Dynamic Programming Models

Many planning and control problems in manufacturing, telecommunications and capital budgeting call for a sequence of decisions to be made at fixed points in time. The initial decision is followed by a second, the second by a third, and so on perhaps infinitely. Because the word *dynamic* describes situations that occur over time and programming is a synonym for planning, the original definition of dynamic programming was "planning over time." In a limited sense, our concern is with decisions that relate to and affect phenomena that are functions of time. This is in contrast to other forms of mathematical programming that often, but not always, describe static decision problems. As is true in many fields, the original definition has been broadened somewhat over the years to connote an analytic approach to problems involving decisions that are not necessarily sequential but can be viewed as such. In this expanded sense, dynamic programming (DP) has come to embrace a solution methodology in addition to a class of planning problems. It is put to the best advantage when the decision set is bounded and discrete, and the objective function is nonlinear.

This chapter is primarily concerned with modeling of deterministic, discrete systems. Although it is possible to handle certain problems with continuous variables, either directly or indirectly by superimposing a grid on the decision space, such problems will not be pursued here because they are better suited for other methods. In any case, modeling requires definitions of states and decisions, as well as the specification of a measure of effectiveness. For the usual reasons, a reduction in complexity of the real problem is also necessary. From a practical point of view, it is rarely possible to identify and evaluate all the factors that are relevant to a realistic decision problem. Thus the analyst will inevitably leave out some more or less important descriptors of the situation. From a computational point of view, only problems with relatively simple state descriptions will be solvable by dynamic programming. Thus abstraction is necessary to arrive at a formulation that is computationally tractable. Often a particular problem may have several representations in terms of the state and decision variables. It is important that the analyst realize that the choice of formulation can greatly affect his or her ability to find solutions.

Dynamic programming has been described as the most general of the optimization approaches because conceivably it can solve the broadest class of problems. In many instances, this promise is unfulfilled because of the attending computational requirements. Certain problems, however, are particularly suited to the model structure and lend themselves to efficient computational procedures; in cases involving discontinuous functions or discrete variables, dynamic programming may be the only practical solution method.

In the next section, we present an investment example to introduce general concepts and notation. The solution approach common to all dynamic programming is then outlined to motivate the need for the new notation. In the remainder of the chapter we describe several problem classes and their individual model characteristics. Solution procedures are left to the DP Methods chapter, as are situations with stochastic elements.

19.1 Investment Example

A portfolio manager with a fixed budget of \$100 million is considering the eight investment opportunities shown in Table 1. The manager must choose an investment level for each alternative ranging from \$0 to \$40 million. Although an acceptable investment may assume any value within the range, we discretize the permissible allocations to intervals of \$10 million to facilitate the modeling. This restriction is important to what follows. For convenience we define a unit of investment to be \$10 million. In these terms, the budget is 10 and the amounts to invest are the integers in the range from 0 to 4.

Table 1. Annual returns for alternative investments

Amount Invested (\$10 million)	Opportunity							
	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0
1	4.1	1.8	1.5	2.2	1.3	4.2	2.2	1.0
2	5.8	3.0	2.5	3.8	2.4	5.9	3.5	1.7
3	6.5	3.9	3.3	4.8	3.2	6.6	4.2	2.3
4	6.8	4.5	3.8	5.5	3.9	6.8	4.6	2.8

Table 1 provides the net annual returns from the investment opportunities expressed in millions of dollars. A ninth opportunity, not shown in the table, is available for funds left over from the first eight investments. The return is 5% per year for the amount invested, or equivalently, \$0.5 million for each \$10 million invested. The manager's goal is to maximize the total annual return without exceeding the budget.

Using notation introduced in the text, a mathematical programming statement of the problem is as follows.

$$\text{Maximize } z = r(1, x_1) + r(2, x_2) + \cdots + r_n(n, x_n) + e\, x_s$$

$$\text{subject to } x_1 + x_2 + \cdots + x_n + x_s = b$$

$$0 \le x_j \le u_j \text{ and integer, } j = 1, \ldots, n$$

In the model, x_j is the amount to invest in alternative j, $r(j, x_j)$ is the return from alternative j written as a function of x_j, u_j is an upper bound on the amount invested in opportunity j, and b is the initial budget. The funds remaining after all allocations are made is represented by the slack variable x_s. The unit return for any unspent money is e. Table 1 quantifies the function $r(j, x_j)$.

The problem as stated is similar in structure to the knapsack problem but the objective function is nonlinear. To formulate it as a mixed-integer linear program it would be necessary to introduce 32 binary variables, one for each nonzero level of investment. The slack variable, x_s, can be treated

as continuous. Rather than pursuing the MILP formulation, though, we will use the problem as an introduction to dynamic programming.

To begin, we ask the manager to decide on the amount to invest sequentially. That is, we first ask her to consider opportunity 1, then opportunity 2, and so on. This is difficult to do in a way that maximizes the total return. The manager notes that the more she invests in opportunity 1, the greater the annual return. Consequently, she might feel that a greedy approach is called for -- one that invests in the highest possible level, 4 in this case. It should be apparent, though, that such an approach may yield very poor results. Committing a large portion of the budget to early opportunities precludes potentially more attractive returns later on. Instead, we ask the manager to solve the problem backwards, starting with opportunity 8 conditioned on the funds available, then 7 and so on until she makes a decision for opportunity 1. With a little organization, we find that this procedure is possible.

First we ask, how many units should we invest in opportunity 8? The manager responds that she cannot make that decision unless she knows how much of the budget has already been spent for the first 7 opportunities. Then, the decision is obvious. For example, if all the budget is spent by the previous opportunities, the investment in 8 must be zero. If one unit of budget remains she will invest 1 in opportunity 8 if the return exceeds the 0.5, the value for leftover funds. In general, if x units are already spent, she can invest up to $10 - x$ in opportunity 8. Of course when x is 5 or less, 4 units can be invested and there will still be money left over.

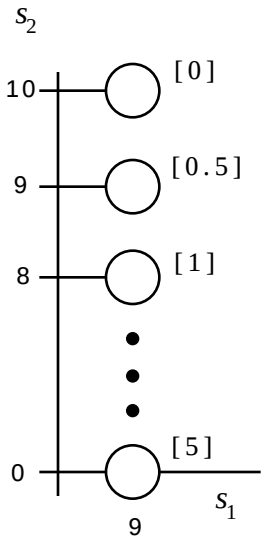


Figure 1. States for $s_1 = 9$

To formalize the dynamic programming approach, we define *states* and *decisions.* Here, a state can be described by the opportunity index s_1, and the amount already spent s_2. The state variables are contained in the vector $\mathbf{s} = (s_1, s_2)$.

We use $s_1 = 9$ to mean that there are no more opportunities. For this problem, we call any state that has s_1 equal to 9 a *final state* because there are no more decisions in our sequential process. The set of final states is $\boldsymbol{F}$. For the investment problem

$$\boldsymbol{F} = \{\mathbf{s} : s_1 = 9,\ 0 \le s_2 \le 10\}.$$

A final state has a value defined by the *final value function, $f(\mathbf{s})$* for $\mathbf{s} \in \boldsymbol{F}$. For this problem, the final value is the annual return of the funds not spent.

$$f(\mathbf{s}) = 0.5(10 - s_2) \text{ for } \mathbf{s} \in \boldsymbol{F}.$$

Graphically, we represent a state as a node in a network as in Fig. 1 where only the final states are shown. The final state values are in brackets next to the nodes.

Now we address the question of finding the optimal decision for opportunity 8. In general, a decision is identified by the *decision variable*, d, the amount to invest. Let d_8 be the number of units selected for opportunity 8. In state (8, 10) no budget remains so the optimal value of d_8 must be 0. In state (8, 9), the choice is between investing 0 or 1. For $d_8 = 0$, a unit of budget will remain for a final return of 0.5. The return for $d_8 = 1$ is 1, a clearly better result. The details of the decision process for four states are given in Table 2.

Table 2. Optimal decision process for opportunity 8

State **s**	Decision d	Decision objective $r(\mathbf{s}, d)$	Next state **s**'	Next state value $f(\mathbf{s}')$	Total return $r(\mathbf{s}, d)+ f(\mathbf{s}')$	Optimal decision $d^*(\mathbf{s})$	Optimal return $f(\mathbf{s})$
(8, 10)	0	0	(9, 10)	0	0	0	0
(8, 9)	0	0	(9, 9)	0.5	0.5		
	1	1.0	(9, 10)	0	1.0	1	1.0
(8, 8)	0	0	(9, 8)	1.0	1.0		
	1	1.0	(9, 9)	0.5	1.5		
	2	1.7	(9, 10)	0	1.7	2	1.7
(8, 7)	0	0	(9, 7)	1.5	1.5		
	1	1.0	(9, 8)	1.0	2.0		
	2	1.7	(9, 9)	0.5	2.2		
	3	2.3	(9, 10)	0	2.3	3	2.3
(8, 6)	0	0	(9. 6)	2.0	2.0		
	1	1.0	(9, 7)	1.5	2.5		
	2	1.7	(9, 8)	1.0	2.7		
	3	2.3	(9, 9)	0.5	2.8		
	4	2.8	(9, 10)	0	2.8	3 or 4	2.8

The computations for a particular state are shown between the parallel solid lines in the table. We see that a separate optimization is carried out for each state **s**. The column labeled d shows all feasible values of the decision variable. A value not in this list would use more than the budget available. The *decision objective* is the annual return for selecting the amount d. This value comes from Table 1. The column labeled **s**' is the *next state* reached by making decision d while in state **s**. The next state is given by the *transition function* $T(\mathbf{s}, \mathbf{d})$. For this problem the transition function is

$$\mathbf{s}' = (s_1', s_2') = T(\mathbf{s}, \mathbf{d}) \text{ where}$$

$$s_1' = s_1 + 1 \text{ and } s_2' = s_2 + d.$$

The value of the next state, $f(\mathbf{s}')$, has already been computed and is shown in the next column. The total return is the sum of the decision return and the next state value and is the quantity to be maximized.

For each value of $\mathbf{s}$, we compare the total returns for the different values of d and choose the one that gives the maximum. This is a simple one-dimensional problem solved by enumerating the alternatives. The optimal decision is $d^*(\mathbf{s})$, where the argument $\mathbf{s}$ indicates that the decision is a function of $\mathbf{s}$. Finally the optimal return is the value $f(\mathbf{s})$.

The computations are performed by solving the following *backward recursive equation*.

$$f(s_1, s_2) = \text{Max}\{r(s_1, d) + f(s_1 + 1, s_2 + d) : 0 \le d \le 4, \text{ and } s_1 \le 8, s_2 + d \le 10\}$$

It is a recursive equation because the function $f(\mathbf{s})$ appears on both sides of the equality sign. We can solve it, only if we proceed in a backward direction through the states. The details of the solution process are discussed in the chapter on DP methods.

The optimal decisions for opportunity 8 are shown in Fig. 2 as the lines between the states. When a tie occurs it can be broken arbitrarily. State (8, 6), for example, admits two optimal decisions $d^*(8, 6) = 3$ and $d^*(8, 6) = 4$. We have chosen 3 for the illustration. The values for the states are shown adjacent to the nodes.

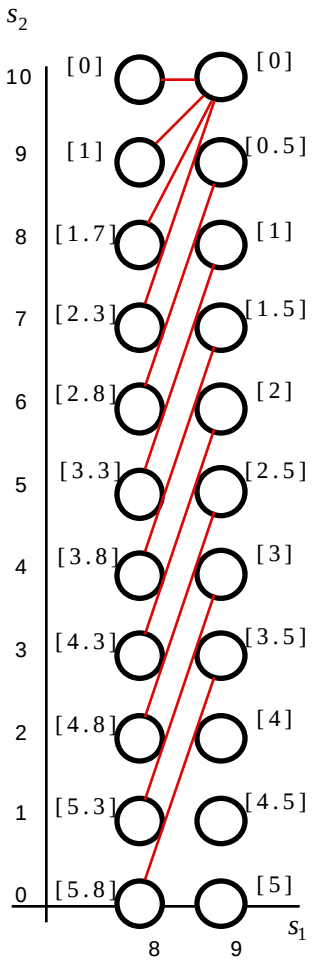


Figure 2. Optimal decisions for $s_1 = 8$

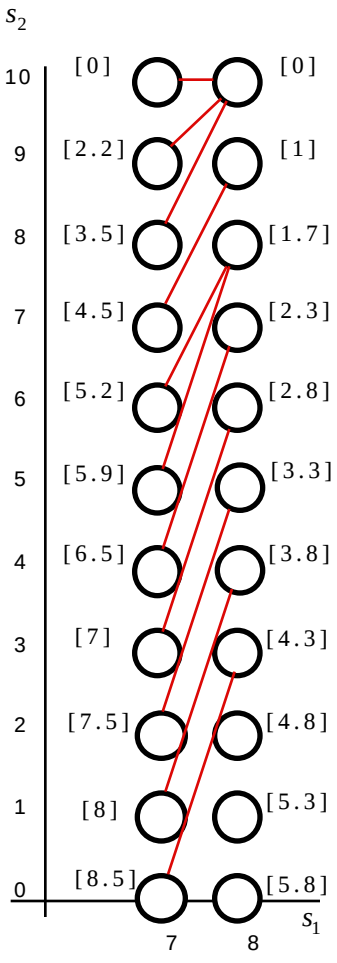


Figure 3. Optimal decisions for $s_1 = 7$

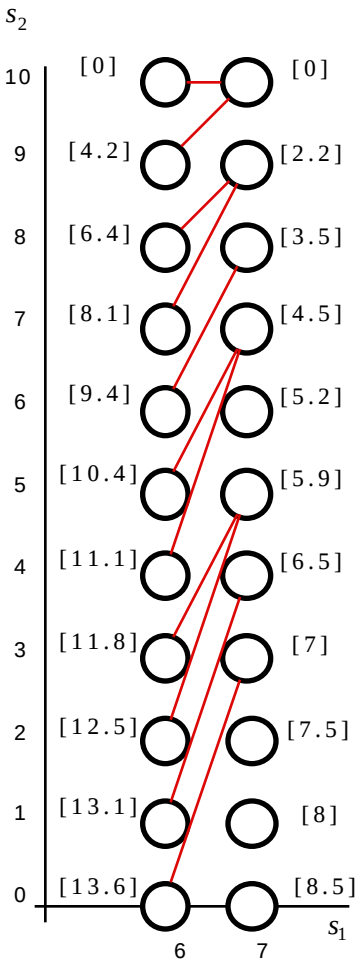


Figure 4. Optimal decisions for $s_1 = 6$

In a similar manner, once we know the function values for each state with $s_1 = 8$, we can compute the optimal decisions and function values for the states with $s_1 = 7$. These results are shown in Fig. 3. Again we take a backward step in Fig. 4 to compute the optimal decisions and function values when $s_1 = 6$.

