *six* additional minutes, depending upon whether we move up or down. To minimize delay, we move *up* and incur a total delay in this intersection and *all remaining intersections* of $2+4=6$ minutes. The remaining computations in this column are summarized in Fig. 11.4, where the bold-faced numbers reflect the optimal total delays in each intersection with two stages, or two intersections, to go.

Once we have computed the optimal delays in each intersection with two stages to go, we can again move back one column and determine the optimal delays and the optimal decisions with three intersections to go. In the same way, we can continue to move back one stage at a time, and compute the optimal delays and decisions with four and five intersections to go, respectively. Figure 11.5 summarizes these calculations.

Figure 11.5(c) shows the optimal solution to the problem. The least possible delay through the network is 18 minutes. To follow the least-cost route, a commuter has to start at the second intersection from the bottom. According to the optimal decisions, or arrows, in the diagram, we see that he should next move down to the bottom-most intersection in column 4. His following decisions should be up, down, up, down, arriving finally at the bottom-most intersection in the last column.

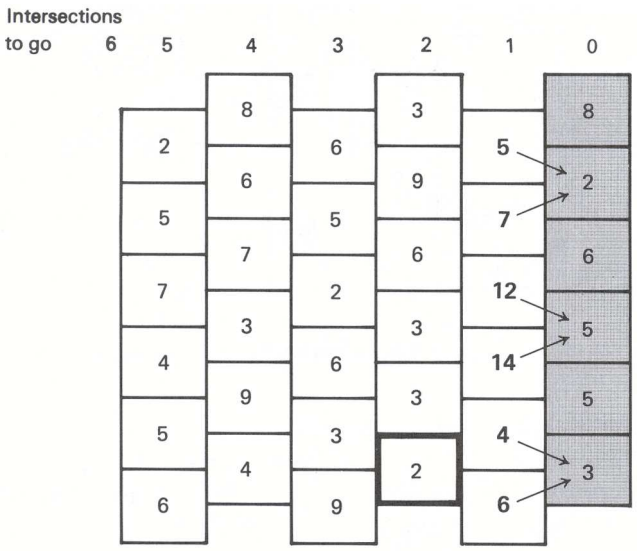


Figure 11.3 Decisions and delays with one intersection to go.

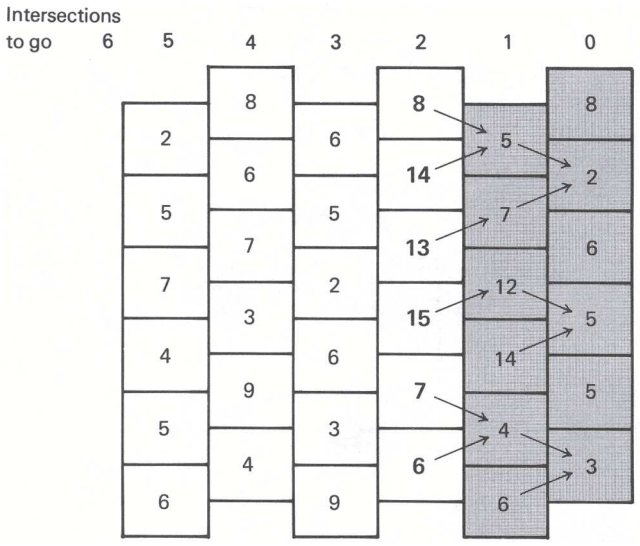


Figure 11.4 Decisions and delays with two intersections to go.

However, the commuters are probably not free to arbitrarily choose the intersection they wish to start from. We can assume that their homes are adjacent to only one of the leftmost intersections, and therefore each commuter's starting point is fixed. This assumption does not cause any difficulty since we have, in fact, determined the routes of minimum delay from the downtown parking lots to *all* the commuter's homes. Note that this assumes that commuters do not care in which downtown lot they park. Instead of solving the minimum-delay problem for only a particular commuter, we have *embedded* the problem of the particular commuter in the more general problem of finding the minimum-delay paths from all homes to the group of downtown parking lots. For example, Fig. 11.5 also indicates that the commuter starting at the topmost intersection incurs a delay of 22 minutes if he follows his optimal policy of down, up, up, down, and then down. He presumably parks in a lot close to the second intersection from the top in the last column. Finally, note that three of the intersections in the last column are not entered by any commuter. The analysis has determined the minimum-delay paths from each of the commuter's homes to the group of downtown parking lots, not to each particular parking lot.

Using dynamic programming, we have solved this minimum-delay problem sequentially by keeping track of how many intersections, or stages, there were to go. In dynamic-programming terminology, each point where decisions are made is usually called a *stage* of the decision-making process. At any stage, we need only know which intersection we are in to be able to make subsequent decisions. Our subsequent decisions do not depend upon how we arrived at the particular intersection. Information that summarizes the knowledge required about the problem in order to make the current decisions, such as the intersection we are in at a particular stage, is called a *state* of the decision-making process.

In terms of these notions, our solution to the minimum-delay problem involved the following intuitive idea, usually referred to as the *principle of optimality*.

> *Any optimal policy has the property that, whatever the current state and decision, the remaining decisions must constitute an optimal policy with regard to the state resulting from the current decision.*

To make this principle more concrete, we can define the *optimal-value function* in the context of the minimum-delay problem.

$v_n(s_n) =$ Optimal value (minimum delay) over the current and subsequent stages (intersections), given that we are in state s_n (in a particular intersection) with n stages (intersections) to go.

The optimal-value function at each stage in the decision-making process is given by the appropriate column

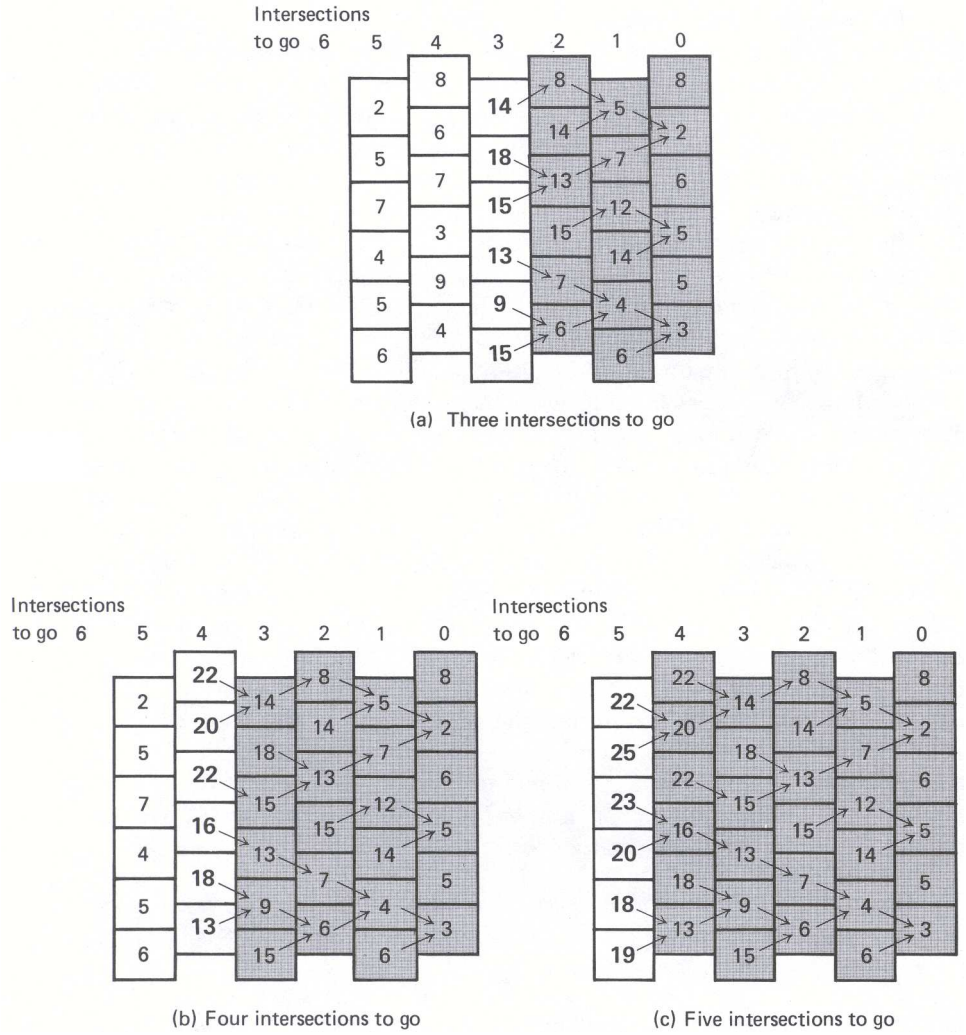


Figure 11.5 Charts of optimal delays and decisions.

of Fig. 11.5(c). We can write down a *recursive* relationship for computing the optimal-value function by recognizing that, at each stage, the decision in a particular state is determined simply by choosing the minimum total delay. If we number the states at each stage as $s_n = 1$ (bottom intersection) up to $s_n = 6$ (top intersection), then

$$v_n(s_n) = \text{Min}\{t_n(s_n) + v_{n-1}(s_{n-1})\}, \tag{1}$$

subject to:

$$s_{n-1} = \begin{cases} s_n + 1 & \text{if we choose up and } n \text{ even,} \\ s_n - 1 & \text{if we choose down and } n \text{ odd,} \\ s_n & \text{otherwise,} \end{cases}$$

where $t_n(s_n)$ is the delay time in intersection s_n at stage n.

The columns of Fig. 11.5(c) are then determined by starting at the right with

$$v_0(s_0) = t_0(s_0) \qquad (s_0 = 1, 2, \ldots, 6), \tag{2}$$

and successively applying Eq. (1). Corresponding to this optimal-value function is an *optimal-decision function*, which is simply a list giving the optimal decision for each state at every stage. For this example, the optimal decisions are given by the arrows leaving each box in every column of Fig. 11.5(c).

The method of computation illustrated above is called *backward induction*, since it starts at the right and moves back one stage at a time. Its analog, *forward induction*, which is also possible, starts at the left and moves forward one stage at a time. The spirit of the calculations is identical but the interpretation is somewhat different. The optimal-value function for forward induction is defined by:

$u_n(s_n) =$ Optimal value (minimum delay) over the current andcompleted stages (intersections), given that we are in state s_n (in a particular intersection) with n stages (intersections) to go.

The recursive relationship for forward induction on the minimum-delay problem is

$$u_{n-1}(s_{n-1}) = \text{Min}\{u_n(s_n) + t_{n-1}(s_{n-1})\}, \tag{3}$$

subject to:

$$s_{n-1} = \begin{cases} s_n + 1 & \text{if we choose up and } n \text{ even,} \\ s_n - 1 & \text{if we choose down and } n \text{ odd,} \\ s_n & \text{otherwise,} \end{cases}$$

where the stages are numbered in terms of intersections to go. The computations are carried out by setting

$$u_5(s_5) = t_5(s_5) \qquad (s_5 = 1, 2, \ldots, 6), \tag{4}$$

and successively applying (3). The calculations for forward induction are given in Fig. 11.6. When performing forward induction, the stages are usually numbered in terms of the number of stages *completed* (rather than the number of stages to go). However, in order to make a comparison between the two approaches easier, we have avoided using the ‘‘stages completed" numbering.

The columns of Fig. 11.6(f) give the optimal-value function at each stage for the minimum-delay problem, computed by forward induction. This figure gives the minimum delays from each particular downtown parking lot to the *group* of homes of the commuters. Therefore, this approach will only guarantee finding the minimum delay path from the downtown parking lots to *one* of the commuters’ homes. The method, in fact, finds the minimum-delay path to a particular origin only if that origin may be reached from a downtown parking lot by a backward sequence of arrows in Fig. 11.6(f).

If we select the minimum-delay path in Fig. 11.6(f), lasting 18 minutes, and follow the arrows backward, we discover that this path leads to the intersection second from the bottom in the first column. This is the same minimum-delay path determined by backward induction in Fig. 11.5(c).

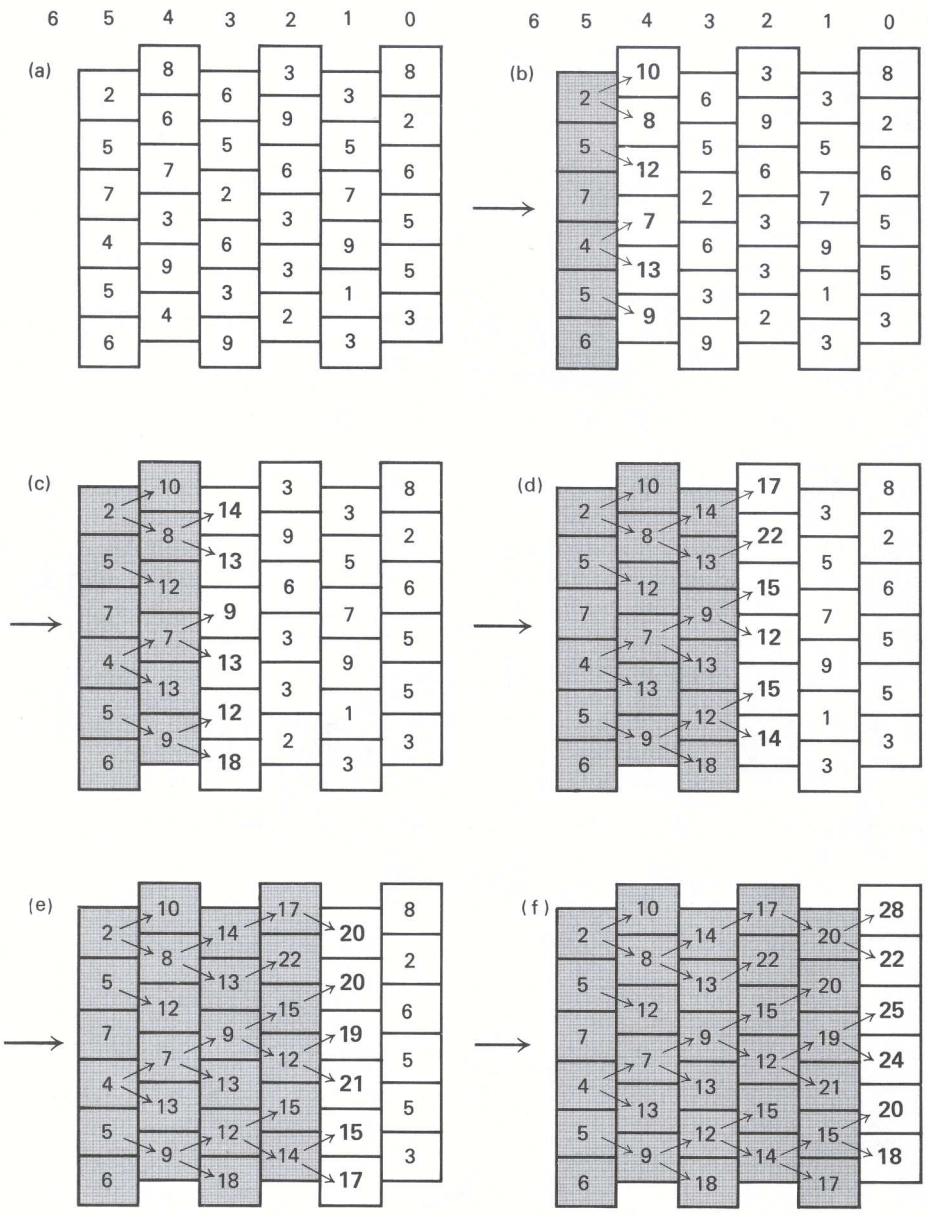


Figure 11.6 Solution by forward induction.

