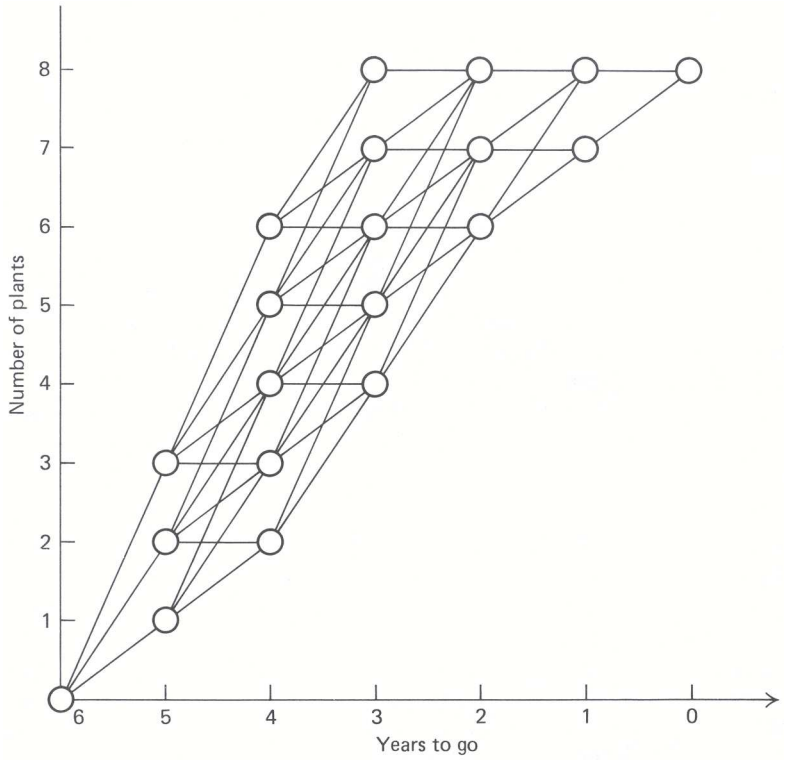

Figure 11.8 Allowable capacity (states) for each tage

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:

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

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