The process continues until $s_1 = 1$. At this point, the manager must make a decision for opportunity 1. Since this is the first decision, she knows how much of the budget is already spent. It must be 0. Accordingly, we call (1, 0) the *initial state*. Now it is possible to decide on the amount to invest in opportunity 1 because the value of the remaining budget for opportunities 2 through 8 is known. The decision process associated with opportunity 1 is shown in Fig. 5.

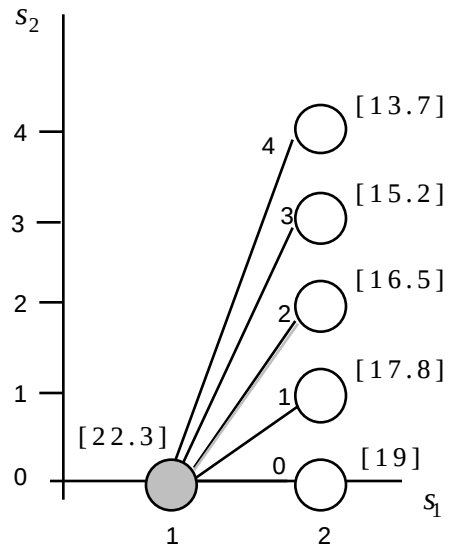


Figure 5. Optimal decision for state (1, 0)

The figure shows the five decisions associated with the initial state. The recursive equation used to determine the optimal value and decision at (1, 0) is

$$f(1,0) = \text{Max} \begin{Bmatrix} 0 + 19 \\ 4.1 + 17.8 \\ 5.8 + 16.5 \\ 6.8 + 13.7 \end{Bmatrix} = \text{Max} \begin{Bmatrix} 19 \\ 21.9 \\ 22.3 \\ 20.5 \end{Bmatrix} = 22.3$$

with the optimal decision $d^* = 2$.

Figure 6 depicts the collection of states used to solve the problem. The set of all feasible states is called the *state space* $\boldsymbol{S}$. When we represent all feasible states by nodes and all feasible decisions by arcs, the resultant network is called the *decision network.* Figure 6 is really a subnetwork for the investment problem because it includes only the states that can be reached by decisions from the initial state, and highlights only the optimal decisions. A critical feature of this model is that the return associated with each arc depends only on the states at its two end points.

The procedure just described allowed us to uncover the path through the decision network with the largest total return. For the investment problem, this path starts at the initial state (1,0) and terminates at the final state (9,10). It is marked by bold arcs in Fig. 6.

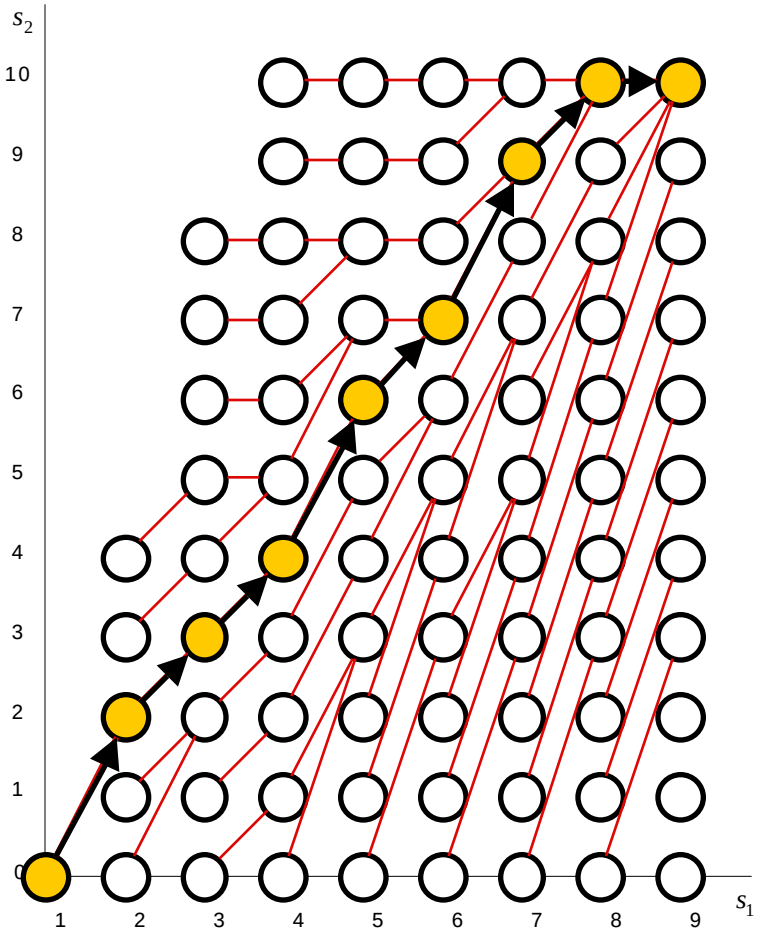


Figure 6. Decision network with optimal solution

The algorithm computes the optimal decision leaving each state. For the example, the decision in state (1, 0) is 2, leading to state (2, 2). The optimal decision in state (2, 2) is 1, leading to state (3, 3), and so on. The process is called *forward recovery* since it begins in an initial state and moves in the forward direction until a final state is reached. Table 3 lists the optimal decisions from one state to the next.

Table 3. Path for the optimal solution

Index	State s_1	State s_2	Decision d	Return $f(s_1, s_2)$
1	1	0	2	22.3
2	2	2	1	16.5
3	3	3	1	14.7
4	4	4	2	13.2
5	5	6	1	9.4
6	6	7	2	8.1
7	7	9	1	2.2
8	8	10	0	0
9	9	10	—	0

In addition to the path leaving (1, 0), it is possible to identify an optimal path for every state $\mathbf{s}_k$ by tracing the optimal decisions from $\mathbf{s}_k$ to a final state $\mathbf{s}_f$, where $\mathbf{s}_f \in \boldsymbol{F}$. The complete set of decisions is called a *policy* because it specifies an action for every state. In general, the amount of work required to determine an optimal policy is proportional to the size of the state space. During the solution process for the investment example, a simple problem characterized by a single variable was solved for each state. This gave us an optimal policy for all $\mathbf{s} \in \boldsymbol{S}$.

19.2 Model Components

The language of dynamic programming is quite different from that used in other areas of mathematical programming. Although it is common to have an objective to be optimized and a set of constraints that limits the decisions, a DP model represents a sequential decision process rather than an algebraic statement of a problem. The two principal components of the dynamic programming model are the states and decisions. A state is like a snapshot of the situation at some point in time. It describes the developments in sufficient detail so that alternative courses of action starting from the current state, can be evaluated. A decision is an action that causes the state to change in some predefined way. Thus a decision causes a movement from one state to another. The state-transition equations govern the movement. A sequential decision process starts in some initial state and advances forward, continuing until some final state is reached. The alternating sequence of states and decisions describes a path through the state space.

Although many situations can be modeled in this way, the principal difficulty is to define the state space so that sufficient information is provided to evaluate alternative choices. For a chess game, the state must describe the arrangement of pieces on the board at any point in the game. Enumerating the states is a well defined task, but not practical because the number of possible board arrangements is unmanageably large. The same is true for many combinatorial optimization problems such as the traveling salesman problem (TSP). The state space of the TSP grows exponentially with the number of cities.

Another aspect of the model that requires careful consideration is the measure of effectiveness used to evaluate alternative paths through the state space. The optimal path is the one that maximizes or minimizes this measure. A dynamic programming algorithm aims at finding at least one such path or sequence. There are a number of ways of doing this, but for the moment it is sufficient to mention that solution methods are closely linked to modeling conventions. This follows from the desire to make the computational procedures as universally applicable as possible. If a procedure is to solve a wide variety of problems, a standard form must be established for model input. In this section, we define the notation more carefully using the investment problem as an example.

General Format

> As we have seen, the components of a DP model consist of the state vector, the decision vector, the feasible state space, the feasible decision set for each state, the initial states, the final states, the transition function, the form of the path objective, and the final value function. Although several of these terms are similar to those used in describing the mathematical programming models discussed up until now, the differences are what stand out. Table 4 defines the individual components of a dynamic program in such a way that allows for a broad range of applications.

Table 4. Components of the general dynamic programming model

Component	Description
State	$\mathbf{s} = (s_1, s_2, \ldots, s_m)$, where s_i is the value of state variable i and m is the number of state variables
Initial state set	$\boldsymbol{I}$ = {**s** : nodes in decision network with only leaving arcs}
Final state set	$\boldsymbol{F}$ = {**s** : nodes in decision network with only entering arcs}
State space	$\boldsymbol{S}$ = {**s** : **s** is feasible}
Decision	$\mathbf{d}(\mathbf{s}) = (d_1, d_2, \ldots, d_p)$, where d_j is the value of the jth decision variable and p is the number of decision variables
Feasible decision set	$\boldsymbol{D}(\mathbf{s})$ = {**d** : **d** leads to a feasible state from state **s**}
Transition function	$\mathbf{s}' = T(\mathbf{s}, \mathbf{d})$, a function that determines the next state, **s**', reached when decision **d** is taken from state **s**
Decision objective	$z(\mathbf{s}, \mathbf{d})$, the measure of effectiveness associated with decision **d** taken in state **s**
Path objective	$z(\mathbf{P})$, the measure of effectiveness defined for path **P**. This function describes how the objective terms for each state on the path and the final value function are combined to obtain a measure for the entire path.
Final value function	$f(\mathbf{s})$ given for all **s** $\boldsymbol{F}$

Sequential Decision Problem

To formulate a problem as a DP, it must be stated in terms of a sequential set of decisions. As presented, the investment problem does not have this characteristic. In particular, the solution is a statement of investment allocations presumably all to be made at the same time rather than serially over a finite time horizon. To accommodate the sequential nature of dynamic programming, the numbers assigned to the investments were used to provide an artificial order. Thus we first decide on the amount to invest in opportunity 1, then in opportunity 2, and finally in opportunity n.

States

The problem must be described in a manner such that a solution is a sequence of alternating states and decisions. The state represents the current alternative under consideration and the amount of the budget used to this point. Thus two pieces of information are described by the state requiring the introduction of two state variables. In general, we call the m-dimensional vector $\mathbf{s} = (s_1, \ldots, s_m)$ the state, and its components s_i the state variables. For the investment problem, $m = 2$ and

s_1 = index of the current alternative being considered ($s_1 = 1, \ldots, n+1$)

s_2 = amount of the budget used prior to this investment opportunity

($s_2 = 0, 1, \ldots, b$)

$\mathbf{s} = (s_1, s_2)$.

In some textbook expositions on dynamic programming, the term "stage" is used to identify the sequence of decisions. In our exposition, we will not use the concept of a stage but rather include the stage information as the first component of the state vector. Although this approach may seem awkward at first to those already familiar with the stage terminology, it allows a more general class of problems to be modeled as dynamic programs.

Decisions

The decision at any particular state is the amount to invest in the opportunity identified by s_1. In general, the set of all possible decisions is denoted by $\boldsymbol{D}$, whereas a particular decision as a function of state $\mathbf{s}$ is denoted by $\mathbf{d}(\mathbf{s})$. To accommodate cases in which the decision has more than one dimension, $\mathbf{d} = (d_1, d_2, \ldots)$ is specified as a vector with each component identified by a lowercase subscripted letter d_l. In the present instance the decision is just the amount to invest, so $\mathbf{d}$ has only one dimension.

$\boldsymbol{D} = \{0, 1, \ldots, b\}$

d = amount to invest in opportunity s_1

$\mathbf{d}(\mathbf{s}) = (d(\mathbf{s}))$

Solution

A solution is an alternating sequence of states and decisions that have indices indicating their place in the sequence. The process starts in state $\mathbf{s}_1$, called the initial state. For the investment problem, the initial state is

$$\mathbf{s}_1 = (1, 0)$$

indicating that this is the first opportunity and none of the budget has been allocated. The first decision is $d(\mathbf{s}_1)$, the investment in opportunity 1. The new state $\mathbf{s}_2$ must be equal to

$$\mathbf{s}_2 = (2, d(\mathbf{s}_1))$$

since the value of the decision variable $d(\mathbf{s}_1)$ is precisely the amount invested in the first alternative and the next opportunity is 2. The decision $d(\mathbf{s}_2)$ moves the process from state $\mathbf{s}_2$ to $\mathbf{s}_3$. The value of $\mathbf{s}_3$ must be

$$\mathbf{s}_3 = (3, d(\mathbf{s}_1) + d(\mathbf{s}_2)).$$

The first component, $s_1 = 3$, of the vector $\mathbf{s}_3$ indicates the index of the next alternative to be considered, and the second component gives the total investment associated with the first two decisions.

Transition Function

As each decision is made, the state changes in a predictable way. The function that gives the value of the next state vector in terms of the current state and decision is called the transition function. Let the state at the kth step of the process be $\mathbf{s}_k$, and let the decision taken at this step be $\mathbf{d}_k$. The next state is $\mathbf{s}_{k+1}$ and is determined by the transition function as follows.

$$\mathbf{s}_{k+1} = T_k(\mathbf{s}_k, \mathbf{d}_k) \tag{1}$$

Very often $T_k(\mathbf{s}_k, \mathbf{d}_k)$ does not depend on the sequence number of the decision, or, as in our example, the sequence number is included as the first component of the state vector. In such cases, one can simplify the notation by denoting the current state by $\mathbf{s}$, the current decision by $\mathbf{d}$, and the next state by $\mathbf{s}'$. Now the transition function can be written without the subscript; i.e.,

$$\mathbf{s}' = (s'_1, s'_2) = T(\mathbf{s}, \mathbf{d}) \tag{2}$$

where $T(\cdot, \cdot)$ is a general function of $\mathbf{s}$ and $\mathbf{d}$. When there is no ambiguity, we will always use Eq. (2) without the index k, rather than Eq. (1). Note that when the state vector has more than one component, the transition function is multidimensional. It must describe how each component of the state vector changes.

For the investment problem, the transition function is separable in the two state variables and can be written as

$$s'_1 = s_1 + 1 \text{ and } s'_2 = s_2 + d.$$

State Space

The collection of all feasible states is called the state space and is identified by the symbol $\boldsymbol{S}$. For the example, the first state variable, s_1, ranges from 1 to n +1. (For modeling purposes, the decision associated with opportunity n results in the transition to $s'_1 = n + 1$.) Because only positive integer investments are allowed and the total investment cannot exceed the budget, it is clear that the second state variable, s_2, must be integer and lie between 0 and b. Thus the state space for the example is

$$\boldsymbol{S} = \{(1, 0) \quad \{\mathbf{s} : 2 \le s_1 < n + 1, 0 \le s_2 \le b, s_1 \text{ and } s_2 \text{ integer}\}\}.$$

Decision Network

As mentioned, a conceptually useful representation of a DP model is a decision network, partially illustrated for the investment problem in Fig. 6. The elements of the state space are shown as the nodes in the figure. Because a decision leads from one state to another as defined by the

transition function, the decisions are represented as arcs. Only the optimal decisions are shown in Fig. 6; the full network would include an arc for each feasible decision. When both the decision space and state space are discrete, it is always possible to construct a decision network for a DP although it may be impractical to do so when the number of states is large.