Forward induction determined the minimum-delay paths from each individual parking lot to the *group of homes*, while backward induction determined the minimum-delay paths from each individual home to the *group of downtown parking lots*. The minimum-delay path between the two groups is guaranteed to be the same in each case but, in general, the remaining paths determined may be different. Therefore, when using dynamic programming, it is necessary to think about whether forward or backward induction is best suited to the problem you want to solve.

11.2 FORMALIZING THE DYNAMIC-PROGRAMMING APPROACH

The elementary example presented in the previous section illustrates the three most important characteristics of dynamic-programming problems:

Stages

The essential feature of the dynamic-programming approach is the structuring of optimization problems into multiple *stages*, which are solved sequentially one stage at a time. Although each one-stage problem is solved as an ordinary optimization problem, its solution helps to define the characteristics of the next one-stage problem in the sequence.

Often, the stages represent different time periods in the problem's planning horizon. For example, the problem of determining the level of inventory of a single commodity can be stated as a dynamic program. The decision variable is the amount to order at the beginning of each month; the objective is to minimize the total ordering and inventory-carrying costs; the basic constraint requires that the demand for the product be satisfied. If we can order only at the beginning of each month and we want an optimal ordering policy for the coming year, we coulddecompose the problem into 12 stages, each representing the ordering decision at the beginning of the corresponding month.

Sometimes the stages do not have time implications. For example, in the simple situation presented in the preceding section, the problem of determining the routes of minimum delay from the homes of the commuters to the downtown parking lots was formulated as a dynamic program. The decision variable was whether to choose *up* or *down* in any intersection, and the stages of the process were defined to be the number of intersections to go. Problems that can be formulated as dynamic programs with stages that do not have time implications are often difficult to recognize.

States

Associated with each stage of the optimization problem are the *states* of the process. The states reflect the information required to fully assess the consequences that the current decision has upon future actions. In the inventory problem given in this section, each stage has only one variable describing the state: the inventory level on hand of the single commodity. The minimum-delay problem also has one state variable: the intersection a commuter is in at a particular stage.

The specification of the states of the system is perhaps the most critical design parameter of the dynamic-programming model. There are no set rules for doing this. In fact, for the most part, this is an art often requiring creativity and subtle insight about the problem being studied. The essential properties that should motivate the selection of states are:

i) The states should convey enough information to make future decisions without regard to how the process reached the current state; and

ii) The number of state variables should be small, since the computational effort associated with the dynamic-programming approach is prohibitively expensive when there are more than two, or possibly three, state variables involved in the model formulation.

This last feature considerably limits the applicability of dynamic programming in practice.

Recursive Optimization

The final general characteristic of the dynamic-programming approach is the development of a *recursive optimization* procedure, which builds to a solution of the overall N-stage problem by first solving a one-stage problem and sequentially including one stage at a time and solving one-stage problems until the overall optimum has been found. This procedure can be based on a *backward induction* process, where the first stage to be analyzed is the final stage of the problem and problems are solved moving back one stage at a time until all stages are included. Alternatively, the recursive procedure can be based on a *forward induction* process, where the first stage to be solved is the initial stage of the problem and problems are solved moving forward one stage at a time, until all stages are included. In certain problem settings, only one of these induction processes can be applied (e.g., only backward induction is allowed in most problems involving uncertainties).

The basis of the recursive optimization procedure is the so-called *principle of optimality*, which has already been stated: an optimal policy has the property that, whatever the current state and decision, the remaining decisions must constitute an optimal policy with regard to the state resulting from the current decision.

General Discussion

In what follows, we will formalize the ideas presented thus far. Suppose we have a multistage decision process where the *return* (or cost) for a particular *stage* is:

$$f_n(d_n, s_n), \tag{5}$$

where d_n is a permissible *decision* that may be chosen from the set D_n, and s_n is the *state* of the process with n stages to go. Normally, the set of feasible decisions, D_n, available at a given stage depends upon the state of the process at that stage, s_n, and could be written formally as $D_n(s_n)$. To simplify our presentation, we will denote the set of feasible decisions simply as D_n. Now, suppose that there are a total of N stages in the process and we continue to think of n as the number of stages *remaining* in the process. Necessarily, this view implies a finite number of stages in the decision process and therefore a specific horizon for a problem involving time. Further, we assume that the state s_n of the system with n stages to go is a full description of the system for decision-making purposes and that knowledge of prior states is unnecessary. The next state of the process depends entirely on the current state of the process and the current decision taken. That is, we can define a *transition function* such that, given s_n, the state of the process with n stages to go, the subsequent state of the process with $(n-1)$ stages to go is given by

$$s_{n-1} = t_n(d_n, s_n), \tag{6}$$

where d_n is the decision chosen for the current stage and state. Note that there is no uncertainty as to what the next state will be, once the current state and current decision are known. In Section 11.7, we will extend these concepts to include uncertainty in the formulation.

Our multistage decision process can be described by the diagram given in Fig. 11.7. Given the current state s_n which is a complete description of the system for decision-making purposes with n stages to go, we want to choose the decision d_n that will maximize the total return over the remaining stages. The decision d_n, which must be chosen from a set D_n of permissible decisions, produces a return at this stage of $f_n(d_n, s_n)$ and results in a new state s_{n-1} with $(n-1)$ stages to go. The new state at the beginning of the next stage is determined by the transition function $s_{n-1} = t_n(d_n, s_n)$, and the new state is a complete description of the system for decision-making purposes with $(n-1)$ stages to go. Note that the stage returns are independent of one another.

In order to illustrate these rather abstract notions, consider a simple inventory example. In this case, the state s_n of the system is the inventory level I_n with n months to go in the planning horizon. The decision d_n is the amount O_n to order this month. The resulting inventory level I_{n-1} with $(n-1)$ months to go is given by the usual inventory-balance relationship:

$$I_{n-1} = I_n + O_n - R_n,$$

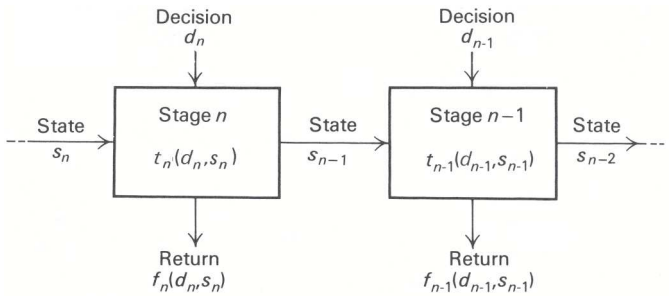


Figure 11.7 Multistage decision process.

where R_n is the demand requirement this month. Thus, formally, the transition function with n stages to go is defined to be:

$$I_{n-1} = t_n(I_n, O_n) = I_n + O_n - R_n.$$

The objective to be minimized is the total ordering and inventory-carrying costs, which is the sum of the one-stage costs $C_n(I_n, O_n)$.

For the general problem, our objective is to maximize the sum of the return functions (or minimize the sum of cost functions) over all stages of the decision process; and our only constraints on this optimization are that the decision chosen for each stage belong to some set D_n of permissible decisions and that the transitions from state to state be governed by Eq. (6). Hence, given that we are in state s_n with n stages to go, our optimization problem is to choose the decision variables $d_n, d_{n-1}, \ldots, d_0$ to solve the following problems:

$$v_n(s_n) = \text{Max}\left[f_n(d_n, s_n) + f_{n-1}(d_{n-1}, s_{n-1}) + \cdots + f_0(d_0, s_0) \right],$$

subject to:

$$\begin{aligned} &s_{m-1} = t_m(d_m, s_m) && (m = 1, 2, \ldots, n), \\ &d_m \in D_m && (m = 0, 1, \ldots, n). \end{aligned} \tag{7}$$

We call $v_n(s_n)$ the *optimal-value function*, since it represents the maximum return possible over the n stages to go. Formally, we define:

$v_n(s_n)$ = Optimal value of all subsequent decisions, given that we are in state s_n with n stages to go.

Now since $f_n(d_n, s_n)$ involves only the decision variable d_n and not the decision variables $d_{n-1}, \ldots, d_0$, we could first maximize over this latter group for every possible d_n and then choose d_n so as to maximize the entire expression. Therefore, we can rewrite Eq. (7) as follows:

$$\begin{aligned} v_n(s_n) = \text{Max}\{ & f_n(d_n, s_n) + \text{Max}\left[f_{n-1}(d_{n-1}, s_{n-1}) + \cdots + f_0(d_0, s_0) \right] \}, \\ & \text{subject to:} \qquad\qquad \text{subject to:} \\ & s_{n-1} = t_n(d_n, s_n) \qquad s_{m-1} = t_m(d_m, s_m) \qquad (m = 1, 2, \ldots, n-1), \\ & d_n \in D_n, \qquad\qquad d_m \in D_m, \qquad\qquad (m = 0, 1, \ldots, n-1). \end{aligned} \tag{8}$$

Note that the second part of Eq. (8) is simply the optimal-value function for the $(n-1)$-stage dynamic-programming problem defined by replacing n with $(n-1)$ in (7). We can therefore rewrite Eq. (8) as the following recursive relationship:

$$v_n(s_n) = \text{Max}\left[f_n(d_n, s_n) + v_{n-1}(s_{n-1}) \right],$$

subject to:

$$\begin{aligned} &s_{n-1} = t_n(d_n, s_n), \\ &d_n \in D_n. \end{aligned} \tag{9}$$

To emphasize that this is an optimization over d_n, we can rewrite Eq. (9) equivalently as:

$$v_n(s_n) = \text{Max}\{f_n(d_n, s_n) + v_{n-1}[t_n(d_n, s_n)]\}, \tag{10}$$

subject to:

$$d_n \in D_n.$$

The relationship in either Eq. (9) or (10) is a formal statement of the *principle of optimality*. As we have indicated, this principle says that an optimal sequence of decisions for a multistage problem has the property that, regardless of the current decision d_n and current state s_n, all subsequent decisions must be optimal, given the state s_{n-1} resulting from the current decision.

Since $v_n(s_n)$ is defined recursively in terms of $v_{n-1}(s_{n-1})$, in order to solve Eqs. (9) or (10) it is necessary to initiate the computation by solving the ‘‘stage-zero" problem. The stage-zero problem is not defined recursively, since there are no more stages after the final stage of the decision process. The stage-zero problem is then the following:

$$v_0(s_0) = \text{Max}\, f_0(d_0, s_0), \tag{11}$$

subject to:

$$d_0 \in D_0.$$

Often there *is* no stage-zero problem, as $v_0(s_0)$ is identically zero for all final stages. In the simple example of the previous section, where we were choosing the path of minimum delay through a sequence of intersections, the stage-zero problem consisted of accepting the delay for the intersection corresponding to each final state.

In this discussion, we have derived the optimal-value function for *backward induction*. We could easily have derived the optimal-value function for *forward induction*, as illustrated in the previous section. However, rather than develop the analogous result, we will only state it here. Assuming that we continue to number the states ‘‘backwards," we can define the optimal-value function for forward induction as follows:

$u_n(s_n) =$ Optimal value of all prior decisions, given that we are in state s_n with n stages to go.

The optimal-value function is then given by:

$$u_{n-1}(s_{n-1}) = \text{Max}\,[u_n(s_n) + f_n(d_n, s_n)], \tag{12}$$

subject to:

$$s_{n-1} = t_n(d_n, s_n),$$
$$d_n \in D_n,$$

where the computations are usually initialized by setting

$$u_n(s_n) = 0,$$

or by solving some problem, external to the recursive relationship, that gives a value to being in a particular initial state. Note that, for forward induction, you need to think of the problem as one of examining all the combinations of current states and actions that produce a specific state at the next stage, and then choose optimally among these combinations.

It should be pointed out that nothing has been said about the specific form of the stage-return functions or the set of permissible decisions at each stage. Hence, what we have said so far holds regardless of whether the decisions are discrete, continuous, or mixtures of the two. All that is necessary is that the recursive relationship be solvable for the optimal solution at each stage, and then a *global* optimal solution to the overall problem is determined. The optimization problem that is defined at each stage could lead to the application of a wide variety of techniques, i.e., linear programming, network theory, integer programming, and so forth, depending on the nature of the transition function, the constraint set D_n, and the form of the function to be optimized.

It should also be pointed out that nowhere in developing the fundamental recursive relationship of dynamic programming was any use made of the fact that there were a finite number of states at each stage. In fact, Eqs. (9), (10), and (12) hold independent of the number of states. The recursive relationship merely needs to be solved for all possible states of the system at each stage. If the state space, i.e., the set of possible states, is continuous, and therefore an infinite number of states are possible at each stage, then the number of states is usually made finite by making a discrete approximation of the set of possible states, and the same procedures are used. An example of a dynamic-programming problem with a continuous state space is given in Section 11.6.

Finally, we have assumed certainty throughout out discussion so far; this assumption will be relaxed in Section 11.7, and a very similar formal structure will be shown to hold.

11.3 OPTIMAL CAPACITY EXPANSION

In this section, we further illustrate the dynamic-programming approach by solving a problem of optimal capacity expansion in the electric power industry.

A regional electric power company is planning a large investment in nuclear power plants over the next few years. A total of eight nuclear power plants must be built over the next six years because of both increasing demand in the region and the energy crisis, which has forced the closing of certain of their antiquated fossil-fuel plants. Suppose that, for a first approximation, we assume that demand for electric power in the region is known with certainty and that we must satisfy the minimum levels of cumulative demand indicated in Table 11.1. The demand here has been converted into equivalent numbers of nuclear power plants required by the end of each year. Due to the extremely adverse public reaction and subsequent difficulties with the public utilities commission, the power company has decided at least to meet this minimum-demand schedule.

The building of nuclear power plants takes approximately one year. In addition to a cost directly associated with the construction of a plant, there is a common cost of $1.5 million incurred when any plants are constructed in any year, independent of the number of plants constructed. This common cost results from contract preparation and certification of the impact statement for the Environmental Protection Agency. In any given year, at most three plants can be constructed. The cost of construction per plant is given in Table 11.1 for each year in the planning horizon. These costs are currently increasing due to the elimination of an investment tax credit designed to speed investment in nuclear power. However, new technology should be available by 1984, which will tend to bring the costs down, even given the elimination of the investment tax credit.

