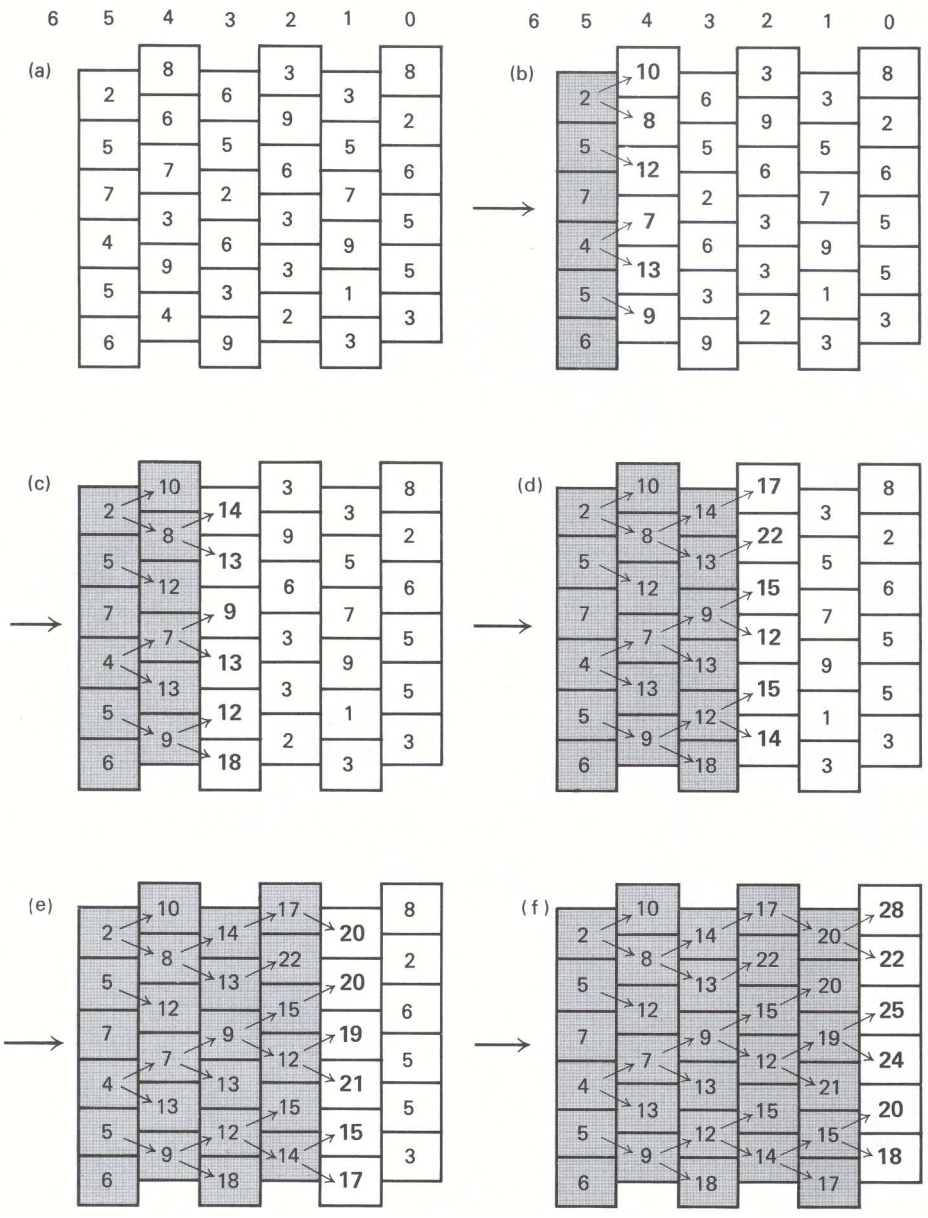

Figure 11.6 Solution by forward induction.

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:

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

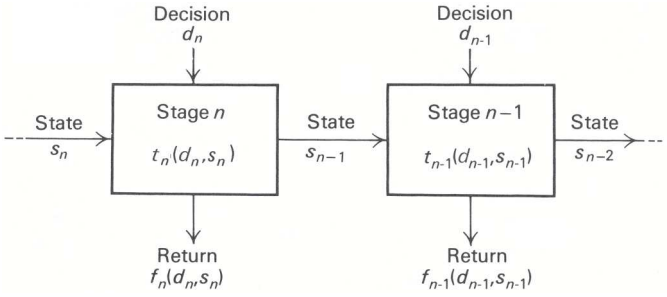

Figure 11.7 Multistage decision process.

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:

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

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:

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

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 ($ $\times$1000)

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