Path

A solution to the problem is a sequence of states and decisions that defines a path through the network. We represent a path $\mathbf{P}$ as a vector of alternating states and decisions beginning at the initial state $\mathbf{s}_1$ and ending at a final state $\mathbf{s}_{n+1}$. For the investment problem, we have

$$\mathbf{P} = (\mathbf{s}_1, \mathbf{d}_1, \mathbf{s}_2, \mathbf{d}_2, \mathbf{s}_3, \ldots, \mathbf{s}_8, \mathbf{d}_8, \mathbf{s}_9).$$

Every feasible solution to the problem can be associated with some path through the network.

Acyclic Decision Network

With the help of Fig. 6, several new definitions can be introduced. First note that all the arcs in the figure are drawn from left to right. This indicates that there are no cycles in this network making it acyclic. With an acyclic decision network, there must be some set of nodes that has no entering arcs. These nodes comprise the initial states and are identified as the set $\boldsymbol{I}$. State (1, 0) is the sole initial state in the investment example. Also, there must be some set of nodes that has no leaving arcs. The collection of these nodes is the final set $\boldsymbol{F}$. There are 11 final states for the example (9, s_2), where s_2 ranges from 0 to 10. It should be clear that a path corresponding to a solution begins at a state in $\boldsymbol{I}$ and ends at a state in $\boldsymbol{F}$.

Feasible Decision Set

Only certain decisions from a given state will lead to a feasible state. If $\mathbf{s}' = T(\mathbf{s}, \mathbf{d})$ and $\mathbf{s}'$ is a feasible state, then $\mathbf{d} \in \boldsymbol{D}$ is called a feasible decision for state $\mathbf{s}$. The set of feasible decisions for state $\mathbf{s}$ is $\boldsymbol{D}(\mathbf{s}) \subseteq \boldsymbol{D}$. For the investment problem

$$\boldsymbol{D}(\mathbf{s}) = \{\mathbf{d} : 0 \le s_2 + d \le b, \text{ and } d \text{ is integer}\}$$

where $\mathbf{d} = (d)$.

Path Objective

The next step in the modeling process is to define a measure of effectiveness for comparing alternative paths and selecting the optimum. We denote the path objective by $z(\mathbf{P})$. The optimal path, $\mathbf{P}^*$, is determined by solving either

$$z(\mathbf{P}^*) = \text{Min}_{\mathbf{P}}\, z(\mathbf{P}) \quad \text{or} \quad z(\mathbf{P}^*) = \text{Max}_{\mathbf{P}}\, z(\mathbf{P}).$$

Whether the objective is to maximize or minimize depends on the problem under consideration. The optimization is conducted over all feasible paths.

For the investment problem, the goal is to maximize return so the objective can be written in the following manner.

$$\text{Max}_{\mathbf{P}}\, z(\mathbf{P}) = \text{Max}_{\mathbf{s}\ \ \boldsymbol{S},\mathbf{d}\ \ \boldsymbol{D}(\mathbf{s})}\, z(\mathbf{s}_1, \mathbf{d}_1, \mathbf{s}_2, \ldots, \mathbf{s}_8, \mathbf{d}_8, \mathbf{s}_9)$$

For computational tractability, we require that the objective function take a separable form that can be expressed as the sum of $n + 1$ terms that individually consist of a state and a decision. In general, we have

$$z(\mathbf{P}) = z(\mathbf{s}_1, \mathbf{d}_1) + z(\mathbf{s}_2, \mathbf{d}_2) + \cdots + z(\mathbf{s}_n, \mathbf{d}_n,) + f(\mathbf{s}_{n+1})$$

or
$$z(\mathbf{P}) = \sum_{k=1}^{n} z(\mathbf{s}_k, \mathbf{d}_k) + f(\mathbf{s}_{n+1}) \qquad (3)$$

where the last term $f(\mathbf{s}_{n+1})$ assigns a payoff to the final state and is called the final value function. This function must be given as part of the model. For the example, the final state is $\mathbf{s}_9 = (9, s_2)$, and it was assumed that $f(\mathbf{s}_9) = e(b - s_2)$, where e is the unit value of unallocated budget.

Because $\mathbf{s}'$ can be computed from $(\mathbf{s}, \mathbf{d})$ via the transition function it is sometimes useful to express the summation terms in the objective function as explicit functions of $\mathbf{s}'$ such as $z(\mathbf{s}, \mathbf{d}, \mathbf{s}')$ or $z(\mathbf{s}, \mathbf{s}')$. In Eq. (3), each term depends only on the current state and the current decision. This type of function arises frequently. For notational convenience, we write the path objective function in such a way that omits the index of the decisions. In this form, the objective is the sum over the states and the corresponding decisions taken at those state in the path $\mathbf{P}$ plus the value of the final state denoted by $\mathbf{s}_f$. We also omit the explicit dependence of $\mathbf{P}$ on $\mathbf{s}$ and $\mathbf{d}$.

$$z(\mathbf{P}) = \sum_{\mathbf{s}\ \ \boldsymbol{S},\mathbf{d}\ \boldsymbol{D}(\mathbf{s})} z(\mathbf{s}, \mathbf{d}) + f(\mathbf{s}_f)$$

For the investment problem, the function z can be written in terms of only s_1, the index of the current opportunity, and d, the amount of investment. Thus

$$z(\mathbf{s}, \mathbf{d}) = f(s_1, d)$$

where $f(s_1, d)$ is the return obtained by investing amount d in opportunity s_1. As mentioned, we take the final value function to be a linear expression of the unspent funds.

$$f(\mathbf{s}) = e(b - s_2) \text{ for } \mathbf{s} \quad \boldsymbol{F}$$

The rest of the chapter describes dynamic programming models associated with several problem classes.

19.3 Resource Allocation Problems

Describing a problem in a format that is suitable for the computational techniques of dynamic programming is perhaps more of an art than for other mathematical programming methods. In the remaining sections we attempt to aid the modeling process by identifying several problem classes. Familiarity with these classes may suggest how a given problem should be stated. It may fit directly into one of the classes or more commonly may require a series of minor modifications.

Resource allocation problems can be viewed as generalizations of the investment problem considered in the first section. In particular, suppose that n investment opportunities are available, each having a payoff that depends on the level of investment. Let the decision d_j represent the level of investment in alternative j for $j = 1, \ldots, n$. Only nonnegative integer values of d_j will be considered up to some finite upper limit. When $d_j = 0$, no investment takes place; when $d_j = 1$, we are investing at the first level; when $d_j = 2$ we are investing at the second level, and so on. This structure gives us flexibility in that the levels do not necessarily have to equal the amounts of the investment.

Mathematical Programming Model

Let $c(j, d_j)$ be the return for investing at level d_j in opportunity j. The returns are functions of j and d_j and may or may not be linear. The objective of the problem is to maximize the total return from the investment policy; that is,

$$\text{Maximize } z = \sum_{j=1}^{n} c(j, d_j) \tag{4}$$

Portions of one or more resources are consumed with each allocation. A typical resource is the budget; however, there may be several others. In a multiperiod problem, for example, the investments may incur claims on future funds and so will require a resource constraint for each time period. In addition, there may be limits on the amount of money that can be invested in a certain instrument or particular sector of the economy.

In defining the model, we make use of the following notation.

m = number of resources

b_i = amount of resource i available

$a(i, j, d_j)$ = amount of resource i used by investing at level d_j in alternative j

We further restrict the values of $a(i, j, d_j)$ and b_i to be integer. Now, algebraic constraints can be written that limit the amount of each resource that may be used by a feasible solution.

$$\sum_{j=1}^{n} a(i,\ j,\ d_j) \le b_i\ ,\ \ i = 1,\ldots,m \qquad (5)$$

The objective function (4) together with the inequalities in (5) give a mathematical programming statement of an investment problem with resource constraints. Since the decision variables d_j are required to assume only discrete values, neither linear nor nonlinear programming is an appropriate solution technique. An enumerative solution procedure such as integer or dynamic programming must be used.

Dynamic Programming Model

To formulate this problem as a dynamic program, a solution must be described as a sequence of states and decisions. The sequence of decisions is easily obtained by arbitrarily ordering the investment opportunities. Thus the first decision is the level of investment in alternative 1, the second is the level for alternative 2, and so on. To complete the model we must define each of its components in Table 3. Many variations in the problem statement can be accommodated with minor variations in the model.

Table 5. General resource allocation model

Component	Description
State	$\mathbf{s} = (s_1,\ s_2,\ldots,\ s_{m+1})$, where s_1 = alternative currently under consideration s_i = amount of resource $i-1$ used up prior to the current decision, $i = 2,\ldots,m+1$
Initial state set	$\boldsymbol{I} = \{(1,\ 0,\ldots,0)\}$ We start with alternative 1 and no resources used.
Final state set	$\boldsymbol{F} = \{\mathbf{s} : s_1 = n + 1\}$ After all the alternatives have been considered we are finished.
State space	$\boldsymbol{S} = \boldsymbol{I} \quad \{\mathbf{s} : 2 \le s_1 \le n + 1,\ 0 \le s_i \le b_{i-1},\ \ i = 2,\ldots,m+1\}$ Integrality is also required for all elements of $\boldsymbol{S}$ which consists of the initial state set plus all the integer values within the specified ranges.
Decision	$\mathbf{d}(\mathbf{s}) = (d)$, where d is the investment level for alternative s_1
Feasible decision set	$\boldsymbol{D}(\mathbf{s}) = \{d : 0 \le s_i + a(i-1,\ s_1,\ d) \le b_{i-1},\ \ i = 2,\ldots,m+1;\ d \ge 0$ and integer$\}$ All decisions that do not exceed the resources are feasible.

Transition function	$\mathbf{s}' = T(\mathbf{s}, d)$, where $s'_1 = s_1 + 1$ $s'_i = s_i + a(i-1, s_1, d), \quad i = 2, \ldots, m+1$ We move to the next alternative with the amount of resources used up by the decisions made previously.
Decision objective	$z(\mathbf{s}, \mathbf{d}) = c(s_1, d)$ The contribution to the objective is determined by the payoff function of the current alternative for the current decision. It does not depend on the successor state $\mathbf{s}'$.
Path objective	Maximize $z(\mathbf{P}) = \sum_{\mathbf{s} \in S, \mathbf{d} \in D(\mathbf{s})} z(\mathbf{s}, \mathbf{d}) + f(\mathbf{s}_f)$ The total return is the sum of the decision objectives over the opportunities.
Final value function	$f(\mathbf{s}) = 0$ for $\mathbf{s} \in F$ If there is a value for leftover resources we can include it in the final value function. Here we assume the value is 0.

Example 1 – *Binary Knapsack Problem*

Consider a boy scout packing his knapsack for an overnight camping trip. He has a set of n items that he can bring. There are no duplicates and item j weighs an integer amount $w(j)$. Unfortunately, the total weight of the items that he is considering is greater than the amount W that he can reasonably carry. To help determine which items to pack, he has assigned a benefit $c(j)$ to item j. The goal is to maximize total benefit. In the integer programming chapters, this problem was called the binary knapsack problem because it was modeled using 0-1 decision variables whose individual values corresponded to either selecting or not selecting an item.

The boy scout clearly faces a resource allocation problem, so it should be possible to describe his situation in dynamic programming terms. The specific components of the model are listed in Table 6. Because weight is the only resource, $m = 1$. The number n in the general model corresponds to the number of items under consideration.

Table 6. Binary knapsack problem

Components	Description
State	$\mathbf{s} = (s_1, s_2)$, where s_1 = item currently under consideration s_2 = weight allocated prior to the current decision
Initial state set	$\boldsymbol{I} = \{(1, 0)\}$
Final state set	$\boldsymbol{F} = \{\mathbf{s} : s_1 = n + 1, 0 \le s_2 \le W\}$
State space	$\boldsymbol{S} = \boldsymbol{I} \quad \{\mathbf{s} : 2 \le s_1 \le n + 1, 0 \le s_2 \le W\}$ Integrality is also required for all elements of $\boldsymbol{S}$.
Decision	$\mathbf{d}(\mathbf{s}) = (d)$, where $d = \begin{cases} 0 & \text{if item } s_1 \text{ is not packed} \\ 1 & \text{if item } s_1 \text{ is packed} \end{cases}$
Feasible decision set	$\boldsymbol{D}(\mathbf{s}) = \{d : 0 \le s_2 + w(s_1)d \le W\}$ The decision $d = 0$ is always feasible, while $d = 1$ is feasible if it does not violate the weight constraint.
Transition function	$\mathbf{s}' = T(\mathbf{s}, \mathbf{d})$, where $s_1' = s_1 + 1$ and $s_2' = s_2 + w(s_1)d$
Decision objective	$z(\mathbf{s}, \mathbf{d}) = c(s_1)d$ This term does not depend on the successor state $\mathbf{s}'$.
Path objective	Maximize $z(\mathbf{P}) = \sum_{\mathbf{s} \in \boldsymbol{S}, \mathbf{d} \in \boldsymbol{D}(\mathbf{s})} z(\mathbf{s}, \mathbf{d}) + f(\mathbf{s}_f)$
Final value function	$f(\mathbf{s}) = 0$ for all $\mathbf{s} \in \boldsymbol{F}$ There is no value associated with any amount of unused resource.

As an example, consider a knapsack problem with 15 items. The benefits and weights are listed in Table 7. We have chosen the parameters so that no item is dominated by another; that is, there is no item with a weight that is greater than some other but with a smaller benefit.

Table 7. Data for binary knapsack problem

Item, j	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Benefit, $c(j)$	8	9.5	11.5	12.8	14.1	18.6	19.8	22.4	23.6	25	29.1	32	33	34.6	37.2
Weight, $w(j)$	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

With a weight limitation of 30, there are 466 states in $\boldsymbol{S}$. This number can be reduced somewhat by removing states that cannot be reached from the initial state (1, 0). We report the optimum in the order of the decisions starting from (1, 0). The optimal decision for this state is to bring 0 of item 1, so the next state is computed from the transition equation as

$$s_1' = s_1 + 1 = 1 + 1 = 2$$

and $$s_2' = s_2 + w(s_1)d = 0 + 6 \times 0 = 0.$$

In a similar manner, the entire optimal sequence of decisions is derived. The objective function value is the sum of the decision returns. The value for (1, 0) is the total benefit of the knapsack, $z(\mathbf{P}^*)$. Table 8 indicates the optimal path in the format provided by the Teach DP Excel add-in. The solution calls for items 10 and 12 to be included in the knapsack giving a total value of 57 for $z(\mathbf{P}^*)$. The computations were performed using backward recursion and forward recovery.