We can structure this problem as a dynamic program by defining the state of the system in terms of the cumulative capacity attained by the end of a particular year. Currently, we have no plants under construction, and by the end of each year in the planning horizon we must have completed a number of plants equal to or greater than the cumulative demand. Further, it is assumed that there is no need ever to construct more than eight plants. Figure 11.8 provides a graph depicting the allowable capacity (states) over time. Any node of this graph is completely described by the corresponding year number and level of cumulative capacity, say the node (n, p). Note that we have chosen to measure time in terms of *years to go* in the planning horizon. The cost of traversing any upward-sloping arc is the common cost of $1.5 million

Table 11.1 Demand and cost per plant ($ ×1000)

Year	*Cumulative demand* (in number of plants)	*Cost per plant* ($ × 1000)
1981	1	5400
1982	2	5600
1983	4	5800
1984	6	5700
1985	7	5500
1986	8	5200

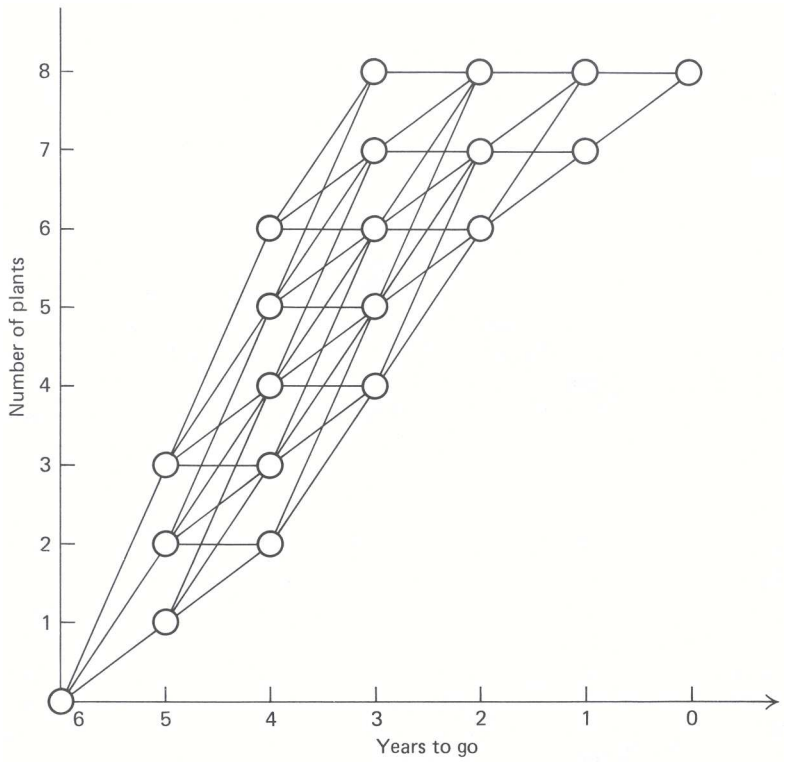


Figure 11.8 Allowable capacity (states) for each tage

plus the plant costs, which depend upon the year of construction and whether 1, 2, or 3 plants are completed. Measured in thousands of dollars, these costs are

$$1500 + c_n x_n,$$

where c_n is the cost per plant in the year n and x_n is the number of plants constructed. The cost for traversing any horizontal arc is zero, since these arcs correspond to a situation in which no plant is constructed in the current year.

Rather than simply developing the optimal-value function in equation form, as we have done previously, we will perform the identical calculations in tableau form to highlight the dynamic-programming methodology. To begin, we label the final state zero or, equivalently define the ‘‘stage-zero" optimal-value function to be zero for all possible states at stage zero. We will define a state as the cumulative total number of plants completed. Since the only permissible final state is to construct the entire cumulative demand of eight plants, we have $s_0 = 8$ and

$$v_0(8) = 0.$$

Now we can proceed recursively to determine the optimal-value function with one stage remaining. Since the demand data requires 7 plants by 1985, with one year to go the only permissible states are to have completed 7 or 8 plants. We can describe the situation by Tableau 1.

The dashes indicate that the particular combination of current state and decision results in a state that is not permissible. In this table there are no choices, since, if we have not already completed eight plants, we will construct one more to meet the demand. The cost of constructing the one additional plant is the $1500 common cost plus the $5200 cost per plant, for a total of $6700. (All costs are measured in thousands of dollars.) The column headed $d_1^*(s_1)$ gives the optimal decision function, which specifies the optimal number of plants to construct, given the current state of the system.

Now let us consider what action we should take with two years (stages) to go. Tableau 2 indicates the possible costs of eachstate:

Tableau 1

Plants completed (s_1)	Possible new plants: 0	Possible new plants: 1	$v_1(s_1)$	$d_1^*(s_1)$
8	0	—	0	0
7	—	6,700	6,700	1

$c_1(s_1, d_1)$

Tableau 2

s_2 \ d_2	0	1	2	$v_2(s_2)$	$d_2^*(s_2)$
8	0	—	—	0	0
7	6,700	7,000	—	6,700	0
6	—	13,700	12,500	12,500	2

$c_2(s_2, d_2) + v_1(s_1)$

If we have already completed eight plants with two years to go, then clearly we will not construct any more. If we have already completed seven plants with two years to go, then we can either construct the one plant we need this year or postpone its construction. Constructing the plant now costs \$1500 in common costs plus \$5500 in variable costs, and results in state 8 with one year to go. Since the cost of state 8 with one year to go is zero, the total cost over the last two years is \$7000. On the other hand, delaying construction costs zero this year and results in state 7 with one year to go. Since the cost of state 7 with one year to go is \$6700, the total cost over the last two years is \$6700. If we arrive at the point where we have two years to go and have completed seven plants, it pays to delay the production of the last plant needed. In a similar way, we can determine that the optimal decision when in state 6 with two years to go is to construct two plants during the next year.

To make sure that these ideas are firmly understood, we will determine the optimal-value function and optimal decision with three years to go. Consider Tableau 3 for three years to go:

Tableau 3

s_3 \ d_3	0	1	2	3	$v_3(s_3)$	$d_3^*(s_3)$
8	0	—	—	—	0	0
7	6,700	7,200	—	—	6,700	0
6	12,500	13,900	12,900	—	12,500	0
5	—	19,700	19,600	18,600	18,600	3
4	—	—	25,400	25,300	25,300	3

$c_3(s_3, d_3) + v_2(s_2)$

Now suppose that, with three years to go, we have completed five plants. We need to construct at least one plant this year in order to meet demand. In fact, we can construct either 1, 2, or 3 plants. If we construct one plant, it costs \$1500 in common costs plus \$5700 in plant costs, and results in state 6 with two years to go. Since the minimum cost following the optimal policy for the remaining two years is then \$12,500, our total cost for three years would be \$19,700. If we construct two plants, it costs the \$1500 in common costs plus \$11,400 in plant costs and results in state 7 with two years to go. Since the minimum cost following the optimal policy for the remaining two years is then \$6700, our total cost for three years would be \$19,600. Finally, if we construct three plants, it costs the \$1500 in common costs plus \$17,100 in plant costs and results in state 8 with two years to go. Since the minimum cost following the optimal policy for the remaining

Tableau 4

s_4 \ d_4	0	1	2	3	$v_4(s_4)$	$d_4^*(s_4)$
6	12,500	14,000	13,100	—	12,500	0
5	18,600	19,800	19,800	18,900	18,600	0
4	25,300	25,900	25,600	25,600	25,300	0
3	—	32,600	31,700	31,400	31,400	3
2	—	—	38,400	37,500	37,500	3

$c_4(s_4, d_4) + v_3(s_3)$

Tableau 5

s_5 \ d_5	0	1	2	3	$v_5(s_5)$	$d_5^*(s_5)$
3	31,400	32,400	31,300	30,800	30,800	3
2	37,500	38,500	38,000	36,900	36,900	3
1	—	44,600	44,100	43,600	43,600	3

$c_5(s_5, d_5) + v_4(s_4)$

Tableau 6

s_6 \ d_6	0	1	2	3	$v_6(s_6)$	$d_6^*(s_6)$
0	—	50,500	49,200	48,800	48,800	3

$c_6(s_6, d_6) + v_5(s_5)$

Figure 11.9 Tableaus to complete power-plant example.

two years is then zero, our total cost for three years would be \$18,600. Hence, the optimal decision, having completed five plants (being in state 5) with three years (stages) to go, is to construct three plants this year. The remaining tableaus for the entire dynamic-programming solution are determined in a similar manner (see Fig. 11.9).

Since we start the construction process with no plants (i.e., in state 0) with six years (stages) to go, we can proceed to determine the optimal sequence of decisions by considering the tableaus in the reverse order. With six years to go it is optimal to construct three plants, resulting in state 3 with five years to go. It is then optimal to construct three plants, resulting in state 6 with four years to go, and so forth. The optimal policy is then shown in the tabulation below:

Years to go	*Construct*	*Resulting state*
6	3	3
5	3	6
4	0	6
3	0	6
2	2	8
1	0	8

Hence, from Tableau 6, the total cost of the policy is \$48.8 million.

11.4 DISCOUNTING FUTURE RETURNS

In the example on optimal capacity expansion presented in the previous section, a very legitimate objection might be raised that the *present value of money* should have been taken into account in finding the optimal construction schedule. The issue here is simply that a dollar received today is clearly worth more than a

dollar received one year from now, since the dollar received today could be invested to yield some additional return over the intervening year. It turns out that dynamic programmingis extremely well suited to take this into account.

We will define, in the usual way, the one-period *discount factor* β as the present value of one dollar received *one period from now*. In terms of interest rates, if the interest rate for the period were i, then one dollar invested now would accumulate to $(1+i)$ at the end of one period. To see the relationship between the discount factor β and the interest rate i, we ask the question ''How much must be invested now to yield one dollar one period from now?" This amount is clearly the present value of a dollar received one period from now, so that $\beta(1+i)=1$ determines the relationship between β and i, namely, $\beta = 1/(1+i)$. If we invest one dollar now for n periods at an interest rate per period of i, then theaccumulated value at the end of n periods, assuming the interest is compounded, is $(1+i)^n$. Therefore, the present value of one dollar received n periods from now is $1/(1+i)^n$ or, equivalently, β^n.

The concept of discounting can be incorporated into the dynamic-programming framework very easily since we often have a return per period (stage) that we may wish to discount by the per-period discount factor. If we have an n-stage dynamic-programming problem, the optimal-value function, including the appropriate discounting of future returns, is given by

$$v_n(s_n) = \text{Max}\,[f_n(d_n, s_n) + \beta f_{n-1}(d_{n-1}, s_{n-1}) + \beta^2 f_{n-2}(d_{n-2}, s_{n-2}) + \cdots + \beta^n f_0(d_0, s_0)],$$

subject to:

$$\begin{aligned} &s_{m-1} = t_m(d_m, s_m) \quad (m = 1, 2, \ldots, n), \\ &d_m \in D_m, \qquad\qquad (m = 0, 1, \ldots, n), \end{aligned} \tag{13}$$

where the stages (periods) are numbered in terms of *stages to go*. Making the same argument as in Section 11.3 and factoring out the β, we can rewrite Eq. (13) as:

$$v_n(s_n) = \text{Max}\,\{f_n(d_n, s_n) + \beta\,\text{Max}\,[f_{n-1}(d_{n-1}, s_{n-1}) + \beta f_{n-2}(d_{n-2}, s_{n-2}) + \cdots + \beta^{n-1} f_0(d_0, s_0)],$$

$$\begin{array}{lll} \text{subject to:} & \text{subject to:} & \\ s_{n-1} = t_n(d_n, s_n) & s_{m-1} = t_m(d_m, s_m) & (m = 1, 2, \ldots, n-1), \\ d_n \in D_n & d_m \in D_m & (m = 0, 1, \ldots, n-1). \end{array} \tag{14}$$

Since the second part of Eq. (14) is simply the optimal-value function for the $(n-1)$-stage problem multiplied by β, we can rewrite Eq. (14) as

$$v_n(s_n) = \text{Max}[f_n(d_n, s_n) + \beta v_{n-1}(s_{n-1})],$$

subject to:

$$\begin{aligned} &s_{n-1} = t_n(d_n, s_n), \\ &d_n \in D_n, \end{aligned} \tag{15}$$

which is simply the recursive statement of the optimal-value function for backward induction with discounting. If $\beta = 1$, we have the case of *no discounting* and Eq. (15) is identical to Eq. (9). Finally, if the discount rate depends on the period, β can be replaced by β_n and (15) still holds.

We can look at the impact of discounting future-stage returns by considering again the optimal capacity expansion problem presented in the previous section. Suppose that the alternative uses of funds by the electric power company result in a 15 percent return on investment. This corresponds to a yearly discount factor of approximately 0.87. If we merely apply backward induction to the capacity expansion problem according to Eq. (15), using $\beta = 0.87$, we obtain the optimal-value function for each stage as given in Fig. 11.10.

s_1 \ d_1	0	1	$v_1(s_1)$	$d_1^*(s_1)$
8	0	—	0	0
7	—	6,700	6,700	1

$c_1(s_1, d_1)$

s_2 \ d_2	0	1	2	$v_2(s_2)$	$d_2^*(s_2)$
8	0	—	—	0	0
7	5,829	7,000	—	5,829	0
6	—	12,825	12,500	12,500	2

$c_2(s_2, d_2) + \beta v_1(s_1)$

s_3 \ d_3	0	1	2	3	$v_3(s_3)$	$d_3^*(s_3)$
8	0	—	—	—	0	0
7	5,071	7,200	—	—	5,071	0
6	10,875	12,271	12,900	—	10,875	0
5	—	18,075	17,971	18,600	17,971	2
4	—	—	23,775	23,671	23,671	3

$c_3(s_3, d_3) + \beta v_2(s_2)$

s_4 \ d_4	0	1	2	3	$v_4(s_4)$	$d_4^*(s_4)$
6	9,461	11,712	13,100	—	9,461	0
5	15,635	16,761	17,512	18,900	15,635	0
4	20,594	22,935	22,561	23,312	20,594	0
3	—	27,894	28,735	28,361	27,894	1
2	—	—	33,694	34,535	33,694	2

$c_4(s_4, d_4) + \beta v_3(s_3)$

s_5 \ d_5	0	1	2	3	$v_5(s_5)$	$d_5^*(s_5)$
3	24,268	25,017	26,302	26,531	24,268	0
2	29,314	31,368	30,617	31,902	29,314	0
1	—	36,414	36,968	36,217	36,217	3

$c_5(s_5, d_5) + \beta v_4(s_4)$

s_6 \ d_6	0	1	2	3	$v_6(s_6)$	$d_6^*(s_6)$
0	—	38,409	37,803	38,813	37,803	2

$c_6(s_6, d_6) + \beta v_5(s_5)$

Figure 11.10 Optimal-value and decision functions with discounting.

Given that the system is in state zero with six stages to go, we determine the optimal construction strategy by considering the optimal decision function $d_n^*(s_n)$ from stage 6 to stage 0. The optimal construction sequence is then shown in the following tabulation:

Stages to go	*Construct*	*Resulting state*
6	2	2
5	0	2
4	2	4
3	3	7
2	0	7
1	1	8

and the optimal value of the criterion function, *present value of total future costs*, is \$37.8 million for this strategy. Note that this optimal strategy is significantly different from that computed in the previous section without discounting. The effect of the discounting of future costs is to delay construction in general, which is what we would expect.

*11.5 SHORTEST PATHS IN A NETWORK

Although we have not emphasized this fact, dynamic-programming and shortest-path problems are very similar. In fact, as illustrated by Figs. 11.1 and 11.8, our previous examples of dynamic programming can both be interpreted as shortest-path problems.

In Fig. 11.8, we wish to move through the network from the starting node (initial state) at stage 6, with no plants yet constructed, to the end node (final state) at stage 0 with eight plants constructed. Every path in the network specifies a strategy indicating how many new plants to construct each year.

Since the cost of a strategy sums the cost at each stage, the total cost corresponds to the ‘‘length" of a path from the starting to ending nodes. The minimum-cost strategy then is just the shortest path.

Figure 11.11 illustrates a shortest-path network for the minimum-delay problem presented in Section 11.1. The numbers next to the arcs are delay times. An end node representing the group of downtown parking lots has been added. This emphasizes the fact that we have assumed that the commuters do not care in which lot they park. A start node has also been added to illustrate that the dynamic-programming solution by *backward* induction finds the shortest path from the end node to the start node. In fact, it finds the shortest paths from the end node to *all* nodes in the network, thereby solving the minimum-delay problem for each commuter. On the other hand, the dynamic-programming solution by *forward* induction finds the shortest path from the start node to the end node. Although the *shortest path* will be the same for both methods, forward induction will *not* solve the minimum-delay problem for *all* commuters, since the commuters are not indifferent to which home they arrive.

To complete the equivalence that we have suggested between dynamic programming and shortest paths, we next show how shortest-path problems can be solved by dynamic programming. Actually, several different dynamic-programming solutions can be given, depending upon the structure of the network under study. As a general rule, the more *structured* the network, the more efficient the algorithm that can be developed. To illustrate this point we give two separate algorithms applicable to the following types of networks:

i) *Acyclic networks.* These networks contain no directed cycles. That is, we cannot start from any node and follow the arcs in their given directions to return to the same node.

ii) *Networks without negative cycles.* These networks may contain cycles, but the distance around any cycle (i.e., the sum of the lengths of its arcs) must be nonnegative.

In the first case, to take advantage of the acyclic structure of the network, we order the nodes so that, if the network contains the arc i–j, then $i > j$. To obtain such an ordering, begin with the terminal node, which can be thought of as having only entering arcs, and number it ‘‘one." Then ignore that node and the incident arcs, and number any node that has only incoming arcs as the next node. Since the network is acyclic, there must be

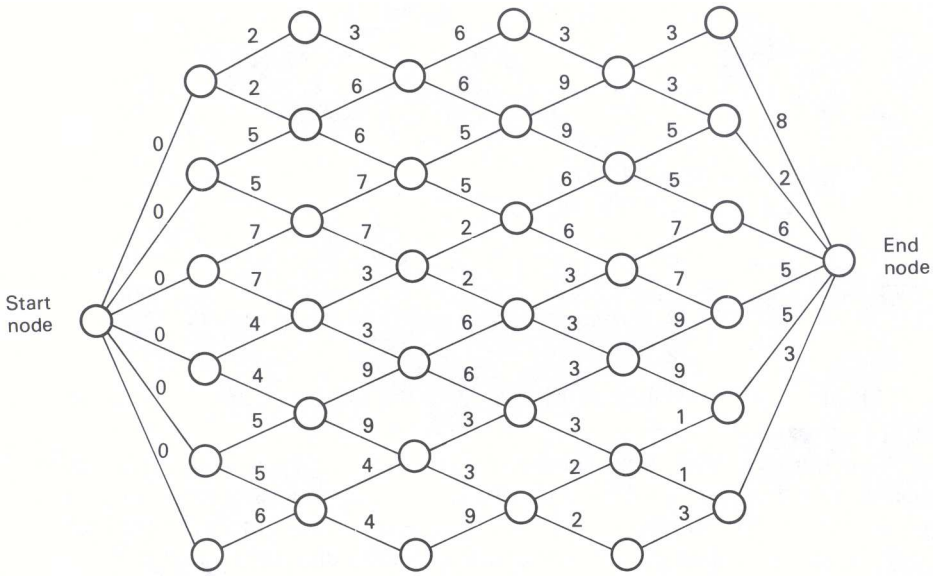


Figure 11.11 Shortest-path network for minimum-delay problem.

such a node. (Otherwise, from any node, we can move along an arc to another node. Starting from any node and continuing to move away from any node encountered, we eventually would revisit a node, determining a cycle, contradicting the acyclic assumption.) By ignoring the numbered nodes and their incident arcs, the procedure is continued until all nodes are numbered.

This procedure is applied, in Fig. 11.12, to the longest-path problem introduced as a critical-path scheduling example in Section 8.1.

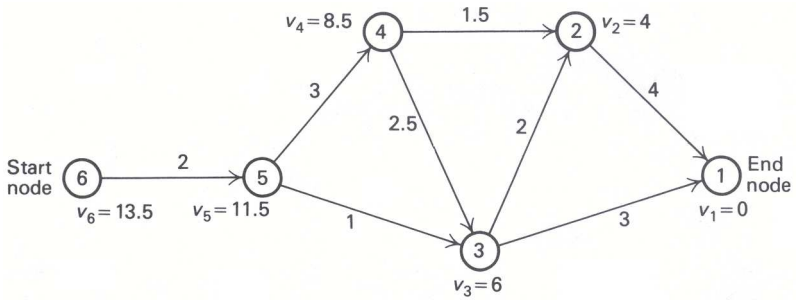


Figure 11.12 Finding the longest path in an acyclic network.

We can apply the dynamic-programming approach by viewing each node as a stage, using either backward induction to consider the nodes in ascending order, or forward induction to consider the nodes in reverse order. For backward induction, v_n will be interpreted as the longest distance from node n to the end node. Setting $v_1 = 0$, dynamic programming determines $v_2, v_3, \ldots, v_N$ in order, by the recursion

$$v_n = \text{Max}[d_{nj} + v_j], \quad j < n,$$

where d_{nj} is the given distance on arc n–j. The results of this procedure are given as node labels in Fig. 11.12 for the critical-path example.

For a shortest-path problem, we use minimization instead of maximization in this recursion. Note that the algorithm finds the longest (shortest) paths from every node to the end node. If we want only the longest path to the start node, we can terminate the procedure once the start node has been labeled. Finally, we could have found the longest distances from the start node to all other nodes by labeling the nodes in the reverse order, beginning with the start node.

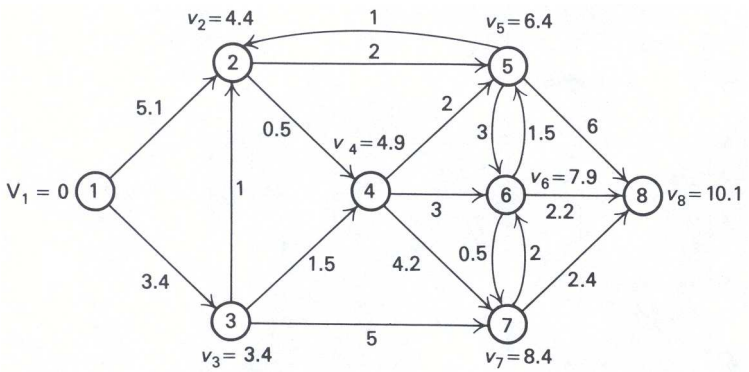


Figure 11.13 Shortest paths in a network without negative cycles.

A more complicated algorithm must be given for the more general problem of finding the shortest path between two nodes, say nodes 1 and N, in a network without negative cycles. In this case, we can devise a dynamic-programming algorithm based upon a value function defined as follows:

$v_n(j)$ = Shortest distance from node 1 to node j along paths using at most n intermediate nodes.

By definition, then,

$$v_0(j) = d_{1j} \quad \text{for } j = 2, 3, \ldots, N,$$

the length d_{1j} of arc 1–j since no intermediate nodes are used. The dynamic-programming recursion is

$$v_n(j) = \text{Min}\,\{d_{ij} + v_{n-1}(i)\}, \quad 1 \le j \le N, \tag{16}$$

which uses the principle of optimality: that any path from node 1 to node j, using at most n intermediate nodes, arrives at node j from node i along arc i–j after using the shortest path with at most $(n-1)$ intermediate nodes from node j to node i. We allow $i = j$ in the recursion and take $d_{jj} = 0$, since the optimal path using at most n intermediate nodes may coincide with the optimal path with length $v_{n-1}(j)$ using at most $(n-1)$ intermediate nodes.

The algorithm computes the shortest path from node 1 to every other node in the network. It terminates when $v_n(j) = v_{n-1}(j)$ for every node j, since computations in Eq. (16) will be repeated at every stage from n on. Because no path (without cycles) uses any more than $(N-1)$ intermediate nodes, where N is the total number of nodes, the algorithm terminates after at most $(N-1)$ steps.

As an application of the method, we solve the shortest-path problem introduced in Chapter 8 and given in Fig. 11.13.

Initially the values $v_0(j)$ are given by

$$v_0(1) = 0, \quad v_0(2) = d_{12} = 5.1, \quad v_0(3) = d_{13} = 3.4,$$

and

$$v_0(j) = \infty \quad \text{for } j = 4, 5, 6, 7, 8,$$

since these nodes are not connected to node 1 by an arc. The remaining steps are specified in Tableaus 7, 8, and 9. The computations are performed conveniently by maintaining a table of distances d_{ij}. If the list $v_0(i)$ is placed to the left of this table, then recursion Eq. (14) states that $v_1(j)$ is given by the smallest of the comparisons:

$$v_0(i) + d_{ij} \quad \text{for } i = 1, 2, \ldots, 8.$$

That is, place the column $v_0(i)$ next to the jth column of the d_{ij} table, add the corresponding elements, and take $v_1(j)$ as the smallest of the values. If $v_1(j)$ is recorded below the jth column, the next iteration to find $v_2(j)$ is initiated by replacing the column $v_0(i)$ with the elements $v_1(j)$ from below the distance table.

As the reader can verify, the next iteration gives $v_4(j) = v_3(j)$ for all j. Consequently, the values $v_3(j)$ recorded in Tableau 9 are the shortest distances from node 1 to each of the nodes $j = 2, 3, \ldots, 8$.

Tableau 7[†]

		Node j							
Node i	$v_0(i)$	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
1	0	0	5.1	3.4					
2	5.1		0		.5	2			
3	3.4		1	0	1.5			5	
4	$+\infty$				0	2	3	4.2	
5	$+\infty$					0	3		6
6	$+\infty$					2	0	.5	2.2
7	$+\infty$						2	0	2.4
8	$+\infty$								0
$v_1(j) = \min\{d_{ij} + v_0(i)\}$		0	4.4	3.4	4.9	7.1	$+\infty$	8.4	$+\infty$

Tableau 8[†]

		Node j							
Node i	$v_1(i)$	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
1	0	0	5.1	3.4					
2	4.4		0		.5	2			
3	3.4		1	0	1.5			5	
4	4.9				0	2	3	4.2	
5	7.1					0	3		6
6	$+\infty$					2	0	.5	2.2
7	8.4						2	0	2.4
8	$+\infty$								0
$v_2(j) = \min\{d_{ij} + v_1(i)\}$		0	4.4	3.4	4.9	6.4	7.9	8.4	10.8

Tableau 9[†]

		Node j							
Node i	$v_2(i)$	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
1	0	0	5.1	3.4					
2	4.4		0		.5	2			
3	3.4		1	0	1.5			5	
4	4.9				0	2	3	4.2	
5	6.4					0	3		6
6	7.9					2	0	.5	2.2
7	8.4						2	0	2.4
8	10.8								0
$v_3(j) = \min\ d_{ij} + v_2(i)$		0	4.4	3.4	4.9	6.4	7.9	8.4	10.1

[†] $d_{ij} = +\infty$, if blank.

11.6 CONTINUOUS STATE-SPACE PROBLEMS

Until now we have dealt only with problems that have had a finite number of states associated with each stage. Since we also have assumed a finite number of stages, these problems have been identical to finding the shortest path through a network with special structure. Since the development, in Section 11.3, of the fundamental recursive relationship of dynamic programming did not depend on having a finite number of states at each stage, here we introduce an example that has a continuous state space and show that the same procedures still apply.

Suppose that some governmental agency is attempting to perform cost/benefit analysis on its programs in order to determine which programs should receive funding for the next fiscal year. The agency has managed to put together the information in Table 11.2. The benefits of each program have been converted into equivalent tax savings to the public, and the programs have been listed by decreasing benefit-to-cost ratio. The agency has taken the position that there will be no partial funding of programs. Either a program *will* be funded at the indicated level or it will *not* be considered for this budget cycle. Suppose that the agency is fairly sure of receiving a budget of $34 million from the state legislature if it makes a good case that the money is being used effectively. Further, suppose that there is some possibility that the budget will be as high as $42 million. How can the agency make the most effective use of its funds at *either* possible budget level?

Table 11.2 Cost/benefit information by program.

Program	*Expected benefit*	*Expected cost*	*Benefit/Cost*
A	$ 59.2 M	$ 2.8 M	21.1
B	31.4	1.7	18.4
C	15.7	1.0	15.7
D	30.0	3.2	9.4
E	105.1	15.2	6.9
F	11.6	2.4	4.8
G	67.3	16.0	4.2
H	2.3	.7	3.3
I	23.2	9.4	2.5
J	18.4	10.1	1.8
	$364.2 M	$62.5 M	

We should point out that mathematically this problem is an integer program. If b_j is the benefit of the jth program and c_j is the cost of that program, then an integer-programming formulation of the agency's budgeting problem is determined easily.

Letting

$$x_j = \begin{cases} 1 & \text{if program } j \text{ is funded,} \\ 0 & \text{if program } j \text{ is } \textit{not} \text{ funded,} \end{cases}$$

the integer-programming formulation is:

$$\text{Maximize} \sum_{j=1}^{n} b_j x_j,$$

subject to:

$$\sum_{j=1}^{n} c_j x_j \le B,$$

$$x_j = 0 \quad \text{or} \quad 1 \qquad (j = 1, 2, \ldots, n),$$

where B is the total budget allocated. This cost/benefit example is merely a variation of the well-known knapsack problem that was introduced in Chapter 9.We will ignore, for the moment, this integer-programming formulation and proceed to develop a highly efficient solution procedure using dynamic programming.