Table 8. Optimal solution of the binary knapsack problem

Index	**s**		**d**(**s**) = *d*	*z*(**P**)	Action
	s_1	s_2			
1	1	0	0	57	Bring 0 of item 1
2	2	0	0	57	Bring 0 of item 2
3	3	0	0	57	Bring 0 of item 3
4	4	0	0	57	Bring 0 of item 4
5	5	0	0	57	Bring 0 of item 5
6	6	0	0	57	Bring 0 of item 6
7	7	0	0	57	Bring 0 of item 7
8	8	0	0	57	Bring 0 of item 8
9	9	0	0	57	Bring 0 of item 9
10	10	0	1	57	Bring 1 of item 10
11	11	14	0	32	Bring 0 of item 11
12	12	14	1	32	Bring 1 of item 12
13	13	30	0	0	Bring 0 of item 13
14	14	30	0	0	Bring 0 of item 14
15	15	30	0	0	Bring 0 of item 15
16	16	30	—	0	Finished

Example 2 *– Binary Knapsack with Two Constraints*

To illustrate the effect of including a second resource, we solve the same problem but with a volume constraint added. With two constraints another state variable is necessary. The components of the model that have changed are shown in Table 9. In addition to the notation used in Example 1, we

defined V to be the knapsack volume and $v(j)$ to be the volume required by item j.

Table 9. Modifications for binary knapsack problem with two constraints

Components	Description
State	$\mathbf{s} = (s_1, s_2)$, where s_1 = item currently under consideration s_2 = weight allocated prior to the current decision s_3 = volume allocated prior to the current decision
Initial state set	$\boldsymbol{I} = \{(1, 0, 0)\}$
Final state set	$\boldsymbol{F} = \{\mathbf{s} : s_1 = n + 1, 0 \le s_2 \le W, 0 \le s_3 \le V\}$
State space	$\boldsymbol{S} = \boldsymbol{I} \quad \{\mathbf{s} : 2 \le s_1 \le n + 1, 0 \le s_2 \le W, 0 \le s_3 \le V\}$ Integrality is also required for all elements of $\boldsymbol{S}$.
Transition function	$\mathbf{s}' = T(\mathbf{s}, \mathbf{d})$, where $s_1' = s_1 + 1$, $s_2' = s_2 + w(s_1)d$ and $s_3' = s_3 + v(s_1)d$

The volume of each item is given in Table 10 while the benefit and weight data are the same as in Table 7. The total volume was set at 30. The total number of elements in the state space is

$$|\boldsymbol{S}| = 14 \times 31 \times 31 + 1 = 14{,}416.$$

We chose to solve the problem using only the states that are reachable from the initial state. The corresponding state space has only 729 elements. The process used to generate only reachable states is described in the DP Methods chapter and is often leads to sharp reductions in the computational effort. The solution to the new problem is to bring only items 6 and 15. With the additional constraint, the value of the total return decreases to 55.8. The volume constraint is tight but the weight constraint is loose at the optimum.

Table 10. Data for binary knapsack problem

Item, j	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Volume, $v(j)$	5	6	7	11	9	10	15	16	14	14	18	19	19	22	20

Example 3 – *Personnel Assignment Problem*

As a second illustration of a problem with two resources and a two-dimensional decision vector, consider a company employing three electrical

engineers (EE), three mechanical engineers (ME), and an unlimited number of technicians (Techs). The company has four jobs to do in the next week, A, B, C and D. Table 11 identifies the time required to do each job with various combinations of personnel. A synergy exists for certain pairs but at most two engineers can be assigned to a job. An additional restriction is that if Techs are assigned to a job, no engineers are to be used.

Table 11. Time to perform jobs

Job	Techs	1 ME	2 MEs	1 EE	2 EEs	1 ME & 1 EE
A	45	49	30	47	21	15
B	—	73	15	—	27	20
C	60	52	24	78	54	—
D	75	70	57	61	80	57

The problem is to assign workers to jobs so that the total time is minimized. The DP model specified in Table 12 can be used for this purpose. The first state variable, s_1, assumes the values 1 through 4 to correspond to the four jobs A through D. A value of 5 for this variable indicates a final state. The other two state variables hold the number of engineers remaining of each type. Note that the resource state variables (s_2 and s_3) can indicate either the amount of a resource already used up as in Examples 1 and 2, or the amount remaining as defined here.

Table 12. Personnel assignment problem model

Component	Description
State	$\mathbf{s} = (s_1, s_2, s_3)$, where s_1 = job number with A, B, C and D being assigned values 1, 2, 3, and 4. The value 5 indicates that a final state has been reached. s_2 = number of MEs remaining s_3 = number of EEs remaining No state variable is necessary for Techs since there is an unlimited supply.
Initial state set	$\boldsymbol{I} = \{(1, 3, 3)\}$ Start with all ME's and EE's available.
Final state set	$\boldsymbol{F} = \{(5, s_2, s_3) : s_2 = 0, 1, 2, 3 \text{ and } s_3 = 0, 1, 2, 3\}$ It is not necessary to use all the engineers.

State space	$\boldsymbol{S} = \boldsymbol{I} \quad \{\mathbf{s} : 2 \le s_1 \le 5, 0 \le s_2 \le 3, 0 \le s_3 \le 3\}$ Integrality is required for all elements of $\boldsymbol{S}$.
Decision	$\mathbf{d}(\mathbf{s}) = (d_1, d_2)$, where $\begin{cases} d_1 = \text{number of MEs assigned} \\ d_2 = \text{number of EEs assigned} \end{cases}$
Feasible decision set	$\boldsymbol{D}(\mathbf{s}) = \{\mathbf{d} : d_1 + s_2 \le 3,\ d_2 + s_3 \le 3\}$ The assignment must not exceed the number available.
Transition function	$\mathbf{s}' = T(\mathbf{s}, \mathbf{d})$ where $s'_1 = s_1 + 1$, $s'_2 = s_2 + d_1$ and $s'_3 = s_3 + d_2$
Decision objective	$z(\mathbf{s}, \mathbf{d}) = a(\mathbf{s}, \mathbf{d})$ Here $a(\mathbf{s}, \mathbf{d})$ is the cost of doing job s_1 with assignment (d_1, d_2). It does not depend on the successor state s'.
Path objective	Maximize $z(\mathbf{P}) = \sum_{\mathbf{s}\ \boldsymbol{S},\mathbf{d}\ \boldsymbol{D}(\mathbf{s})} z(\mathbf{s}, \mathbf{d}) + f(\mathbf{s}_f)$
Final value function	$f(\mathbf{s}) = 0$ for all $\mathbf{s} \quad \boldsymbol{F}$ No additional benefit is obtained if any of the engineers are not assigned to jobs.

This example illustrates a situation where there are two decisions to make for every job. The decision objective is a nonseparable function of the variables d_1 and d_2. The optimal solution is given in Table 13.

Table 13. Optimal solution of the personnel assignment problem

Index	s_1	s_2	s_3	d_1	d_2	$z(\mathbf{P})$	Decision
1	1	3	3	2	0	140	Job A: use 2 ME, use 0 EE
2	2	1	3	1	1	119	Job B: use 1 ME, use 1 EE
3	3	0	2	0	2	99	Job C: use 0 ME, use 2 EE
4	4	0	0	0	0	75	Job D: use Techs only
5	5	0	0	—	—	0	Finished

19.4 Line Partitioning Problems

Another class of problems for which dynamic programming can be effectively applied involves the partitioning of a line into non-overlapping segments. Applications include cutting sheet metal and cloth, developing a machine overhaul schedule, and setting up inspection stations along an assembly line. In the definition of these problems both the state and decision vectors have a single component, making them easy to solve. From a modeling point of view, they are illustrative of the case where the classical stage representation for dynamic programming is not appropriate.

Problem Statement

Consider a line, as in Fig. 7, with n+1 discrete points or nodes numbered 0 to n starting at the left. The problem is to find an optimal partition of the line into segments such that each segment begins at one node and ends at another. Some objective or payoff function is associated with each continuous subsequence of nodes, and is typically nonlinear. Figure 8 shows one possible partition.

Figure 7. Line with discrete points

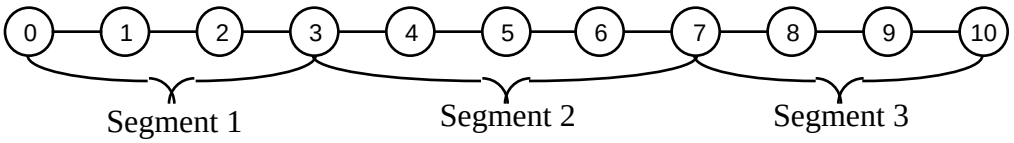


Figure 8. Line divided into segments

A vector of selected nodes defines a solution, with each adjacent pair of nodes defining a segment of the line. Thus a general solution comprising k segments can be written as a vector; that is,

$$(0,\ i_1,\ i_2,\ i_3, \ldots,\ i_{k-1},\ n),$$

where $\quad 0 < i_1 < i_2 < i_3 < \cdots < i_{k-1} < n$

such that $\quad (0,\ i_1),\ (i_1,\ i_2\ ),\ (i_2,\ i_3), \ldots, (i_{k-1},\ n)$

identifies the k ordered segments. Note that the number of segments, k, is a variable in this problem.

A cost function is defined in terms of the nodes the comprise the segments with $c(i\ ,\ j)$ the cost of the segment starting at node i and terminating at node j. The cost of a solution is the sum of the segment costs

$$\sum_{j=1}^{k} c\left(i_{j-1},\ i_j\right) \text{ where } i_0 = 0 \text{ and } i_k = n.$$

Dynamic Programming Model

The nodes on the line comprise the states of the problem. Formally, we introduce a single state variable $\mathbf{s} = (s)$. The state space has $n + 1$ elements

$$\boldsymbol{S} = \{0, 1, 2, \ldots, n\}.$$

The decision vector $\mathbf{d}(\mathbf{s}) = (d)$ also has a single dimension. Here d is the number of intervals to be included in the segment starting at s. A solution is defined by a sequence of states and decisions. To illustrate, consider the solution shown in Fig. 9. The corresponding path begins at the unique initial state $\mathbf{s}_1 = (0)$. The first decision indicates that three intervals are to be included in the first segment so $\mathbf{d}_1 = (3)$. This moves the process to state $\mathbf{s}_2 = (3)$. The next decision, $\mathbf{d}_2 = (4)$, indicates that four intervals are to be included in the next segment, moving the process to $\mathbf{s}_3 = (7)$. The segment starting at state (7) has three intervals, so $\mathbf{d}_3 = (3)$ and the successor state $\mathbf{s}_4 = (10)$ which is the final state for the path. The sequence of states and decisions defines a path starting at state 0 and hopping through the state space until the final state n is encountered. The general path is

$$\mathbf{P} = (\mathbf{s}_1, \mathbf{d}_1, \mathbf{s}_2, \mathbf{d}_2, \ldots, \mathbf{s}_k, \mathbf{d}_k, \mathbf{s}_{k+1}),$$

where $\mathbf{s}_1 = (0)$ and $\mathbf{s}_{k+1} = (n)$.

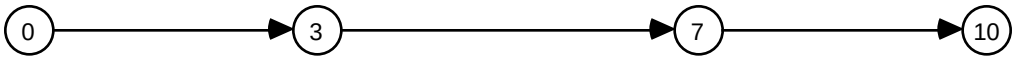


Figure 9. Path through the decision network

The solution given in Fig. 9 defines the path

$$\mathbf{P} = ((0), 3, (3), 4, (7), 3, (10)).$$

Parentheses around alternating elements in $\mathbf{P}$ identify states.

The collection of all possible states and decisions comprise the decision network. Figure 10 depicts all the states but only the decisions starting from state 0. In general, the decision network for this problem will have $n + 1$ nodes and $(n)(n + 1)/2$ arcs. The formal statement of the model is given in Table 13 and includes the parameter m, the maximum number of intervals to be included in a segment. When this maximum is less than n, the number for arcs in the decision network is reduced.

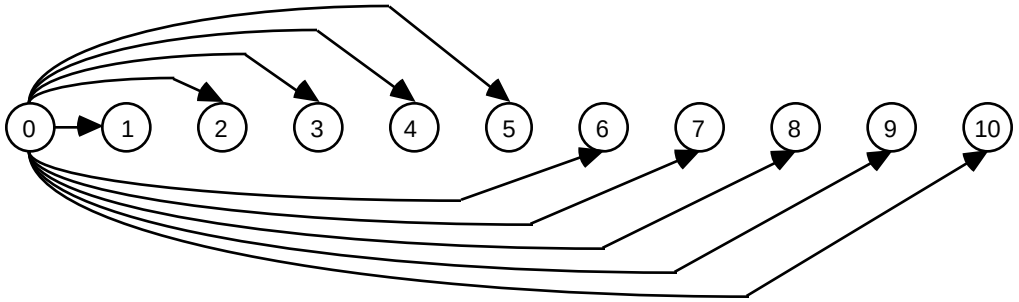


Figure 10. Partial representation of the decision network

Table 13. General model for line partitioning problems

Component	Description
State	$\mathbf{s} = (s)$, where s is a node on the line that begins or ends a segment
Initial state set	$\boldsymbol{I} = \{(0)\}$, the process begins at state 0
Final state set	$\boldsymbol{F} = \{(n)\}$, the process ends at state n
State space	$\boldsymbol{S} = \{0, 1, 2,\ldots,n\}$, the state space has $n + 1$ elements
Decision	$\mathbf{d}(s) = (d)$, the decision is the number of intervals to include in the segment starting at s
Feasible decision set	$\boldsymbol{D}(s) = \{1, 2,\ldots,\text{Min}(m, n - s)\}$ for $s < n$, where m is the maximum number of intervals in a segment. Feasible decisions must remain within the state space.
Transition function	$s' = T(s, d)$, where $s' = s + d$ The new state is the old state plus the number of intervals traversed.
Decision objective	$z(s, d) = c(s, d)$, where the function may be given analytically or in tabular form.
Path objective	Minimize $z(\mathbf{P}) = \sum_{\mathbf{s}\ \boldsymbol{S},\mathbf{d}\ \boldsymbol{D}(\mathbf{s})} z(s, d) + f(\mathbf{s}_f)$
Final value function	$f(\mathbf{s}) = 0$ for all $\mathbf{s}\ \ \boldsymbol{F}$ For specific problems the final value function may be nonzero.

Example 4 - *Line Partitioning*

We wish to divide a 10-inch line into segments whose lengths are in 1-inch multiples. The cost of a segment of length x is

$$c(x) = 10 + x^2$$

The objective is to minimize the total cost of the partition. The parameters that must be specified are n and the cost function $c(s, d)$. For the given problem

$$n = 10 \text{ and } c(s, d) = (10 + d^2), \text{ where } d = s' - s.$$

The optimal solution is shown in Table 14.