In order to approach this problem via dynamic programming, we need to define the stages of the system, the state space for each stage, and the optimal-value function.

Let

$v_k(B)$ = Maximum total benefit obtainable, choosing from the first k programs, with budget limitation B.

With this definition of the optimal-value function, we are letting the first k programs included be the number of ‘‘stages to go" and the available budget at each stage be the state space. Since the possible budget might take on any value, we are allowing for a continuous state space for each stage. In what follows the order of the projects is immaterial although the order given in Table 11.2 may have some computational advantages.

Let us apply the dynamic-programming reasoning as before. It is clear that with $k = 0$ programs, the total benefit must be zero regardless of the budget limitation. Therefore

$$v_0(B_0) = 0 \qquad \text{for } B_0 \geq 0.$$

If we now let $k = 1$, it is again clear that the optimal-value function can be determined easily since the budget is either large enough to fund the first project, or *not*. (See Tableau 10.)

Tableau 10

B_1 \ d_1	$x_1 = 0$	$x_1 = 1$	$v_1(B_1)$	$d_1^*(B_1)$
$2.8 \leqq B$	0	59.2	59.2	1
$0 \leqq B < 2.8$	0	—	0	0
	$c_1(B_1, d_1)$			

Now consider which programs to fund when the first two programs are available. The optimal-value function $v_2(B_2)$ and optimal decision function $d_2^*(B_2)$ are developed in Tableau 11.

Tableau 11

B_2 \ d_2	$x_2 = 0$	$x_2 = 1$	$v_2(B_2)$	$d_2^*(B_2)$
$4.5 \leqq B_2$	59.2	59.2 + 31.4	90.6	1
$2.8 \leqq B_2 < 4.5$	59.2	31.4	59.2	0
$1.7 \leqq B_2 < 2.8$	0	31.4	31.4	1
$0 \leqq B_2 < 1.7$	0	—	0	0
	$c_2(B_2, d_2) + v_1(B_1)$			

Here again the dash means that the current state and decision combination will result in a state that is not permissible. Since this tableau is fairly simple, we will go on and develop the optimal-value function $v_3(B_3)$ and optimal decision function $d_3^*(B_3)$ when the first three programs are available (see Tableau 12).

For any budget level, for example, \$4.0 M, we merely consider the two possible decisions: either funding program C ($x_3 = 1$) or not ($x_3 = 0$). If we fund program C, then we obtain a benefit of \$15.7 M while consuming \$1.0 M of our own budget. The remaining \$3.0 M of our budget is then optimally allocated to the remaining programs, producing a benefit of \$59.2 M, which we obtain from the optimal-value function with the first two programs included (Tableau 11). If we do not fund program C, then the entire amount of \$4.0 M is optimally allocated to the remaining two programs (Tableau 11), producing a benefit of \$59.2. Hence, we should clearly fund program C if our budget allocation is \$4.0 M. Optimal decisions taken for other budget levels are determined in a similar manner.

Tableau 12

B_3 \ d_3	$x_3 = 0$	$x_3 = 1$	$v_3(B_3)$	$d_3^*(B_3)$
$5.5 \leqq B_3$	90.6	90.6 + 15.7	106.3	1
$4.5 \leqq B_3 < 5.5$	90.6	59.2 + 15.7	90.6	0
$3.8 \leqq B_3 < 4.5$	59.2	59.2 + 15.7	74.9	1
$2.8 \leqq B_3 < 3.8$	59.2	31.4 + 15.7	59.2	0
$2.7 \leqq B_3 < 2.8$	31.4	31.4 + 15.7	47.1	1
$1.7 \leqq B_3 < 2.7$	31.4	0 + 15.7	31.4	0
$1.0 \leqq B_3 < 1.7$	0	0 + 15.7	15.7	1
$0 \leqq B_3 < 1.0$	0	—	0	0
		$c_3(B_3, d_3) + v_2(B_2)$		

Although it is straightforward to continue the recursive calculation of the optimal-value function for succeeding stages, we will not do so since the number of ranges that need to be reported rapidly becomes rather large. The general recursive relationship that determines the optimal-value function at each stage is given by:

$$v_n(B_n) = \text{Max } [c_n x_n + v_{n-1}(B_n - c_n x_n)],$$

subject to:

$$x_n = 0 \quad \text{or} \quad 1.$$

The calculation is initialized by observing that

$$v_0(B_0) = 0$$

for all possible values of B_0. Note that the state transition function is simply

$$B_{n-1} = t_n(x_n, B_n) = B_n - c_n x_n.$$

We can again illustrate the usual principle of optimality: Given budget B_n at stage n, whatever decision is made with regard to funding the nth program, the remaining budget must be allocated optimally among the first $(n - 1)$ programs. If these calculations were carried to completion, resulting in $v_{10}(B_{10})$ and $d_{10}^*(B_{10})$, then the problem would be solved for all possible budget levels, not just \$3.4 M and \$4.2 M.

Although this example has a continuous state space, a finite number of ranges can be constructed because of the zero–one nature of the decision variables. In fact, all breaks in the range of the state space either are the breaks from the previous stage, or they result from adding the cost of the new program to the breaks in the previous range. This is not a general property of continuous state space problems, and in most cases such ranges cannot be determined. Usually, what is done for continuous state space problems is that they are converted into discrete state problems by defining an appropriate grid on the continuous state space. The optimal-value function is then computed only for the points on the grid. For our cost/benefit example, the total budget must be between zero and \$62.5 M, which provides a range on the state space, although at any stage a tighter upper limit on this range is determined by the sum of the budgets of the first n programs. An appropriate grid would consist of increments of \$0.1 M over the limits of the range at each stage, since this is the accuracy with which the program costs have been estimated. The difference between problems with continuous state spaces and those with discrete state spaces essentially then disappears for computational purposes.

11.7 DYNAMIC PROGRAMMING UNDER UNCERTAINTY

Up to this point we have considered exclusively problems with deterministic behavior. In a deterministic dynamic-programming process, if the system is in state s_n with n stages to go and decision d_n is selected from the set of permissible decisions for this stage and state, then the stage return $f_n(d_n, s_n)$ and the state of

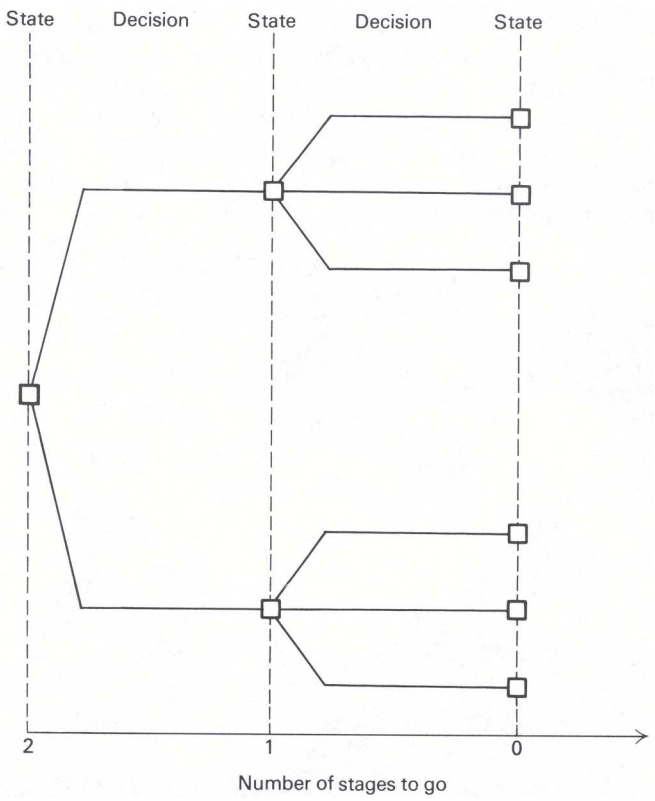


Figure 11.14 Decision tree for deterministic dynamic programming.

the system at the next stage, given by $s_{n-1} = t_n(d_n, s_n)$, are both known with certainty. This deterministic process can be represented by means of the decision tree in Fig. 11.14. As one can observe, given the current state, a specific decision leads with complete certainty to a particular state at the next stage. The stage returns are also known with certainty and are associated with the branches of the tree.

When uncertainty is present in a dynamic-programming problem, a specific decision for a given state and stage of the process does not, by itself, determine the state of the system at the next stage; this decision may not even determine the return for the current stage. Rather, in dynamic programming under uncertainty, given the state of the system s_n with n stages to go and the current decision d_n, an uncertain event occurs which is determined by a random variable $\tilde{e}_n$ whose outcome e_n is *not* under the control of the decision maker. The stage return function may depend on this random variable, that is,

$$f_n(d_n, s_n, \tilde{e}_n),$$

while the state of the system s_{n-1} with $(n-1)$ stages to go invariably will depend on the random variable by

$$\tilde{s}_{n-1} = t_n(d_n, s_n, \tilde{e}_n).$$

The outcomes of the random variable are governed by a probability distribution, $p_n(e_n|d_n, s_n)$, which may be the same for every stage or may be conditional on the stage, the state at the current stage, and even the decision at the current stage.

Figure 11.15 depicts dynamic programming under uncertainty as a *decision tree*, where squares represent states where decisions have to be made and circles represent uncertain events whose outcomes are not under the control of the decision maker. These diagrams can be quite useful in analyzing decisions under uncertainty if the number of possible states is not too large. The decision tree provides a pictorial representation of the sequence of decisions, outcomes, and resulting states, *in the order in which* the decisions must be made and the outcomes become known to the decision maker. Unlike deterministic dynamic programming wherein the

optimal decisions at each stage can be specified at the outset, in dynamic programming under uncertainty, the optimal decision at each stage can be selected only after we know the outcome of the uncertain event at the previous stage. At the outset, all that can be specified is a set of decisions that would be made *contingent* on the outcome of a sequence of uncertain events.

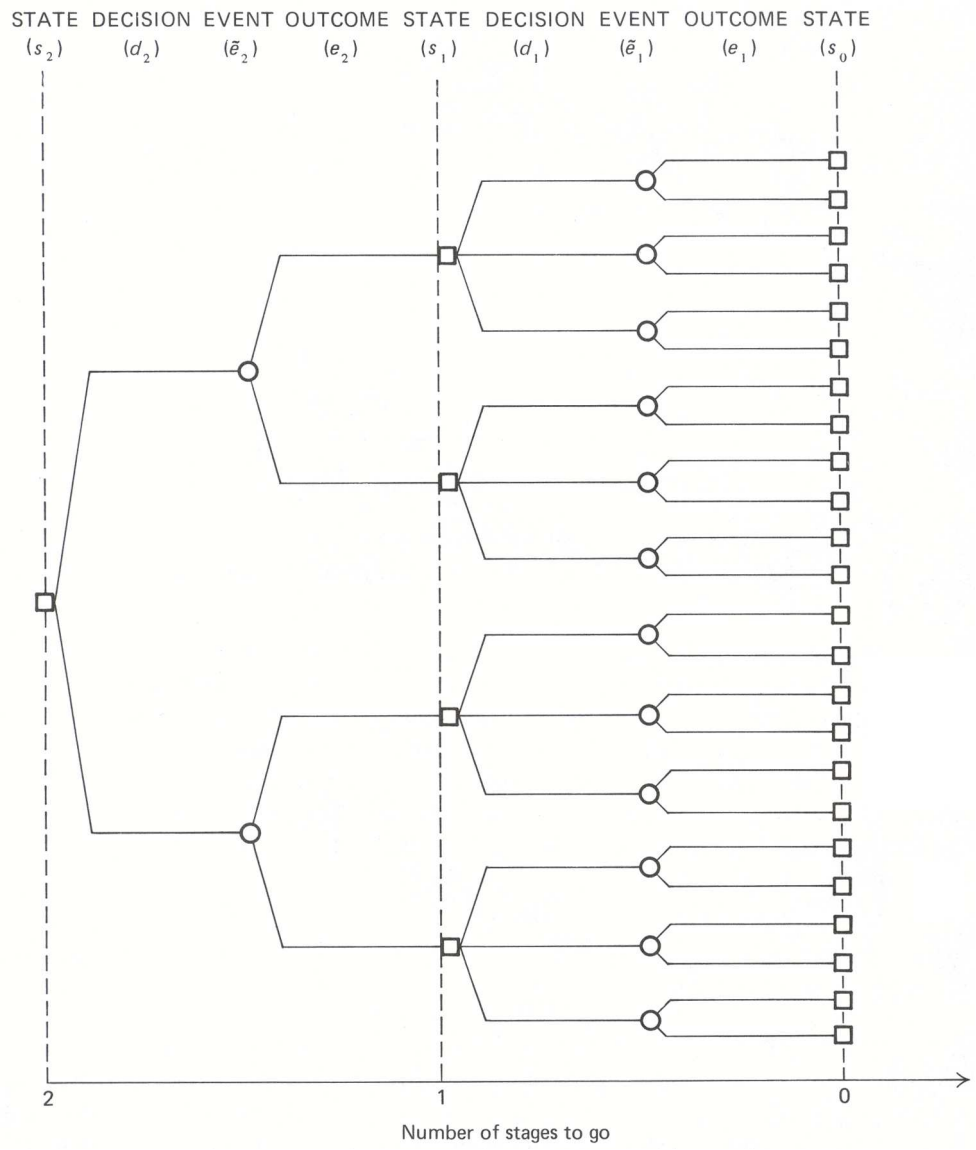


Figure 11.15 Decision tree for dynamic programming under uncertainty.

In dynamic programming under uncertainty, since the stage returns and resulting stage may both be uncertain at each stage, we cannot simply optimize the sum of the stage-return functions. Rather, we must optimize the *expected return* over the stages of the problem, taking into account the sequence in which decisions can be made and the outcomes of uncertain events become known to the decision maker. In this situation, backward induction can be applied to determine the optimal strategy, but forward induction cannot. The difficulty with forward induction is that it is impossible to assign values to states at the next stage that are independent of the uncertain evolution of the process from that future state on. With backward induction, on the other hand, no such difficulties arise since the states with zero stages to go are evaluated first, and then the states with one stage to go are evaluated by computing the expected value of any decision and choosing

optimally.

We start the backward induction process by computing the optimal-value function at stage 0. This amounts to determining the value of ending in each possible stage with 0 stages to go. This determination may involve an optimization problem or the value of the assets held at the horizon. Next, we compute the optimal-value function at the previous stage. To do this, we first compute the expected value of each uncertain event, weighting the stage return plus the value of the resulting state for each outcome by the probability of each outcome. Then, for each state at the previous stage, we select the decision that has the maximum (or minimum) expected value. Once the optimal-value function for stage 1 has been determined, we continue in a similar manner to determine the optimal-value functions at prior stages by backward induction.