Table 14. Optimal solution for line partitioning problem

Index	s	d	$z(\mathbf{P})$	Decision
1	0	3	64	Segment 1: length = 3
2	3	3	45	Segment 4: length = 3
3	6	4	26	Segment 7: length = 4
4	10	—	0	Finished

Example 5 – *Capacity Expansion*

A power company expects a growth in demand of one unit (100 megawatts) of power for each year in the next 20 years. To cover the growth the company will install additional plants with capacities in integer sizes of 1 through 10. The size chosen will determine how many years before the next capacity expansion is required. The cost for the 10 sizes is shown below.

Size	1	2	3	4	5	6	7	8	9	10
Cost	15	16	19	24	30	34	39	45	49	54

The goal is to minimize the present worth of the expansion costs for the next 20 years. We use i to indicate the interest rate for present worth calculations, and assume $i = 5\%$. To illustrate how the objective function is evaluated, say we choose to build plants in the size sequence 4, 6, 5, 5. This means that the expansions occur at times 0, 4, 10 and 15 so the present worth of the costs is

$$z = 24 + \frac{1}{(1+i)^4}(34) + \frac{1}{(1+i)^{10}}(30) + \frac{1}{(1+i)^{15}}(30) = 84.82$$

The situation can be modeled as a line partitioning problem with

$$n = 20,\ m = 10 \text{ and } z(s, d) = \frac{c(d)}{(1+i)^s}$$

where $c(d)$ is given in the cost table as a function of the expansion size d. Note that the discount factor is a function of the state so it is easily included. The solution is displayed Table 16.

Table 16. Optimal solution to capacity expansion problem

Index	s	d	$z(\mathbf{P})$	Decision
1	0	6	83.73	Time 0: size 6
2	6	6	49.73	Time 6: size 6
3	12	4	24.359	Time 12: size 4
4	16	4	10.995	Time 16: size 4
5	20	—	0	Finished

Example 6 – *Production Scheduling*

A manufacturing facility has forecasted demand for the next 20 weeks as shown in the table below. There is a fixed cost for setting up a production run equal to f. In addition, there is a variable cost v that is proportional the number of items produced. If we produce more than the demand in a particular week, the excess items are stored until needed. The inventory cost is proportional to the number of units and the number of weeks stored. The cost per unit per week is w. The problem is to find a production schedule that minimizes the total fixed, variable and inventory costs.

Week	1	2	3	4	5	6	7	8	9	10
Demand	3	1	7	0	0	2	8	2	3	2

Week	11	12	13	14	15	16	17	18	19	20
Demand	9	4	1	8	3	3	8	6	5	5

The appropriate dynamic programming model is similar to that for the line partitioning problem. When applied to the inventory problem, the approach is called the Wagner-Whitin algorithm. The line to be partitioned in this case is the time line. The nodes correspond to the times {0, 1, 2,…,20}. A solution is described by a sequenced set of times at which production occurs: $(0, i_1, i_2, i_3, \ldots, i_{k-1}, n)$. For this case $n = 20$. With the cost computations described above, it can be shown that when production occurs, it is always optimal to produce exactly the quantity demanded for the interval being considered (Dreyfus and Law 1977). Thus production at time 0 satisfies the demands for weeks, 1 through i_1. Production at time i_1 satisfies the demands for times i_1+1 through i_2, and so on.

To express the decision objective in general terms let q_t be the demand in week t, and let $c(s, s')$ be the cost associated with producing at time s to satisfy the demands for periods s+1 to s'. The cost function has three components.

$$z(s, s') = c(s, s') = f + v\sum_{t=s+1}^{s'} q_t + w\sum_{t=s+1}^{s'} (t - s - 1)q_t$$

To illustrate, we calculate $z(s, s')$ for $s = 0$, $d = 6$ and $s' = 6$. The parameter values are $f = 30$, $v = 4$ and $w = 1$.

$$c(0, 6) = f + v\left(\sum_{t=1}^{6} q_t\right) + w\left(0q_1 + 1q_2 + \quad + 5q_6\right) = 30 + 4(13) + 1(25) = 107.$$

Using the line partitioning model with this cost function, the optimal sequence is (0, 6, 10, 13, 16, 20). Table 17 depicts the results. The first five components of this sequence are production times; the corresponding production quantities are 13, 15, 14, 14 and 24, respectively. The total cost is 555.

Table 17. Optimal solution for inventory problem

Index	s	d	$z(\mathbf{P})$	Decision
1	0	6	555	Time 0: Interval 6
2	6	4	448	Time 6: Interval 4
3	10	3	344	Time 10: Interval 3
4	13	3	252	Time 13: Interval 3
5	16	4	157	Time 16: Interval 4
6	20	—	0	Finished

Integer Knapsack Problem

An interesting variation of the line partitioning problem allows the solution of the integer knapsack problem. The mathematical programming model is

$$\text{Maximize } z = \sum_{j=1}^{n} c_j x_j$$

$$\text{subject to} \quad \sum_{j=1}^{n} a_j x_j \le b$$

$$x_j \ge 0 \text{ and integer}, j = 1, \ldots, n$$

where $c_j > 0$ and $a_j > 0$ for all j. Note that the variable x_j is not restricted to binary values but can take on any nonnegative integer value up to $\lfloor b/a_j \rfloor$. To model the problem as a dynamic program we define the state using a single state variable; i.e., $\mathbf{s} = (s)$, where s = amount of resource used by the current solution.

The sequence of decisions to be made is $d_1, d_2, \ldots, d_k$, where d_1 is the index of the first item to be included in the knapsack, d_2 is the index of the second item, and so on. The decision is the index of an item and may

take on the values 1, 2,..., n. Note that different decisions may refer to the same item; e.g., d_2 and d_5 may both refer to item 4. We add one additional possibility denoted by 0 to indicate that no more items are to be included. The complete decision set is

$$\boldsymbol{D} = \{0,\ 1,\ 2,\ldots, n\},$$

and the transition function is

$$s' = s + a_d \text{ for } d > 0$$

$$s' = b \text{ for } d = 0.$$

Thus if the current solution uses s units of the resource and the decision is to bring another item d, the new solution will use a_d additional units. When the constraint coefficients a_j are integer, the states assume integer values between 0 and b. The final transition is associated with the decision $d = 0$ and uses up whatever amount of the resource that remains, so $s_f = b$. If there is no penalty for not completely filling up the knapsack or no benefit for any remaining capacity, then $f(s_f) = 0$.

The decision set for state s consists of any item whose inclusion would not exceed the total capacity b.

$$\boldsymbol{D}(s) = \{0 \text{ or } j\ : s + a_j \le b,\ j = 1,\ldots, n\}.$$

The decision objective $z(\mathbf{s}, \mathbf{d}) = c_d$, where $c_0 = 0$. Thus the integer knapsack problem can be related to the line partitioning problem by viewing the resource b as a line that is being divided successively into segments.

Example 7 – *Unbounded Knapsack*

Consider a single constraint knapsack problem with right-hand side parameter $b = 35$. The values of a_j and c_j are given in the table below. The latter were randomly generated so that no item dominates another. Although no restrictions are placed on the number of items of each type that can be packed, implicit upper bounds exist due to the weight restriction.

Item, j	1	2	3	4	5	6	7	8	9	10
c_j	18	16.6	15	14.8	13.1	11.3	10.5	8.6	6.7	5.2
a_j	15	14	13	12	11	10	9	8	7	6

Table 18 displays the results of the computations and indicates that the optimal policy is to bring two of item 4 and one of item 5. The model has only 36 states. To extend the model to include more than a single resource restriction, additional state variables must be introduced, one for each new constraint.

Table 18. Solution to the integer knapsack problem

Index	s	d	$z(\mathbf{P})$	Decision
1	0	4	42.7	Bring 4
2	12	4	27.9	Bring 4
3	24	5	13.1	Bring 5
4	35	—	0	Final

19.5 Path Problems

When trying to find an optimal path through a network it is natural to use dynamic programming because the optimization problem can be represented explicitly in graphical form. Dijkstra's algorithm introduced in Network Flow Programming Methods chapter for finding the shortest path through a directed network is a typical example. Several variations of the basic problem can also be modeled using dynamic programming. We begin with the grid network depicted in Fig. 11 and define what is called the simple path problem. Nodes represent locations and are identified by their coordinate vector $\mathbf{x} = (x_1, x_2)$ while arcs represent transportation links between nodes. A traveler at a particular node is permitted to move up to the node with the next higher x_2-coordinate (and the same x_1-coordinate) or move right to the node with the next higher x_1-coordinate (and the same x_2-coordinate). The direction traveled will be indicated by the variable d. We take $d = 0$ to mean that travel is up and $d = 1$ to mean that travel is to the right. Clearly, all arcs are one-way. Each arc has a known length given by $a(\mathbf{x}, d)$ where $\mathbf{x}$ describes the node at which the arc begins and d indicates the direction of travel. We assume that the traveler starts at node (1,1) and wants to travel to node (n, n) using the shortest possible route -- the problem objective. The dynamic programming model is straightforward, as defined in Table 19.

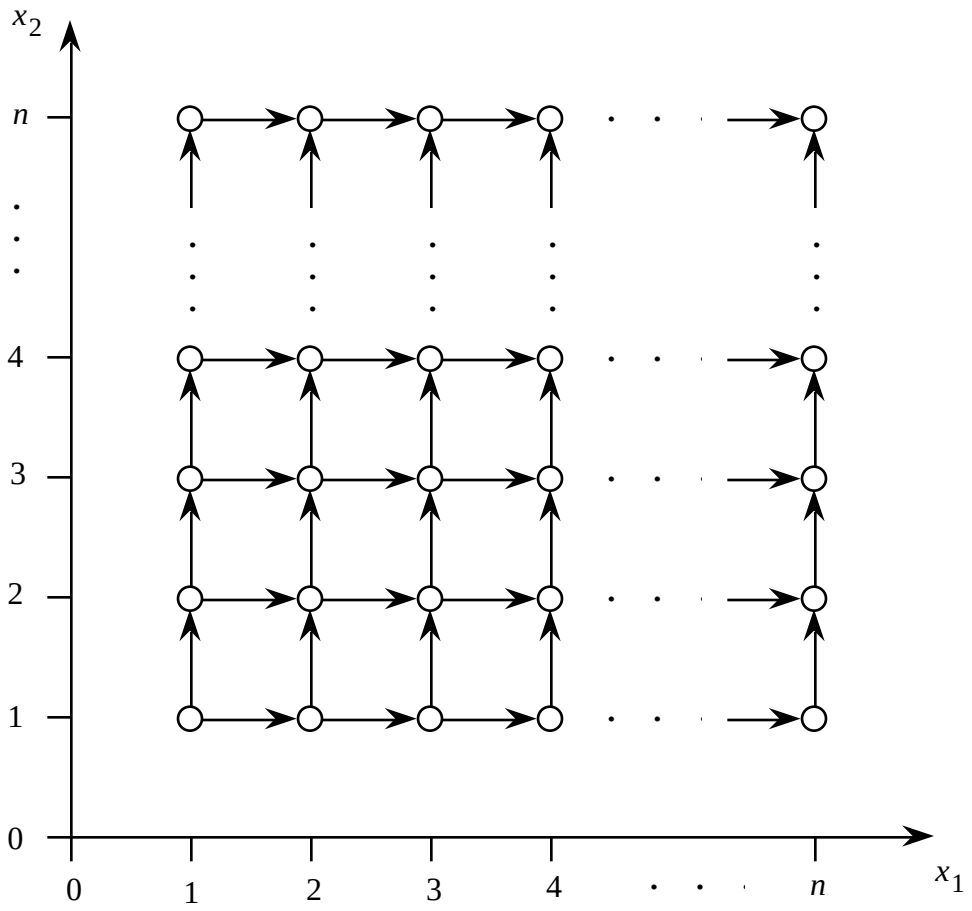


Figure 11. Coordinate representation of path problem

Table 19. General path problem model

Component	Description
State	$\mathbf{s} = (s_1, s_2)$, where $s_1 = x_1$-coordinate $s_2 = x_2$-coordinate
Initial state set	$\boldsymbol{I} = \{(1, 1)\}$
Final state set	$\boldsymbol{F} = \{(m, n)\}$, we generalize the model to allow n rows and m columns.
State space	$\boldsymbol{S} = \{\mathbf{s} : 1 \le s_1 \le m, 1 \le s_2 \le n, s_1 \text{ and } s_2 \text{ integer}\}$
Decision	$\mathbf{d} = (d)$, where d indicates the direction traveled $d = \begin{cases} 0, & \text{go up one node} \\ 1, & \text{go to right one node} \end{cases}$
Feasible decision set	$\boldsymbol{D}(\mathbf{s}) = \{0, 1: s_1 + d \le m \text{ and } s_2 + (1 - d) \le n\}$
Transition function	$\mathbf{s}' = T(\mathbf{s}, \mathbf{d})$ $s_1' = s_1 + d,\ s_2' = s_2 + 1 - d$
Decision objective	$z(\mathbf{s}, \mathbf{d}) = a(\mathbf{s}, d)$
Path objective	Minimize $z(\mathbf{P}) = \sum_{\mathbf{s} \in \boldsymbol{S}, \mathbf{d} \in \boldsymbol{D}(\mathbf{s})} z(\mathbf{s}, \mathbf{d}) + f(\mathbf{s}_f)$
Final value function	$f(\mathbf{s}) = 0$ for $\mathbf{s} \in \boldsymbol{F}$

Example 8 – *Grid Problem*

As an example, consider a 10 × 10 grid with arc lengths as follows

$$a(\mathbf{s}, d) = |s_1 - s_2| \text{ for } d = 0, 1$$

where s_1 and s_2 are the coordinates of the current node. Thus arcs that originate at nodes along the main diagonal have length 0, arcs that originate at nodes one removed from the main diagonal have length 1, and so on. The optimal path for a grid with $n = 10$ is shown in Fig. 12 where the numbers in parentheses along the arcs are their lengths. The path has a total length of 9. Of course, this is not a surprising solution given the arc length definition; however, dynamic programming does not take advantage of

symmetry. The solution is obtained as easily for arbitrary arc lengths as for this special case.

There are many ways to find solutions to the shortest path problem. The linear programming model for this example consists of 180 variables and 100 constraints. The dynamic programming model correspondingly has 100 states in the state space, and 180 arcs, all of which have to be considered in the solution process.