The optimal-value function for dynamic programming under uncertainty is then defined in the following recursive form:

$$v_n(s_n) = \text{Max } E\left[f_n(d_n, s_n, \tilde{e}_n) + v_{n-1}(\tilde{s}_{n-1}) \right], \tag{17}$$

subject to:

$$\tilde{s}_{n-1} = t_n(d_n, s_n, \tilde{e}_n),$$
$$d_n \in D_n,$$

where $E[\cdot]$ denotes the expected value of the quantity in brackets. To initiate the recursive calculations we need to determine the optimal-value function with zero stages to go, which is given by:

$$v_0(s_0) = \text{Max } E\left[f_0(d_0, s_0, \tilde{e}_0) \right],$$

subject to:

$$d_0 \in D_0.$$

The optimization problems that determine the optimal-value function with zero stages to go are not determined recursively, and therefore may be solved in a straight-forward manner. If the objective function is to maximize the expected discounted costs, then Eq. (17) is modified as in Section 11.4 by multiplying the term $v_{n-1}(\tilde{s}_{n-1})$ by β_n, the discount factor for period n.

We can make these ideas more concrete by considering a simple example. A manager is in charge of the replenishment decisions during the next two months for the inventory of a fairly expensive item. The production cost of the item is \$1000/unit, and its selling price is \$2000/unit. There is an inventory-carrying cost of \$100/unit per month on each unit left over at the end of the month. We assume there is no setup cost associated with running a production order, and further that the production process has a short lead time; therefore any amount produced during a given month is available to satisfy the demand during that month. At the present time, there is no inventory on hand. Any inventory left at the end of the next two months has to be disposed of at a salvage value of \$500/unit.

The demand for the item is uncertain, but its probability distribution is identical for each of the coming two months. The probability distribution of the demand is as follows:

Demand	*Probability*
0	0.25
1	0.40
2	0.20
3	0.15

The issue to be resolved is how many units to produce during the first month and, *depending on the actual demand in the first month*, how many units to produce during the second month. Since demand is uncertain, the inventory at the end of each month is also uncertain. In fact, demand could exceed the available units on hand in any month, in which case all excess demand results in lost sales. Consequently, our production decision must find the proper balance between production costs, lost sales, and final inventory salvage value.

The states for this type of problem are usually represented by the inventory level I_n at the beginning of each month. Moreover, the problem is characterized as a two-stage problem, since there are two months involved in the inventory-replenishment decision. To determine the optimal-value function, let

$v_n(I_n) =$ Maximum contribution, given that we have I_n units of inventory with n stages to go.

We initiate the backward induction procedure by determining the optimal-value function with 0 stages to go. Since the salvagevalue is \$500/unit, we have:

I_0	$v_0(I_0)$
0	0
1	500
2	1000
3	1500

To compute the optimal-value function with one stage to go, we need to determine, for each inventory level (state), the corresponding contribution associated with each possible production amount (decision) and level of sales (outcome). For each inventory level, we select the production amount that maximizes the expected contribution.

Table 11.3 provides all the necessary detailed computations to determine the optimal-value function with one stage to go. Column 1 gives the state (inventory level) of the process with one stage to go. Column 2 gives the possible decisions (amount to produce) for each state, and, since demand cannot be greater than three, the amount produced is at most three. Column 3 gives the possible outcomes for the uncertain level of sales for each decision and current state, and column 4 gives the probability of each of these possible outcomes. Note that, in any period, it is impossible to sell more than the supply, which is the sum of the inventory currently on hand plus the amount produced. Hence, the probability distribution of sales differs from that of demand since, whenever demand exceeds supply, the entire supply is sold and the excess demand is lost. Column 5 is the resulting state, given that we currently have I_1 on hand, produce d_1, and sell s_1. The transition function in general is just:

$$\tilde{I}_{n-1} = I_n + d_n - \tilde{s}_n,$$

where the tildes ($\sim$) indicate that the level of sales is uncertain and, hence, the resulting state is also uncertain. Columns 6, 7, and 8 reflect the revenue and costs for each state, decision, and sales level, and column 9 reflects the value of being in the resulting state at the next stage. Column 10 merely weights the sum of columns 6 through 9 by the probability of their occurring, which is an intermediate calculation in determining the expected value of making a particular decision, given the current state. Column 11 is then just this expected value; and the asterisk indicates the optimal decision for each possible state.

Table 11.3 Computation of optimal-value function with one stage to go.

(1) *State* I_1	*(2)* *Produce* d_1	*(3)* *Sell* s_1	*(4)* *Probability* $(\tilde{s}_1 = s_1)$	*(5)* *Resulting state* $\tilde{I}_0$	*(6)* *Production cost*	*(7)* *Sales revenue*	*(8)* *Inventory cost*	*(9)* $v_0(I_0)$	*(10)* *Probability* $\times$ \$	*(11)* *Expected contribution*
0	0	0	1.	0	0	0	0	0	0	0
	1	0	.25	1	−1000	0	−100	500	−150	600*
		1	.75	0	−1000	2000	0	0	750	
	2	0	.25	2	−2000	0	−200	1000	−300	
		1	.40	1	−2000	2000	−100	500	160	560
		2	.35	0	−2000	4000	0	0	700	
	3	0	.25	3	−3000	0	−300	1500	−450	
		1	.40	2	−3000	2000	−200	1000	−80	
		2	.20	1	−3000	4000	−100	500	280	200
		3	.15	0	−3000	6000	0	0	450	
1	0	0	.25	1	0	0	−100	500	100	1600*
		1	.75	0	0	2000	0	0	1500	
	1	0	.25	2	−1000	0	−200	1000	−50	
		1	.40	1	−1000	2000	−100	500	560	1560
		2	.35	0	−1000	4000	0	0	1050	
	2	0	.25	3	−2000	0	−300	1500	−200	
		1	.40	2	−2000	2000	−200	1000	320	
		2	.20	1	−2000	4000	−100	500	480	1200
		3	.15	0	−2000	6000	0	0	600	
2	0	0	.25	2	0	0	−200	1000	200	
		1	.40	1	0	2000	−100	500	960	2560*
		2	.35	0	0	4000	0	0	1400	
	1	0	.25	3	−1000	0	−300	1500	50	
		1	.40	2	−1000	2000	−200	1000	720	
		2	.20	1	−1000	4000	−100	500	680	2200
		3	.15	0	−1000	6000	0	0	750	
3	0	0	.25	3	0	0	−300	1500	300	
		1	.40	2	0	2000	−200	1000	1120	
		2	.20	1	0	4000	−100	500	880	3200*
		3	.15	0	0	6000	0	0	900	

The resulting optimal-value function and the corresponding optimal-decision function are determined directly from Table 11.3 and are the following:

I_1	$v_1(I_1)$	$d_1^*(I_1)$
0	600	1
1	1600	0
2	2560	0
3	3200	0

Next we need to compute the optimal-value function with two stagesto go. However, since we have assumed that there is no initial inventory on hand, it is not necessary to describe the optimal-value function for every possible state, but only for $I_2 = 0$. Table 11.4 is similar to Table 11.3 and gives the detailed computations required to evaluate the optimal-value function for this case.

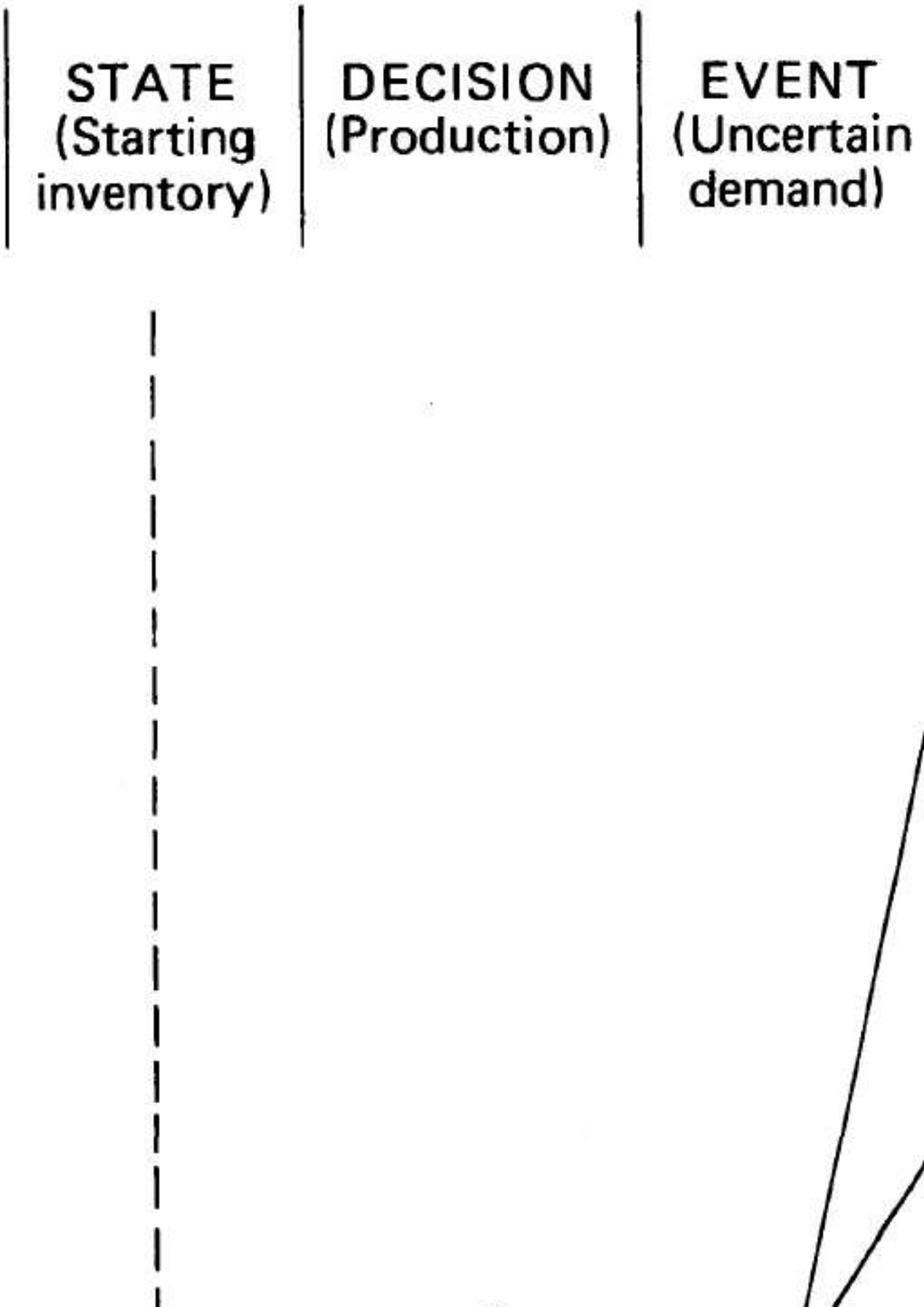
Table 11.4 Computation of optimal-value function with two stages to go, $I_2 = 0$ only.

(1) *State* I_2	(2) *Produce* d_2	(3) *Sell* s_2	(4) *Probability* $(\tilde{s}_2 = s_2)$	(5) *Resulting state* $\tilde{I}_1$	(6) *Production cost*	(7) *Sales revenue*	(8) *Inventory cost*	(9) $v_1(I_1)$	(10) *Probability* $\times$ \$	(11) *Expected contribution*
0	0	0	1.	0	0	0	0	650.	650	650
	1	0	.25	1	−1000	0	0	1600.	150	1350
		1	.75	0	−1000	2000	0	600.	1200	
	2	0	.25	2	−2000	0	−200	2560.	90	1600.*
		1	.40	1	−2000	2000	−100	1600.	600	
		2	.35	0	−2000	4000	0	600	910	
	3	0	.25	3	−3000	0	−300	3200.	−25	1559
		1	.40	2	−3000	2000	−200	2560.	544	
		2	.20	1	−3000	4000	−100	1600.	500	
		3	.15	0	−3000	6000	0	600.	540	

The optimal-value function and the corresponding decision function for $I_2 = 0$ are taken directly from Table 11.4 and are the following:

I_2	$v_2(I_2)$	$d_2^*(I_2)$
0	1600	2

The optimal strategy can be summarized by the decision tree given in Fig. 11.16. The expected contribution determined by the dynamic-programming solution corresponds to weighting the contribution of every path in this tree by the *probability* that this path occurs. The decision tree in Fig. 11.16 emphasizes the contingent nature of the optimal strategy determined by dynamic programming under uncertainty.


STATE
(Starting
inventory)
DECISION
(Production)
EVENT
(Uncertain
demand)

EXERCISES

Solutions to exercises marked with an asterisk (*) involve extensive computations. Formulate these problems as dynamic programs and provide representative computations to indicate the nature of the dynamic programming recursions; solve to completion only if a computer system is available.

1. In solving the minimum-delay routing problem in Section 11.1, we assumed the same delay along each street (arc) in the network. Suppose, instead, that the delay when moving along any arc upward in the network is 2 units greater than the delay when moving along any arc downward. The delay at the intersections is still given by the data in Fig. 11.1. Solve for the minimum-delay route by both forward and backward induction.

2. Decatron Mills has contracted to deliver 20 tons of a special coarsely ground wheat flour at the end of the current month, and 140 tons at the end of the next month. The production cost, based on which the Sales Department has bargained with prospective customers, is $c_1(x_1) = 7500 + (x_1 - 50)^2$ per ton for the first month, and $c_2(x_2) = 7500 + (x_2 - 40)^2$ per ton for the second month; x_1 and x_2 are the number of tons of the flour produced in the first and second months, respectively. If the company chooses to produce more than 20 tons in the first month, any excess production can be carried to the second month at a storage cost of \$3 per ton.

   Assuming that there is no initial inventory and that the contracted demands must be satisfied in each month (that is, no back-ordering is allowed), derive the production plan that minimizes total cost. Solve by both backward and forward induction. Consider x_1 and x_2 as continuous variables, since any fraction of a ton may be produced in either month.

3. A construction company has four projects in progress. According to the current allocation of manpower, equipment, and materials, the four projects can be completed in 15, 20, 18, and 25 weeks. Management wants to reduce the completion times and has decided to allocate an additional \$35,000 to all four projects. The new completion times as functions of the additional funds allocated to each projects are given in Table E11.1.

   How should the \$35,000 be allocated among the projects to achieve the largest total reduction in completion times? Assume that the additional funds can be allocated only in blocks of \$5000.

Table E11.1 Completion times (in weeks)

Additional funds (×1000 dollars)	*Project 1*	*Project 2*	*Project 3*	*Project 4*
0	15	20	18	25
5	12	16	15	21
10	10	13	12	18
15	8	11	10	16
20	7	9	9	14
25	6	8	8	12
30	5	7	7	11
35	4	7	6	10

4. The following table specifies the unit weights and values of five products held in storage. The quantity of each item is unlimited.