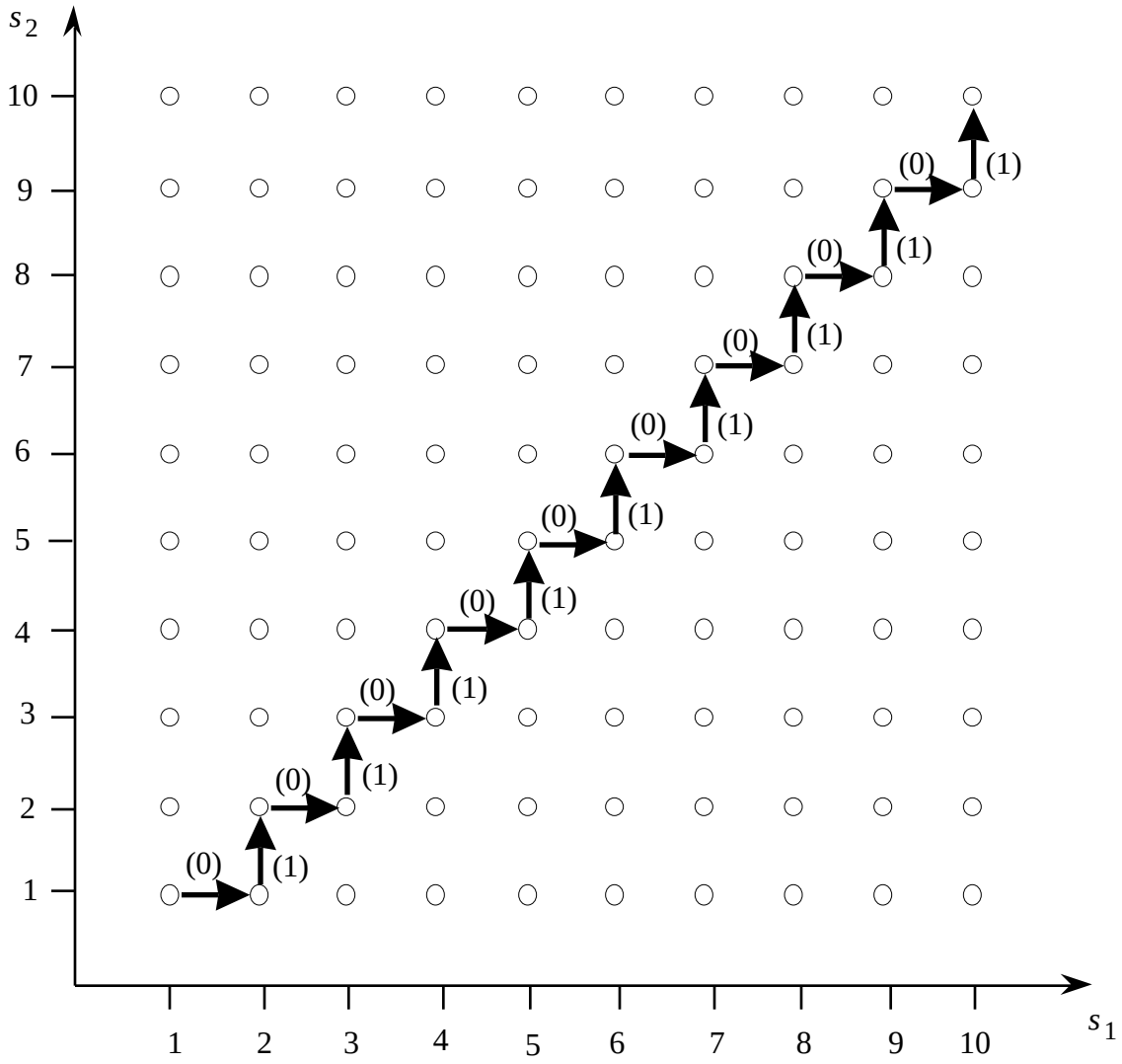


Figure 12. Shortest path from (1, 1) to (10, 10)

Turn Penalties

There are a number of variations of the simple path problem that illustrate the power of dynamic programming. For instance, consider the case where a penalty p_1 is assessed for turning left and a penalty p_2 for turning right. To evaluate a movement from one state to the next, we need to know the last direction traveled as well as the location. This model, given in Table 20, requires an additional state variable to indicate the direction last traveled.

Table 20. General model for the path problem with turn penalties

Component	Description
State	$\mathbf{s} = (s_1, s_2, s_3)$, where $s_1 = x_1$-coordinate $s_2 = x_2$-coordinate s_3 = direction last traveled
Initial state set	$\boldsymbol{I} = \{(1, 1, 0), (1, 1, 1)\}$
Final state set	$\boldsymbol{F} = \{(n, n, 0), (n, n, 1)\}$
State space	$\boldsymbol{S} = \{\mathbf{s} : 1 \le s_1 \le n, 1 \le s_2 \le n,\ s_1 \text{ and } s_2 \text{ integer}, s_3 = 0, 1\}$
Decision	$\mathbf{d} = (d)$, where d indicates the direction traveled
	$d = \begin{cases} 0, & \text{go up one node} \\ 1, & \text{go to right one node} \end{cases}$
Feasible decision set	$\boldsymbol{D}(\mathbf{s}) = \{0, 1 : s_1 + d \le n \text{ and } s_2 + (1 - d) \le n\}$
Transition function	$\mathbf{s}' = T(\mathbf{s}, \mathbf{d})$ $s_1' = s_1 + d,\ s_2' = s_2 + 1 - d$ and $s_3 = d$
Decision objective	$z(\mathbf{s}, d) = a(\mathbf{s}, d)$ for $s_3 = d$ $z(\mathbf{s}, d) = a(\mathbf{s}, d) + p_1$ for $d = 0$ and $s_3 = 1$ $z(\mathbf{s}, d) = a(\mathbf{s}, d) + p_2$ for $d = 1$ and $s_3 = 0$
Path objective	Minimize $z(\mathbf{P}) = \sum_{\mathbf{s} \in \boldsymbol{S}, \mathbf{d} \in \boldsymbol{D}(\mathbf{s})} z(\mathbf{s}, \mathbf{d}) + f(\mathbf{s}_f)$
Final value function	$f(\mathbf{s}) = 0$ for $\mathbf{s} \in \boldsymbol{F}$

Example 9 – *Grid Problem with Turn Penalties*

The solution for the example 10 × 10 grid with a left-turn penalty of 10 and right-turn penalty of 5 is shown in the Fig. 13. The solution has migrated away from the diagonal to escape excessive turn penalties. The total arc length is now 27. Adding to this 10 for each turn gives a total path value of 47.

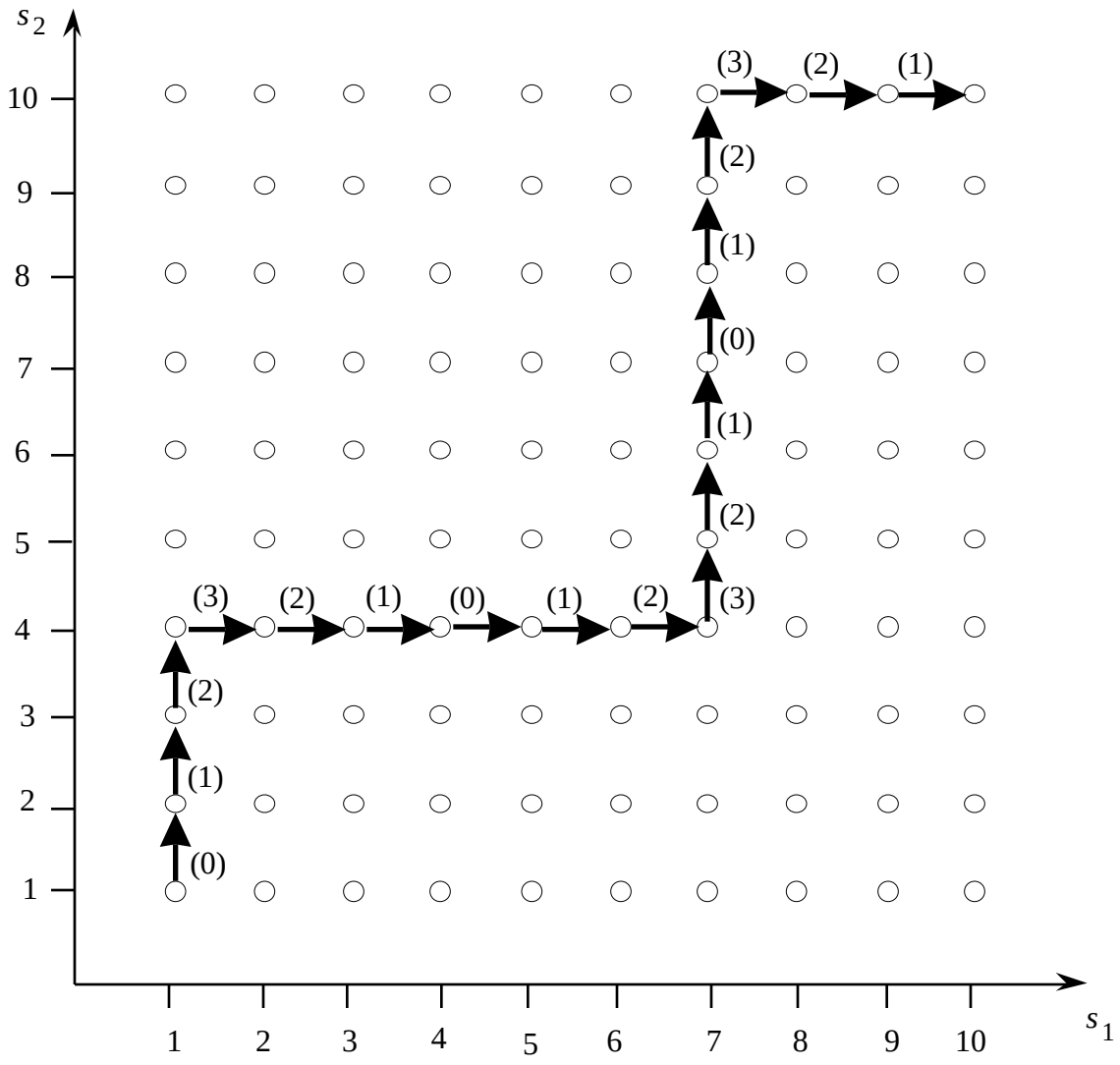


Figure 13. Path solution with turn penalties

The turn penalty problem is an example of the usefulness of dynamic programming. The new model has 200 states and each arc is considered twice in the solution process. An integer programming model for the problem would be considerably more complicated.

19.6 Sequencing Problems

Many operational problems in manufacturing, service and distribution require the sequencing of various types of activities or items. Examples include a production facility in which chassis must be sequenced through an assembly line, an express mail service where parcels and letters must be routed for delivery, and a utility company that must schedule repair work. In general, problems in this class are easily formulated as mathematical programs but with a few exceptions owing to special structure, are difficult to solve. In this section, we introduce a robust dynamic programming formulation that can be used to tackle a number of such problems. In most cases, however, the size of the state space is an exponential function of the number of items being sequenced. In practical instances, the success of the DP approach may depend on our ability to reduce the number of states that must be explored in the search for the optimum. One way to do this is to impose precedence requirements on the items to be sequenced; a second way is to introduce the logic of branch and bound within a dynamic programming algorithm.

Single Machine Scheduling

As a prototype, consider the problem of sequencing a set of n jobs through a single machine that can work on only one job at a time. Once a job is started, it must be completed without preemption. The time required to process job j once the machine begins to work on it is $p(j)$ for $j = 1, \ldots, n$. The associated cost $c(j, t)$ is a function of its completion time t and can take a variety of forms, the simplest being

$$c(j, t) = a(j)t$$

where $a(j)$ is the cost per unit time for job j. As we saw in the first section of the Integer Programming Methods chapter on greedy algorithms, this form admits a very simple solution when the objective is to minimize the total completion cost of all the jobs. The optimum is obtained by computing the ratio $p(j)/a(j)$ for each job and then sequencing them in order of increasing values of this ratio. The job with the smallest ratio is processed first, the job with next smallest ratio is processed second, and so on until all jobs are completed. Ties may be broken arbitrarily.

A much more difficult problem results when each job j has a due date $b(j)$. The cost of a job is zero if it is completed before its due date but increases linearly if it is tardy.

$$c(j, t) = \begin{cases} 0 & \text{for } 0 \le t \le b(j) \\ a(j)(t - b(j)) & \text{for } t > b(j) \end{cases}$$

The goal of the optimization is to determine the sequence that has the smallest total cost. Table 21 gives the relevant parameters for a 4-job instance. There are 4! = 24 possible solutions. For the solution (3, 1, 2, 4), the completion times are 7, 12, 21 and 31 respectively. Jobs 3 and 1 are completed before their due date so no cost is incurred. Job 2 is 11 days late resulting in a cost of $440 and job 4 is 14 days late resulting in a cost of $420. The total cost is therefore $860.

Table 21. Job parameters for a sequencing problem

Job j	Processing time $p(j)$	Due date $b(j)$	Cost per day $a(j)$
1	5	12	\$80
2	9	10	40
3	7	10	100
4	10	17	30

In general, we write a sequence as a vector $(j_1, j_2, j_3, \ldots, j_n)$ which implies that job j_1 is processed first, job j_2 second and so on until the final job j_n. This vector is a permutation of the integers 1 through n and admits $n!$ possible sequences, a number that increases rapidly with n. In fact, it is not possible to find a polynomial function of n that provides a bound on how fast $n!$ grows.

The time at which a job is finished is determined by its place in the sequence. Job j_k is in position k and is not started until the previous $k - 1$ jobs finish processing. It ends at time $t(j_k)$, the sum of the processing times of the previous jobs plus its processing time $p(j_k)$.

$$t(j_k) = \sum_{i=1}^{k} p(j_i)$$

The cost associated with a particular sequence is the sum of the individual job costs as determined by their completion times. The objective function is then

$$z = \sum_{j=1}^{n} c(j, t(j))$$

and the goal is to minimize z.

To solve this problem with dynamic programming, we must first describe it as a sequential decision process. In this case, the description is once again straightforward with the decisions corresponding to places in the sequence. Thus the decision at each step is a job number. The DP model is given in Table 22.