Product	*Weight* (W_i)	*Value* (V_i)
1	7	9
2	5	4
3	4	3
4	3	2
5	1	$\frac{1}{2}$

A plane with a capacity of 13 weight units is to be used to transport the products. How should the plane be loaded to maximize the value of goods shipped? (Formulate the problem as an integer program and solve by dynamic programming.)

5. Any linear-programming problem with n decision variables and m constraints can be converted into an n-stage dynamic-programming problem with m state parameters.

   Set up a dynamic-programming formulation for the following linear program:

$$\text{Minimize} \sum_{j=1}^{n} c_j x_j,$$

   subject to:

$$\sum_{j=1}^{n} a_{ij} x_j \leq b_i \qquad (i = 1, 2, \ldots, m),$$

$$x_j \geq 0 \qquad (j = 1, 2, \ldots, n).$$

   Why is it generally true that the simplex method rather than dynamic programming is recommended for solving linear programs?

6. Rambling Roger, a veteran of the hitchhiking corps, has decided to leave the cold of a Boston winter and head for the sunshine of Miami. His vast experience has given him an indication of the expected time in hours it takes to hitchhike over certain segments of the highways. Knowing he will be breaking the law in several states and wishing to reach the warm weather quickly, Roger wants to know the least-time route to take. He summarized his expected travel times on the map in Fig. E11.1. Find his shortest time route.

7. J. J. Jefferson has decided to move from the West Coast, where he lives, to a mid-western town, where he intends to buy a small farm and lead a quiet life. Since J. J. is single and has accumulated little furniture, he decides to rent a small truck for $200 a week or fraction of a week (one-way, no mileage charge) and move his belongings by himself. Studying the map, he figures that his trip will require four stages, regardless of the particular routing. Each node shown in Fig. E11.2 corresponds to a town where J. J. has either friends or relatives and where he plans to spend one day resting and visiting if he travels through the town. The numbers in brackets in Fig. E11.2 specify the travel time in days between nodes. (The position of each node in the network is not necessarily related to its geographical position on the map.) As he will travel through different states, motel rates, tolls, and gas prices vary significantly; Fig. E11.2 also shows the cost in dollars for traveling (excluding truck rental charges) between every two nodes. Find J. J.'s cheapest route between towns 1 and 10, including the truck rental charges.

8. At THE CASINO in Las Vegas, a customer can bet only in dollar increments. Betting a certain amount is called ''playing a round.'' Associated with each dollar bet on a round, the customer has a 40% chance to win another dollar and a 60% chance to lose his, or her, dollar. If the customer starts with $4 and wants to maximize the chances of finishing with at least $7 after two rounds, how much should be bet on each round? [*Hint*. Consider the number of dollars available at the beginning of each round as the state variable.]

*9. In a youth contest, Joe will shoot a total of ten shots at four different targets. The contest has been designed so that Joe will not know whether or not he hits any target until after he has made all ten shots. He obtains 6 points if any shot hits target 1, 4 points for hitting target 2, 10 points for hitting target 3, and 7 points for hitting target 4. At each shot there is an 80% chance that he will miss target 1, a 60% chance of missing target 2, a 90% chance of missing target 3, and a 50% chance of missing target 4, given that he aims at the appropriate target.

   If Joe wants to maximize his expected number of points, how many shots should he aim at each target?

10. A monitoring device is assembled from five different components. Proper functioning of the device depends upon its total weight q so that, among other tests, the device is weighted; it is accepted only if $r_1 \leq q \leq r_2$, where the two limits r_1 and r_2 have been prespecified.

    The weight $q_j (j = 1, 2, \ldots, 5)$ of each component varies somewhat from unit to unit in accordance with a normal distribution with mean μ_j and variance σ_j^2. As $q_1, q_2, \ldots, q_5$ are independent, the total weight q will also be a normal variable with mean $\mu = \sum_{j=1}^{5} \mu_j$ and variance $\sigma^2 = \sum_{j=1}^{5} \sigma_j^2$.

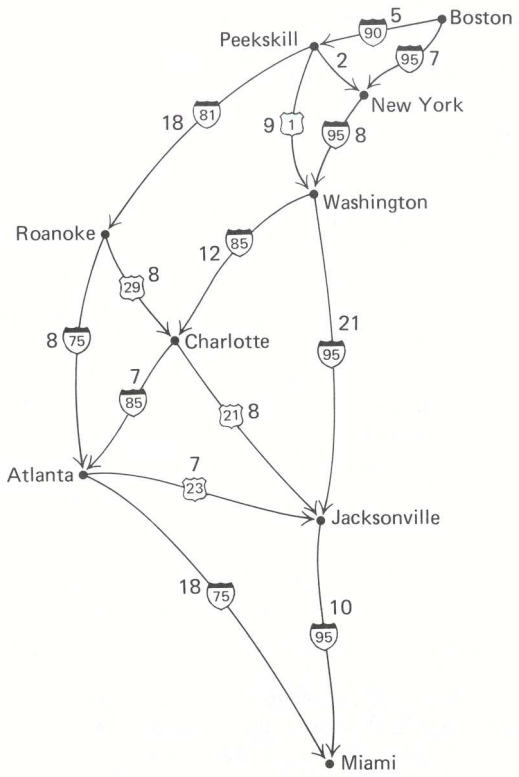


Figure E11.1 Travel times to highways.

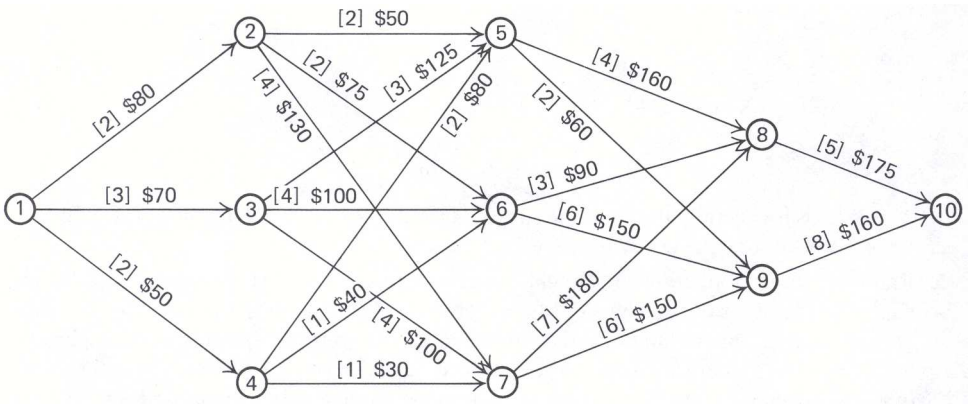


Figure E11.2 Routing times and costs.

Clearly, even if μ can be adjusted to fall within the interval $[r_1, r_2]$, the rejection rate will depend upon σ^2; in this case, the rejection rate can be made as small as desired by making the variance σ^2 sufficiently small. The design department has decided that $\sigma^2 = 5$ is the largest variance that would make the rejection rate of the monitoring device acceptable. The cost of manufacturing component j is $c_j = 1/\sigma_j^2$.

Determine values for the design parameters σ_j^2 for $j = 1, 2, \ldots, 5$ that would minimize the manufacturing cost of the components while ensuring an acceptable rejection rate. [*Hint*. Each component is a stage; the state variable is that portion of the total variance σ^2 not yet distributed. Consider σ_j^2's as continuous variables.]

*11. A scientific expedition to Death Valley is being organized. In addition to the scientific equipment, the expedition also has to carry a stock of spare parts, which are likely to fail under the extreme heat conditions prevailing in that area. The estimated number of times that the six critical parts, those sensitive to the heat conditions, will fail during the expedition are shown below in the form of probability distributions.

Part 1

# of Failures	*Probability*
0	0.5
1	0.3
2	0.2

Part 2

# of Failures	*Probability*
0	0.4
1	0.3
2	0.2
3	0.1

Part 3

# of Failures	*Probability*
0	0.7
1	0.2
2	0.1

Part 4

# of Failures	*Probability*
0	0.9
1	0.1

Part 5

# of Failures	*Probability*
0	0.8
1	0.1
2	0.1

Part 6

# of Failures	*Probability*
0	0.8
1	0.2

The spare-park kit should not weight more than 30 pounds. If one part is needed and it is not available in the spare-park kit, it may be ordered by radio and shipped by helicopter at unit costs as specified in Table E11.2, which also gives the weight of each part.

Table E11.2 Spare-Part Data

Part	*Weight* (pounds/unit)	*Shipping cost* ($/unit)
1	4	100
2	3	70
3	2	90
4	5	80
5	3	60
6	2	50

Determine the composition of the spare-park kit to minimize total expected ordering costs.

*12. After a hard day at work I frequently wish to return home as quickly as possible. I must choose from several alternate routes (see Fig. E11.3); the travel time on any road is uncertain and depends upon the congestion at the nearest major

Table E11.3 Travel time on the road

Road i–j	*Congestion at initial intersection* (*i*)	*Travel-time distribution*	
		Travel time (minutes)	*Probability*
5–4	Heavy	4	$\frac{1}{4}$
		6	$\frac{1}{2}$
		10	$\frac{1}{4}$
	Light	2	$\frac{1}{3}$
		3	$\frac{1}{3}$
		5	$\frac{1}{3}$
5–3	Heavy	5	$\frac{1}{2}$
		12	$\frac{1}{2}$
	Light	3	$\frac{1}{2}$
		6	$\frac{1}{2}$
4–2	Heavy	7	$\frac{1}{3}$
		14	$\frac{2}{3}$
	Light	4	$\frac{1}{2}$
		6	$\frac{1}{2}$
3–2	Heavy	5	$\frac{1}{4}$
		11	$\frac{3}{4}$
	Light	3	$\frac{1}{3}$
		5	$\frac{1}{3}$
		7	$\frac{1}{3}$
3–1	Heavy	3	$\frac{1}{2}$
		5	$\frac{1}{2}$
	Light	2	$\frac{1}{2}$
		3	$\frac{1}{2}$
2–1	Heavy	2	$\frac{1}{2}$
		4	$\frac{1}{2}$
	Light	1	$\frac{1}{2}$
		2	$\frac{1}{2}$

intersection preceding that route. Using the data in Table E11.3, determine my best route, given that the congestion at my starting point is heavy.

Assume that if I am at intersection i with heavy congestion and I take road *i–j*, then

$$\text{Prob (intersection } j \text{ is heavy)} = 0.8.$$

If the congestion is light at intersection i and I take road *i–j*, then

$$\text{Prob (intersection } j \text{ is heavy)} = 0.3.$$

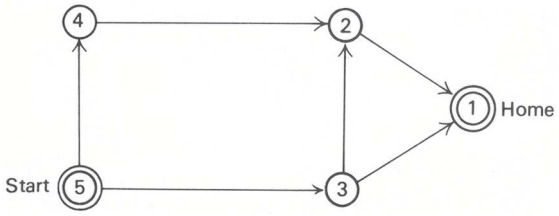


Figure E11.3

13. Find the shortest path from node 1 to every other node in the network given in Fig. E11.4, using the shortest-route algorithm for acyclic networks. The number next to each arc is the ''length'' of that arc.

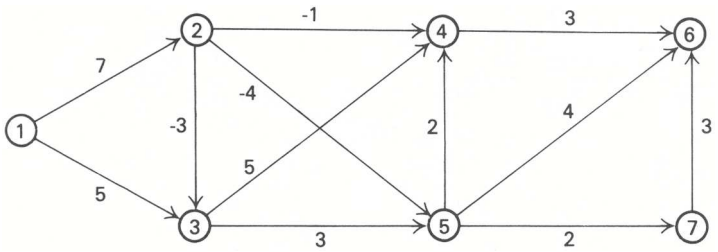


Figure E11.4

14. Find the shortest path from node 1 to every other node in the network given in Fig. E11.5.

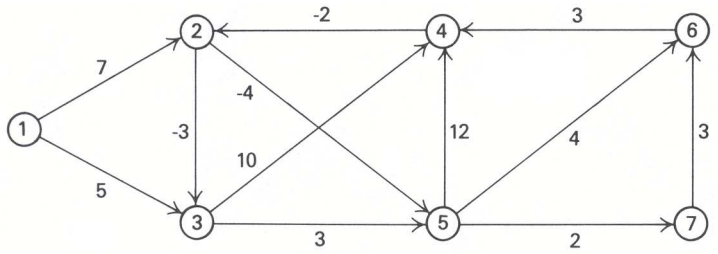


Figure E11.5

15. a) Give a dynamic-programming recursion for finding the shortest path from every node to a particular node, node k, in a network without negative cycles.
    b) Apply the recursion from part (a) to find the shortest path from every node to node 6 in the network specified in the previous exercise.

16. A state's legislature has R representatives. The state is sectioned into s districts, where District j has a population p_j and $s < R$. Under strictly proportional representation. District j would receive $Rp_j/(\sum_{j=1}^{s} p_j) = r_j$ representatives; this allocation is not feasible, however, because r_j may not be integer-valued. The objective is to allocate y_j representatives to District j for $j = 1, 2, \ldots, s$, so as to minimize, over all the districts, the maximum difference between y_j and r_j; that is, minimize [maximum $(|y_1 - r_1|, |y_2 - r_2|, \ldots, |y_s - r_s|)$].

    a) Formulate the model in terms of a dynamic-programming recursion.
    b) Apply your method to the data $R = 4$, $s = 3$, and $r_1 = 0.4$, $r_2 = 2.4$, and $r_3 = 1.2$.
    c) Discuss whether the solution seems reasonable, given the context of the problem.

17. In a textile plant, cloth is manufactured in rolls of length L. Defects sometimes occur along the length of the cloth. Consider a specific roll with $(N - 1)$ defects appearing at distances $y_1, y_2, \ldots, y_{N-1}$ from the start of the roll ($y_{i+1} > y_i$ for all i). Denote the start of the roll by y_0, the end by y_N.

The roll is cut into pieces for sale. The value of a piece depends on its length and the number of defects. Let

$$v(x, m) = \text{Value of a piece of length } x \text{ having } m \text{ defects.}$$

Assume that all cuts are made *through* defects and that such cutting removes the defect.

Specify how to determine where to cut the cloth to maximize total value.

18. A manufacturing company, receiving an order for a special product, has worked out a production plan for the next 5 months. All components will be manufactured internally except for one electronic part that must be purchased. The purchasing manager in charge of buying the electronic part must meet the requirements schedule established by the production department. After negotiating with several suppliers, the purchasing manager has determined the best possible price for the electronic part for each of the five months in the planning horizon. Table E11.4 summarizes the requirement schedule and purchase price information.