Table 22. General sequencing problem

Component	Description
State	To determine the state definition, consider the information necessary to specify the set of feasible decisions and to evaluate the cost associated with a decision. At a particular step in the sequence, a job is a feasible choice if it hasn't been chosen before. Thus the minimal information the state must provide is the set of jobs previously included in the sequence. This also is the information necessary to compute the time of completion of the job and hence the associated cost. To describe the state we need a vector with n components $\mathbf{s} = (s_1, s_2, \ldots, s_n)$, where $s_j = \begin{cases} 0 & \text{if job } j \text{ has not been included in the sequence} \\ 1 & \text{if job } j \text{ has already been included in the sequence} \end{cases}$
Initial state set	$\boldsymbol{I} = \{(0, 0, \ldots, 0)\}$ None of the jobs has been scheduled.
Final state set	$\boldsymbol{F} = \{(1, 1, \ldots, 1)\}$ All jobs are scheduled.
State space	The state vector is a binary vector with n components. Therefore, there are 2^n members of the state space representing all possible combinations. $\boldsymbol{S} = \{\mathbf{s} : s_j = 0 \text{ or } 1,\ j = 1, \ldots, n\}$
Decision	The decision vector has a single component that identifies the next job to be processed. $\mathbf{d} = (d)$, where d = the next job in the sequence
Feasible decision set	The feasible decisions at a given state are the jobs not already chosen. $\boldsymbol{D}(\mathbf{s}) = \{j : s_j = 0,\ j = 1, \ldots, n\}$
Transition function	The transition function changes the state to reflect the inclusion of an additional job in the sequence. $\mathbf{s}' = T(\mathbf{s}, \mathbf{d})$, where $s'_d = 1$ and $s'_j = s_j$ for $j \neq d$

Decision objective	$z(\mathbf{s}, \mathbf{d}) = c(d, t)$, where $t = \sum_{j=1}^{n} s_j p(j) + p(d)$ The cost function is problem dependent. For the job sequencing problem with tardiness penalties we use the cost function defined above.
Path objective	Minimize $z(\mathbf{P}) = \sum_{\mathbf{s}} \sum_{\mathbf{S},\mathbf{d} \in \mathbf{D}(\mathbf{s})} z(\mathbf{s}, \mathbf{d}) + f(\mathbf{s}_f)$
Final value function	$f(\mathbf{s}) = 0$ for $\mathbf{s} \in \boldsymbol{F}$

Example 10 – *Job Sequencing with Tardiness Penalties*

The decision network for the data given in Table 21 is depicted in Fig. 14. The state vectors are shown in parentheses adjacent to the nodes. Arcs represent the transition from one state to the next, and each has an associated cost (not shown). The solution is determined by finding the shortest path through the network and is shown by the heavy lines in the figure. The optimum is the sequence (3, 1, 2, 4) as before.

Although the shortest path problem on an acyclic network can be solved efficiently, the difficulty here is that there are an exponential number of states, 2^n. This means that the DP approach as given in Table 22 does not lead to an efficient solution procedure for most sequencing problems; that is, the amount of computations is not bounded by a polynomial function of n. Because of the large number of states, problems can be solved only for small values of n. For example, a 10-job problem with 1024 states took about 8 minutes on a Macintosh G3 running at 400 Mz.

The state space is considerably reduced if an ordering between some jobs is imposed. For example, if one specifies that job 3 must precede job 1, the number of feasible states is reduced from 16 to 12. Each additional restriction reduces the number of states in some nonlinear fashion.

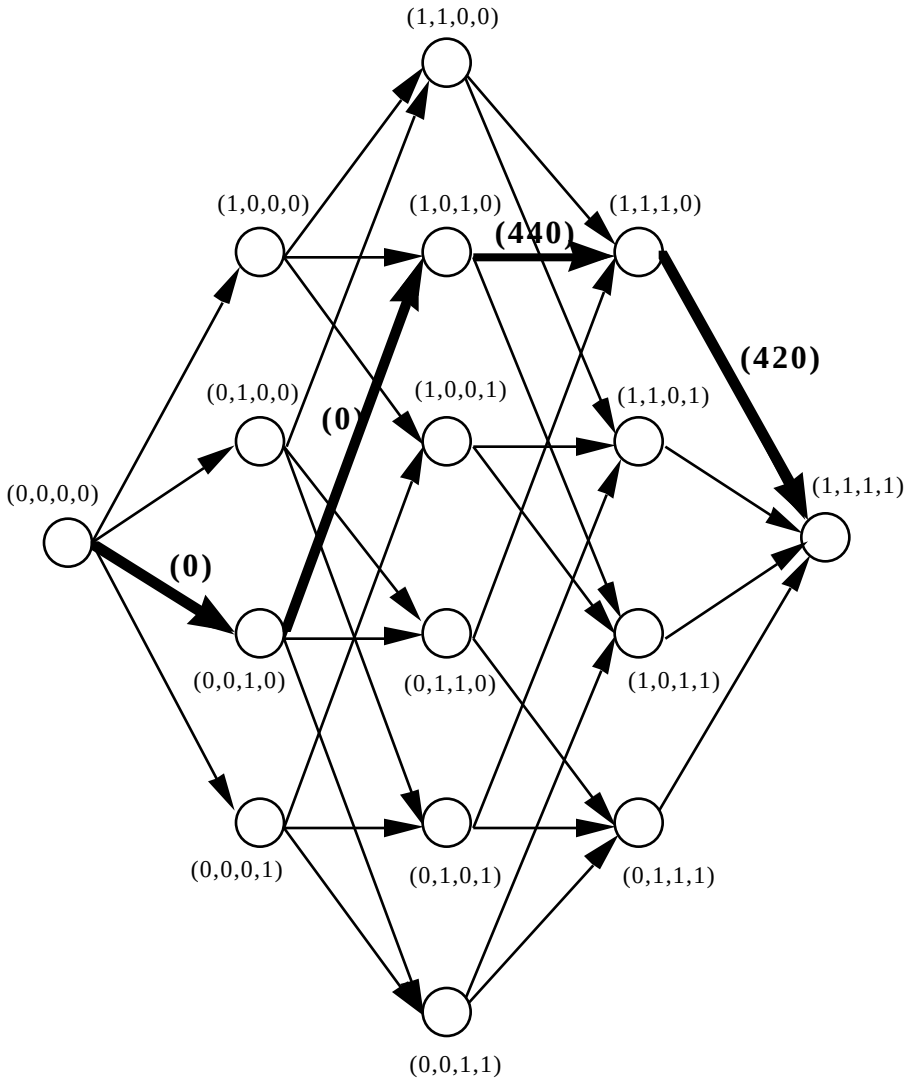


Figure 14. Decision network for 4-job sequencing problem

Traveling Salesman Problem

In our description of the total tardiness problem, the cost associated with a particular job did not depend on its immediate predecessor. There are many situations, though, where these costs are sequence dependent. In manufacturing, for example, it may be necessary to change the tooling between two successive jobs, or in scheduling propane deliveries, the length of the route and hence travel cost depends on the order in which customers are visited. In these cases, it would be necessary to extend the definition of the state space in Table 22 to include an additional state representing the last job processed in the sequence. The traveling salesman problem fits this situation.

Recall that in the TSP a salesman must visit *n* cities starting and ending at his home base. The objective is to minimize some measure of travel cost subject to the restriction that each city be visited once and only

once. A feasible solution is called a tour. We arbitrarily identify city 1 as the home base. Like the sequencing problem, a solution is described by a vector $(1, j_2, j_3, \ldots, j_n)$ which implies that the tour starts at city 1, goes next to city j_2, and so on until the final city j_n is reached. To complete the tour, the salesman must travel from j_n back to city 1. The cost of the tour is

$$z = c(1,j_2) + \sum_{k=2}^{n-1} c(j_k, j_{k+1}) + c(j_n,1)$$

where the function $c(i, j)$ specifies the cost of traveling from city i to city j. When $c(i, j)$ represents the distance between i and j, the objective of the problem is simply to minimize the total distance traveled. For those cases where $c(i, j) = c(j, i)$, the TSP is said to be symmetric; otherwise it is asymmetric.

The dynamic programming model is similar to the sequencing model in that the state identifies the set of cities that have been visited at any point in the tour. To compute the cost of traveling to the next city, though, we need to know the last city visited. An additional state variable is defined for this purpose.

Table 23. Traveling salesman problem

Component	Description
State	$\mathbf{s} = (s_1, s_2, \ldots, s_n, s_{n+1})$, where $s_j = \begin{cases} 0 & \text{if city } j \text{ is not in the sequence} \\ 1 & \text{if city } j \text{ is in the sequence} \end{cases}$ $j = 1,\ldots,n$ s_{n+1} = index of the last city in the sequence
Initial state set	$\boldsymbol{I} = \{(1, 0,\ldots,0, 1)\}$ Only city 1 is in the tour and that is the last city visited.
Final state set	$\boldsymbol{F} = \{(1, 1,\ldots,1, j) : j = 2,\ldots,n\}$ All cities are in the tour. The last city can be any city but 1.
State space	There are 2^{n-1} possible combinations of the first n state variables, since s_1 is fixed as 1. The last state variable can take on up to $n - 1$ values. $\boldsymbol{S} = \{\mathbf{s} : s_1 = 1,\ s_j = 0 \text{ or } 1,\ j = 2,\ldots,n \text{ and } s_{n+1} = 2,\ldots,n\}$ (The actual number of feasible states is about half the cardinality of $\boldsymbol{S}$.)

Decision	The decision vector has a single component that identifies the next city to be included in the tour. $\mathbf{d} = (d)$, where d = the next city in the tour
Feasible decision set	$\boldsymbol{D}(\mathbf{s}) = \{j : s_j = 0,\ j = 2, \ldots, n\}$ The feasible decisions at a given state are the cities not yet visited.
Transition function	The transition function changes the state to reflect the inclusion of an additional city in the tour. The last state variable becomes the decision. $\mathbf{s}' = T(\mathbf{s}, \mathbf{d})$, where $s'_d = 1$, $s'_j = s_j$ for $j \neq d$, and $s_{n+1} = d$
Decision objective	$z(\mathbf{s}, \mathbf{d}) = c(s_{n+1}, d)$ where $c(\cdot, \cdot)$ is defined for all city pairs.
Path objective	Minimize $z(\mathbf{P}) = \sum_{\mathbf{s}\ \mathbf{S},\mathbf{d}\ \boldsymbol{D}(\mathbf{s})} z(\mathbf{s}, \mathbf{d}) + f(\mathbf{s}_f)$
Final value function	$f(\mathbf{s}) = c(s_{n+1}, 1)$ for $\mathbf{s}$ $\boldsymbol{F}$ This function is the cost of traveling from the last city to city 1.

To determine the actual number of feasible states it is necessary to examine the model used in the solution process. For one particular asymmetric formulation, Dryfus and Law [1977] show that when $\sum_{j=2}^{n} s_j = i$, there are $(n-1)\binom{n-2}{i}$ different states ($i = 1, \ldots, n-2$) in the recursion. In addition, there are $n-1$ states that are not evaluated recursively but are associated with boundary conditions. This gives an approximate total of $(n-1)(2^{n-2}-1) + n - 1 = (n-1)2^{n-2}$ states. For the symmetric case, the direction of the tour is immaterial so about half the number of states is required.

Example 11 – *Traveling Salesman Problem*

Consider an 8-city problem on a square grid with the coordinates assigned randomly in the range 0 to 25. The following matrix shows the locations of the cities in the (x, y)-plane.

	x	y
1	7.0	9.2
2	20.0	9.3
3	20.6	15.3
4	9.0	7.5
5	6.6	13.7
6	4.2	5.2
7	4.3	4.7
8	13.9	12.7

For the cost function we use the p-norm distance between a city pair given by

$$c(i, j) = \left(|x_i - x_j|^p + |y_i - y_j|^p \right)^{1/p} .$$

When $p = 2$, this function gives the Euclidean distance between the two points; when $p = 1$, the function gives the rectilinear distance. Other values are possible. For the example, we used $p = 2$.

The dynamic programming model of the problem has 449 states. The optimal solution is shown in Fig. 15. The effort required to solve the problem is primarily influenced by the number of states. It is possible to reduce this number if precedence relations can be specified between city pairs.

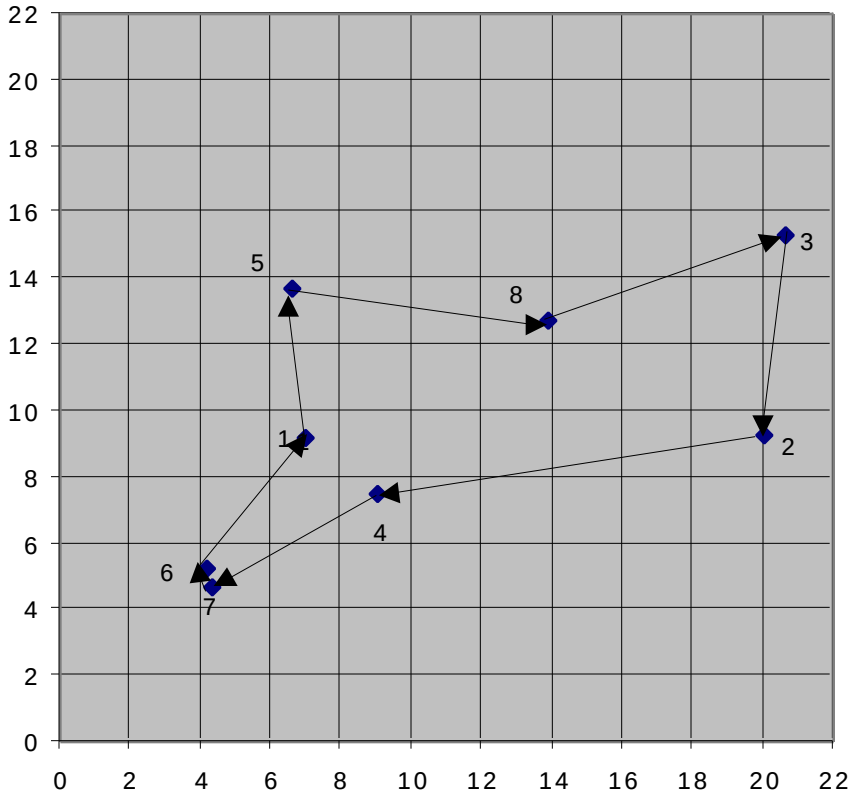


Figure 15. Optimal solution to TSP

19.7 Exercises

Numerical problems in this chapter can be solved with the Teach DP Excel add-in.
The first seven exercises refer to the investment example in Section 19.1.

1. Provide an integer linear programming model for this problem.

2. Assuming a budget of 10, use the dynamic programming model to show the sequence of states and decisions and the path objective values for each of the following investment decisions. The decisions are given as vectors with the investment in the *i*th opportunity shown as the *i*th component of the vector.

   a. (0, 0, 0, 0, 0, 0, 0, 0)

   b. (1, 2, 0, 3, 1, 0, 1, 2)

   c. (0, 3, 3, 0, 0, 0, 3, 0)

   d. (0, 0, 2, 4, 2, 1, 1, 0)

3. For a budget of 10, try to find the optimal investment plan by observation. What was your reasoning?

4. Let the final value function $f(\mathbf{s}) = 2(10 - s_2)$. This represents the value of any remaining funds at the final state $\mathbf{s}_f = \mathbf{s}_9$. Reevaluate each of the decision sets given in Exercise 2 using this function.

5. For the decision network associated with this problem, find the number of nodes, number of arcs, and number of feasible paths as a general function of n and b.

6. Let the second state variable be defined as follows.

   $$s_2 = \text{amount of the budget not yet spent}$$

   Describe initial states, final states, and the transition function for this modification.

7. What modifications to the DP model are necessary if a constraint is added that requires investment in at least 5 alternatives?

8. How would the model for the knapsack problem in Section 19.3 change with the following variations. Each part should be done separately rather than cumulatively.

   a. It may be preferable not to fill the knapsack to capacity. Let y be the difference between the capacity W and the weight associated with the solution, and let $r(y)$ be its corresponding value.

   b. The entire capacity of the knapsack must be used.

   c. Up to u_j units of item j are available, $j = 1, \ldots, n$, and may be packed as long as the weight constraint is not violated. Under what conditions can the integer knapsack model be used for this problem?