Table E11.4 Requirements schedule and purchasing prices

Month	*Requirements* (thousands)	*Purchasing price* ($/thousand pieces)
1	5	10
2	10	11
3	6	13
4	9	10
5	4	12

The storage capacity for this item is limited to 12,000 units; there is no initial stock, and after the five-month period the item will no longer be needed. Assume that the orders for the electronic part are placed once every month (at the beginning of each month) and that the delivery lead time is very short (delivery is made practically instantaneously). No back-ordering is permitted.

a) Derive the monthly purchasing schedule if total purchasing cost is to be minimized.
b) Assume that a storage charge of $250 is incurred for each 1000 units found in inventory at the end of a month. What purchasing schedule would minimize the purchasing and storage costs?

Table E11.5 Profits in response to advertising

Additional investment in advertising (in $100,000)	*Profits* (in $100,000)				
	Product A	*Product* B	*Product* C	*Product* D	*Product* E
0	0	0	0	0	0
1	0.20	0.18	0.23	0.15	0.25
2	0.35	0.30	0.43	0.30	0.45
3	0.50	0.42	0.60	0.45	0.65
4	0.63	0.54	0.75	0.58	0.80
5	0.75	0.64	0.80	0.70	0.90
6	0.83	0.74	0.92	0.81	0.95
7	0.90	0.84	0.98	0.91	0.98
8	0.95	0.92	1.02	1.00	1.01
9	0.98	1.00	1.05	1.04	1.02
10	0.98	1.05	1.06	1.07	1.03

19. Rebron, Inc., a cosmetics manufacturer, markets five different skin lotions and creams: A, B, C, D, E. The company has decided to increase the advertising budget allocated to this group of products by 1 million dollars for next year. The marketing department has conducted a research program to establish how advertising affects the sales levels

of these products. Table E11.5 shows the increase in each product's contribution to net profits as a function of the additional advertisement expenditures.

Given that maximization of net profits is sought, what is the optimal allocation of the additional advertising budget among the five products? Assume, for simplicity, that advertising funds must be allocated in blocks of $100,000.

*20. A machine tool manufacturer is planning an expansion program. Up to 10 workers can be hired and assigned to the five divisions of the company. Since the manufacturer is currently operating with idle machine capacity, no new equipment has to be purchased.

Hiring new workers adds $250/day to the indirect costs of the company. On the other hand, new workers add value to the company's output (i.e., sales revenues in excess of direct costs) as indicated in Table E11.6. Note that the value added depends upon both the number of workers hired and the division to which they are assigned.

Table E11.6 Value added by new workers

New workers (x_n)	*Increase in contribution to overhead* ($/day)				
	Division 1	*Division 2*	*Division 3*	*Division 4*	*Division 5*
0	0	0	0	0	0
1	30	25	35	32	28
2	55	50	65	60	53
3	78	72	90	88	73
4	97	90	110	113	91
5	115	108	120	133	109
6	131	124	128	146	127
7	144	138	135	153	145
8	154	140	140	158	160
9	160	150	144	161	170
10	163	154	145	162	172

The company wishes to hire workers so that the value that they add exceeds the $250/day in indirect costs. What is the minimum number of workers the company should hire and how should they be allocated among the five divisions?

21. A retailer wishes to plan the purchase of a certain item for the next five months. Suppose that the demand in these months is known and given by:

Month	*Demand* (units)
1	10
2	20
3	30
4	30
5	20

The retailer orders at the beginning of each month. Initially he has no units of the item. Any units left at the end of a month will be transferred to the next month, but at a cost of 10¢ per unit. It costs $20 to place an order. Assume that the retailer can order only in lots of 10, 20, ... units and that the maximum amount he can order each month is 60 units. Further assume that he receives the order immediately (no lead time) and that the demand occurs just after he receives the order. He attempts to stock whatever remains but cannot stock more than 40 units—units in excess of 40 are discarded at no additional cost and with no salvage value. How many units should the retailer order each month?

*22. Suppose that the retailer of the previous exercise does not know demand with certainty. All assumptions are as in Exercise 21 except as noted below. The demand for the item is the same for each month and is given by the following distribution:

Demand	*Probability*
10	0.2
30	0.5
30	0.3

Each unit costs \$1. Each unit demanded in excess of the units on hand is lost, with a penalty of \$2 per unit. How many units should be ordered each month to minimize total expected costs over the planning horizon? Outline a dynamic-programming formulation and complete the calculations for the last two stages only.

*23. The owner of a hardware store is surprised to find that he is completely out of stock of ''Safe-t-lock,'' an extremely popular hardened-steel safety lock for bicycles. Fortunately, he became aware of this situation before anybody asked for one of the locks; otherwise he would have lost \$2 in profits for each unit demanded but not available. He decides to use his pickup truck and immediately obtain some of the locks from a nearby warehouse.

Although the demand for locks is uncertain, the probability distribution for demand is known; it is the same in each month and is given by:

Demand	*Probability*
0	0.1
100	0.3
200	0.4
300	0.2

The storage capacity is 400 units, and the carrying cost is \$1 per unit per month, charged to the month's *average inventory* [i.e., (initial + ending)/2]. Assume that the withdrawal rate is uniform over the month. The lock is replenished monthly, at the beginning of the month, in lots of one hundred.

What is the replenishment strategy that minimizes the expected costs (storage and shortage costs) over a planning horizon of four months? No specific inventory level is required for the end of the planning horizon.

*24. In anticipation of the Olympic games, Julius, a famous Danish pastry cook, has opened a coffee-and-pastry shop not far from the Olympic Village. He has signed a contract to sell the shop for \$50,000 after operating it for 5 months.

Julius has several secret recipes that have proved very popular with consumers during the last Olympic season, but now that the games are to be held on another continent, variations in tastes and habits cast a note of uncertainty over his chances of renewed success.

The pastry cook plans to sell all types of common pastries and to use his specialties to attract large crowds to his shop. He realizes that the popularity of his shop will depend upon how well his specialties are received; consequently, he may alter the offerings of these pastries from month to month when he feels that he can improve business. When his shop is not popular, he may determine what type of specialties to offer by running two-day market surveys. Additionally, Julius can advertise in the *Olympic Herald Daily* and other local newspapers to attract new customers.

The shop's popularity may change from month to month. These transitions are uncertain and depend upon advertising and market-survey strategies. Table E11.7 summarizes the various possibilities. The profit figures in this table include advertising expenditures and market-survey costs.

Note that Julius has decided either to advertise or to run the market survey whenever the shop is not popular.

Assume that, during his first month of operation, Julius passively waits to see how popular his shop will be. What is the optimal strategy for him to follow in succeeding months to maximize his expected profits?

25. A particular city contains six significant points of interest. Figure E11.6 depicts the network of major two-way avenues connecting the points; the figure also shows travel time (in both directions) along each avenue. Other streets, having travel times exceeding those along the major avenues, link the points but have been dropped from this analysis.

In an attempt to reduce congestion of traffic in the city, the city council is considering converting some two-way avenues to one-way avenues.

The city council is considering two alternative planning objectives:

Table E11.7 Profit possibilities (p = probability; E = expected profit)

	Popular next month	*Not popular next month*
Popular this month, no advertising	$p = \frac{6}{10}$, $E = \$6000$	$p = \frac{4}{10}$, $E = \$2000$
Popular this month, advertising	$p = \frac{3}{4}$, $E = \$4000$	$p = \frac{1}{10}$, $E = \$3000$
Not popular this month, market survey	$p = \frac{1}{3}$, $E = \$3000$	$p = \frac{2}{3}$, $E = -\$2000$
Not popular this month, advertising	$p = \frac{6}{10}$, $E = \$1000$	$p = \frac{4}{10}$, $E = -\$5000$

a) Given that point (1) is the tourist information center for the city, from which most visitors depart, which avenues should be made one-way so as to minimize the travel times from point (1) to every other point?
b) If the travel times from each point to every other point were to be minimized, which avenues would be converted to one-way?

In both cases, assume that the travel times shown in Fig E11.6 would not be affected by the conversion. If the total conversion cost is proportional to the number of avenues converted to one-way, which of the above solutions has the lowest cost?

*26. Consider the following one-period problem: a certain item is produced centrally in a factory and distributed to four warehouses. The factory can produce up to 12 thousand pieces of the item. The transportation cost from the factory to warehouse n is t_n dollars per thousand pieces.

From historical data, it is known that the demand per period from warehouse n for the item is governed by a Poisson distribution† with mean λ_n (in thousands of pieces). If demand exceeds available stock a penalty of π_n dollars per thousand units out of stock is charged at warehouse n.

The current inventory on hand at warehouse n is q_n thousand units.

a) Formulate a dynamic program for determining the amount to be produced and the optimal allocation to each warehouse, in order to minimize transportation and expected stockout costs.
b) Solve the problem for a four-warehouse system with the data given in Table E11.8.

† The Poisson distribution is given by Prob. $(\tilde{k} = k) = \frac{\lambda^k e^{-\lambda}}{k!}$.

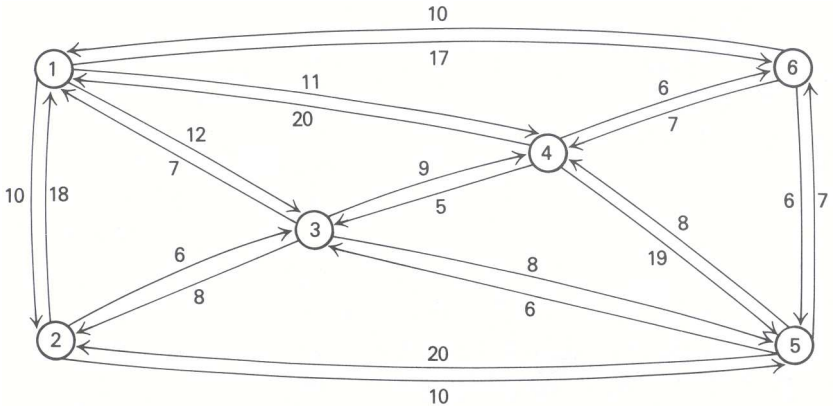


Figure E11.6

Table E11.8

Warehouse (n)	*Demand* λ_n (thousand units)	*Inventory* q_n (thousand units)	*Transportation cost* t_n ($ per 1000 units)	*Stockout penalty* π_n ($ per 1000 units)
1	3	1	100	1500
2	4	2	300	2000
3	5	1	250	1700
4	2	0	200	2200

*27. Precision Equipment, Inc., has won a government contract to supply 4 pieces of a high precision part that is used in the fuel throttle-valve of an orbital module. The factory has three different machines capable of producing the item. They differ in terms of setup cost, variable production cost, and the chance that every single item will meet the high-quality standards (see Table E11.9).

Table E11.9

Machine	*Setup cost* ($)	*Variable cost* ($/unit)	*Probability of meeting standards*
A	100	20	0.50
B	300	40	0.80
C	500	60	0.90

After the parts are produced, they are sent to the engine assembly plant where they are tested. There is no way to recondition a rejected item. Any parts in excess of four, even if good, must be scrapped. If less than 4 parts are good, the manufacturer has to pay a penalty of $200 for each undelivered item.

How many items should be produced on each machine in order to minimize total expected cost? [*Hint*. Consider each machine as a stage and define the state variable as the number of acceptable parts still to be produced.]

*28. One of the systems of a communications satellite consists of five electronic devices connected in series; the system as a whole would fail if any one of these devices were to fail. A common engineering design to increase the reliability of the system is to connect several devices of the same type in parallel, as shown in Fig E11.7. The parallel devices in each group are controlled by a monitoring system, so that, if one device fails, another one immediately becomes operative.

The total weight of the system may not exceed 20 pounds. Table E11.10 shows the weight in pounds and the probability of failure for each device in group j of the system design. How many devices should be connected in

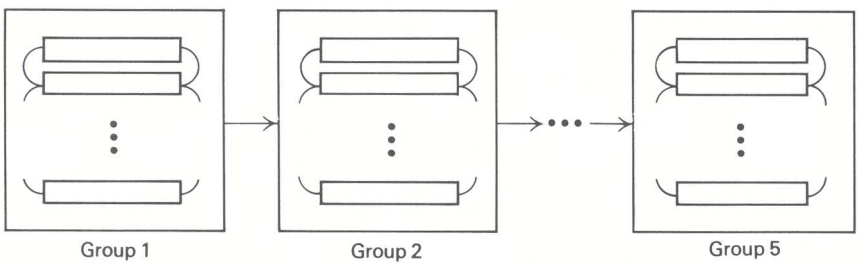


Figure E11.7

parallel in each group so as to maximize the reliability of the overall system?

*29. The production manager of a manufacturing company has to devise a production plan for item AK102 for the

Table E11.10

Group	*Weight* (lbs./device)	*Probability of failure for each device*
1	1	0.20
2	2	0.10
3	1	0.30
4	2	0.15
5	3	0.05

next four months. The item is to be produced at most once monthly; because of capacity limitations the monthly production may not exceed 10 units. The cost of one setup for any positive level of production in any month is \$10.

The demand for this item is uncertain and varies from month to month; from past experience, however, the manager concludes that the demand in each month can be approximated by a Poisson distribution with parameter λ_n (n shows the month to which the distribution refers).

Inventory is counted at the end of each month and a holding cost of \$10 is charged for each unit; if there are stockouts, a penalty of \$20 is charged for every unit out of stock. There is no initial inventory and no outstanding back-orders; no inventory is required at the end of the planning period. Assume that the production lead time is short so that the amount released for production in one month can be used to satisfy demand within the same month.

What is the optimal production plan, assuming that the optimality criterion is the minimum expected cost? Assume that $\lambda_1 = 3$, $\lambda_2 = 5$, $\lambda_3 = 2$, $\lambda_4 = 4$ units.

*30. Just before the Christmas season, Bribham of New England, Inc., has signed a large contract to buy four varieties of Swiss chocolate from a local importer. As it was already late, the distributor could arrange for only a limited transportation of 20 tons of Swiss chocolate to be delivered in time for Christmas.

Chocolate is transported in containers; the weight and the transportation cost per container are given in Table E11.11.

Table E11.11

Variety	*Weight* (tons/container)	*Transportation* (\$/container)	*Shortage cost* (\$/container)	λ_n (tons)
1	2	50	500	3
2	3	100	300	4
3	4	150	800	2
4	4	200	1000	1

A marketing consulting firm has conducted a study and has estimated the demand for the upcoming holiday season as a Poisson distribution with parameter $\lambda_n (n = 1, 2, 3, 4$ indicates the variety of the chocolate). Bribham loses contribution (i.e., shortage cost) for each container that can be sold (i.e., is demanded) but is not available.

How many containers of each variety should the company make available for Christmas in order to minimize total expected cost (transportation and shortage costs)?

ACKNOWLEDGMENTS

The example in Section 11.6 is taken from the State Department of Public Health case by Richard F. Meyer. Exercise 10 is based on Section 10.6 of *Nonlinear and Dynamic Programming*, Addison-Wesley, 1962, by George Hadley.

Exercise 24 is inspired by *Dynamic Programming and Markov Processes*, John Wiley & Sons, 1960, by Ronald A. Howard.