9. For the line partitioning problem described in Example 4 in Section 19.4, give the path and path objective values for the following situations.
   a. The solution has only a single segment.
   b. The solution has 10 segments.
   c. The first two segments have length 2, and the next two segments have length 3.

10. The table below gives the benefits and weights associated with items that might be included in a knapsack whose maximum capacity is 19 lbs. Using the dynamic programming model, show the path and path objective associated with each of the following.
    a. Include one of item 1 and one of item 2.
    b. Include as many as possible of item 4.
    c. The best solution you can find by observation.
    d. Find the optimal solution with the Teach DP add-in.

Item	1	2	3	4	5
Benefit	20	15	10	5	3
Weight	10	7	6	4	2

For Exercises 11 – 15, give the DP model for the following variations of the path problem.

11. You are allowed to reduce the length of one arc of your choice to zero.

12. A toll in the amount of $c(\mathbf{x}, d)$ dollars is charged for each arc traversed. You can't spend more than b dollars on tolls.

13. You want to minimize the length of the longest arc on your path.

14. You can change direction on the solution path at most w times.

15. Once you change direction, you can't change direction again until you have traversed two arcs in the new direction.

16. Find the optimal sequence when the due dates for the jobs in Table 21 are changed to 0.

17. Using the data in Table 21, show the sequence of states and decisions and the path objective values for the following sequences:
    a. (1, 2, 3, 4)
    b. (3, 1, 4, 2)
    c. The sequence found by the greedy algorithm when due dates are not specified.
    d. The best sequence you can find by observation.

e. The optimal sequence found with DP software.

18. Give a DP formulation for the following variations of the path problem.

a. You make only every other decision (beginning with the first). Your spouse, whose goal is to maximize trip length, makes the alternate decisions.

b. You terminate your trip whenever the x_1-coordinate reaches n; however, you must pay a penalty equal to $w(n - x_2)$ if you don't finish at $\mathbf{x} = (n, n)$.

c Assume the network in Fig. 6 represents a maze in an adventure game. Rather than a length, the quantity $a(\mathbf{x}, d)$ represents the probability that you will be killed by an evil force if you travel that arc. Find the route that maximizes your probability of survival.

d. Assume that the network represents alternative routes for a proposed road through a mountain range. There is an additional quantity $b(\mathbf{x}, d)$ associated with each arc that represents the amount of dirt that must be removed or added to the road link to bring it to a specified height above sea level. A positive value of $b(\mathbf{x}, d)$ is the amount that must be added and a negative value is the amount that must be removed. An unlimited quantity of dirt can be obtained at node (1, 1) or disposed of at node (n, n); quantities removed on one link can be deposited on another. Dirt will be moved along the selected arcs but only in the directions indicated by the arcs. The cost of moving dirt on an arc is linear with unit cost $a(\mathbf{x}, d)$. These costs are also incurred on the links where the dirt is removed or deposited. What is the route that minimizes the dirt moving costs?

19. (*Elevator Problem*) A 20 floor building has three elevators. During the morning rush hour they are operated so that each serves a contiguous set of floors, and no two serve the same floor. The problem is to determine which floors are to be served by each elevator. The time it takes for an elevator to travel between two levels that are k floors apart is $15 + 5k$ seconds. The population of each floor is given in the table below.

Floor	2	3	4	5	6	7	8	9	10
Population	30	40	70	20	10	30	50	80	60

11	12	13	14	15	16	17	18	19	20
20	10	20	40	60	80	70	60	30	70

The objective is to minimize the fill time — the time required to bring all the people to their floors. The elevator can hold up to 20 people. Assume that in each run the elevator must stop at all floors to which it is assigned. Set up a dynamic programming model as a line partitioning problem and solve it with the Teach DP add-in.

20. (*Inspection Station Problem*) A manufacturing process consists of a series of operations through which each product must pass in the same order. The operations are numbered 1 to n. Each operation ruins a fixed percent of the products that pass through it. Data for an example are given in the table.

Operation	1	2	3	4	5	6	7	8
Cost/unit processed, $	5	10	8	15	3	20	7	10
Percentage ruined, %	1	2	1	3	1	2	3	1

Ruined products can't be identified except by a careful inspection. The problem is to determine where along the line to include inspection stations. One must be placed at the end of the line but otherwise, they can be placed after any operation. When a ruined product is detected, it is discarded and has no scrap value. Inspection stations cost $1000 per year to run. Find the optimal solution (number and location of stations) if the annual production rate is: (a) 1000 units, (b) 2000 units, and (c) 3000 units. (Set up and solve the DP for this problem for one year).

21. (*Machine Overhaul Problem*) The service of a particular type of machine is needed for the next n years. At present ($t = 0$) the machine is new. As it ages, its operating costs increase so it may be advisable to buy a new machine prior to the end of its useful life, or have the existing one overhauled. There is not limit to the number of overhauls and no degradation in performance. The cost of overhauling a machine of age t (or t years after the last overhaul) is $O(t)$. The cost of operating an overhauled machine for one year ending t years after overhaul is $C_O(t)$. The trade-in value of an overhauled machine t years after overhaul is $T_O(t)$

    The cost of purchasing a new machine is P. A machine that has never been overhauled is called an original machine. The cost of operating an original machine for one year ending t years after purchase is $C_N(t)$. The trade-in value of an original machine t years after purchase is $T_N(t)$. The salvage value at the end of the n year period obeys the same function as the trade-in value.

    The problem is to find the optimal purchase/overhaul policy over the n-year horizon. The decision to be made at the beginning of each year is whether to purchase a new machine or overhaul the current machine, and how many years until the next purchase or overhaul. The decision vector should have two components.

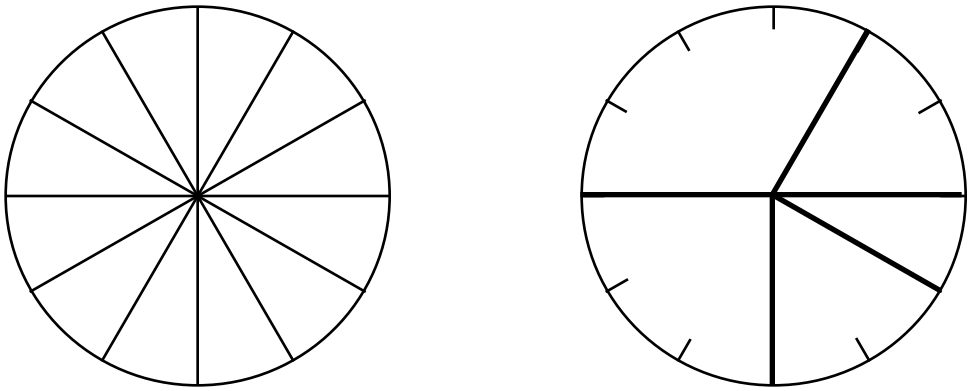
    a. Formulate the problem as a dynamic program.

    b. How would you modify this formulation to account for the time value of money. Assume a constant discount rate i.

22. (*Circle Search Problem*) Consider the problem of locating an item that is known to be lost in an area defined by a circle with a one-mile radius. The circle has been divided into 12 equal segments as shown in part *a* of the figure below. Using various evidence, it is determined that the probability that the item is in segment i is P_i for $i = 1,\ldots,12$. Ten teams are available to search the area. From past experience it has also been determined that if an item is in an area of size A the probability that j teams searching together will find it in the allowed time is

$$P_{\text{Detect}} = e^{-k_1 A}\left(1 - e^{-k_2 j}\right)$$

    where k_1 and k_2 are positive known constants. This is the probability of detection *given* that the item is in the area searched.



a. Circle with 12 segments b. Circle with segments allocated to search areas

Two versions of circle search problem

Set up the dynamic programming formulations for parts *a* and *b* below. In each case the goal is to maximize the probability that the item will be found.

a. The solution should indicate how many teams should be assigned to each segment. A team can only be assigned to one segment.

b. A search area is defined to be one or more contiguous segments as illustrated in part *b* of the figure. Search areas do not overlap. The solution should indicate how the circle should be divided and how many teams should be assigned to each search area.

c. Let $k_1 = 0.04$ and $k_2 = 0.7$. The probabilities P_i are given below. For the problem in part *a*, show the path through the state space and evaluate the solution when one team is assigned to segments 1 and 5, and two teams are each assigned to segments 4, 8, 9 and 10.

i	1	2	3	4	5	6	7	8	9	10	11	12
P_i	0.1	0	0.05	0.15	0.1	0.05	0	0.15	0.2	0.15	0.05	0

d. Solve the problem using the data in part *c*.

23. (*Pumps for a Pipeline*) An oil distribution company is constructing an 800 mile pipeline across Texas. The oil is to flow from west to east. The pressure at the west end is fixed at 100 psi (pounds per square inch). Because of losses due to friction the pressure drops at a rate of 1 psi for each mile of pipe. Pressure at the east end of the pipe is required to be 50 psi. To make up for the losses, pump stations are to be constructed at intervals along the pipe. The pressure must never fall below 30 psi. Pump stations have a fixed cost of $10,000 and a variable cost that depends on the pressure rise provided by the pump, as given in the following table.

Increase in pressure (psi)	10	20	30	40	50	60	70
Variable cost ($1000)	3	6	8	10	11	13	17

The problem is to determine the minimum cost policy of installing pumps. Write out the DP formulation for the following two cases.

a. First assume that a pump will always increase the line pressure to 100 psi at the point where it is located.

b. Alternatively, do not require the condition of part *a.*

c. Solve the problems in parts *a* and *b* using DP software.

24. (*Traveling Salesman Problem - TSP*) The table below gives the cost $c(i, j)$ of going from city i to city j. The values of $c(i, j)$ and $c(j, i)$ are not necessarily equal. Propose a greedy algorithm to find a solution and then write out the corresponding sequence of states and decisions. Solve the TSP with the Teach DP add-in and compare the optimal cost with the cost given by your greedy algorithm.

Cost matrix for TSP

	From city					
To city	1	2	3	4	5	6
1	—	27	43	16	30	26
2	7	—	16	1	30	30
3	20	13	—	35	5	0
4	21	16	25	—	18	18
5	12	46	27	48	—	5
6	23	5	5	9	5	—

25. (*Capacity Expansion Problem*) A city expects the following annual growth in electricity demand (MW) during the next 10 years.

Year	1	2	3	4	5	6	7	8	9	10
Growth (MW)	2	3	1	5	2	3	4	3	2	1

To meet this demand, additional generating capacity must be installed. Construction costs as a function of size (MW) are given in the following table.

Size (MW)	1	2	3	4	5
Cost, $M	20	38	55	70	80

The discount rate for time value of money calculations is 10%. Assume that capacity can be installed instantaneously and that there must always be sufficient capacity to meet demand. Set up and solve the dynamic programming model to find the capacity expansion policy that will minimize the present worth of construction costs.

26. (*Road Repair Problem*) A repair policy is to be determined for a major highway for the next 10 years. After that time, the highway will be completely rebuilt. There are two types of repairs that can be performed: the first will be referred to as a long term fix,

and the second as a short term fix. Relevant parameters are given in the table below. In addition to the cost of repair, it is necessary to factor in an annual maintenance cost that depends on the type of repair last done. The current time is 0 and the road must now be repaired in some fashion. The short-term fix cannot be done two times in succession. Give the dynamic programming formulation that will determine a policy that minimizes the total cost over the 10 year planning horizon. All costs are in thousands of dollars. Setup and solve your model using the Excel add-in.

Type of repair	Long term	Short term
Cost of repair	1500	400
Annual cost of maintenance	50	100
Time until next repair	5	2

27. (*Production Scheduling over a Finite Horizon*) Use the approach described for Example 6 in Section 19.4 to solve the production scheduling problem for the demand data given below. Assume that the fixed charge per order is \$100 and that the inventory holding cost per unit per week is \$1. State your solution as a path through the state space.

Period	1	2	3	4	5	6	7	8	9	10
Demand	15	20	5	40	25	4	15	10	40	20

28. (*Soot Collection*) A pollution control device in a smoke stack collects soot (particulate matter from combustion). The amount of soot deposited varies from month to month due to furnace usage, as indicated in the table below. The device must be cleaned occasionally to remove the soot. It may be cleaned at the end of any month for a fixed cost of \$100.

    The stack does not operate as efficiently when there is soot in the device as when it is clean. If at the beginning of any month the amount of soot present is y, the increase in operating cost associated with the soot is $20y$. At most 10 units of soot is permitted to be in the device at the beginning of any month.

    Set up the dynamic programming model that can be used to solve the problem of determining the optimal cleaning schedule for a 10-month period. Assume that we are at the beginning of month 1 and that the device is clean. In addition to other times that might be selected, the device must be cleaned at the end of the 10th month. Use the Excel add-in to solve the problem.

Month	1	2	3	4	5	6	7	8	9	10
Soot	5	3	2	8	6	2	3	6	5	3

29. (*Optimal Redundancy Problem*) An electronic system has n components. The reliability of a component is the probability that it will not fail during operation. The reliability of component i is given as r_i. The reliability of the system is the probability that none of its components fail. This is computed as the product of the component reliabilities.

To increase the reliability of the system, extra units may be included as backups for the original components. These are called redundant components since they are not required unless the originals fail. Assume a component of type i costs c_i dollars. The probability that a collection of x components of the same type fails is the probability that they all fail. Thus the reliability of a component type with x redundant units is

$$R_i = 1 - (1 - r_i)^{(1+x)}$$

The reliability of the system, R_S, is the product of the component type reliabilities; i.e., $R_S = \prod_{i=1}^{n} R_i$. We want to determine how many redundant components of each type to provide without exceeding the available budget b.

a. Formulate this problem as a dynamic program.

b. Explain how you would incorporate additional system constraints such as weight and power limits.

c. For the data given below, describe the state space for the problem. Use the Excel add-in to solve the problem. Show the optimal path through the state space by listing the sequence of states and decisions. The amount spent on redundant components should be no more than \$500.

Item, i	1	2	3	4
Reliability, r_i	0.9	0.8	0.95	0.75
Cost/item, c_i	\$100	\$50	\$40	\$200

Bibliography

Bard, J.F., "Short-Term Scheduling of Thermal-Electric Generators Using Lagrangian Relaxation," *Operations Research*, Vol. 36, No. 5, pp. 756-766, 1988.

Bard, J.F. and W.A. Bejjani, "Designing Telecommunications Networks for the Reseller Market," *Management Science*, Vol. 37, No. 9, pp. 1125-1146, 1991.

Beasley, J.E. and B. Cao, "A Dynamic Programming Based Algorithm for the Crew Scheduling Problem," *Computers & Operations Research*, Vol. 25, No. 7/8, pp. 567-582, 1998.

Bellman, R.E. and S.E. Dreyfus, *Applied Dynamic Programming*, Princeton University Press, Princeton, NJ, 1962.

Chatwin, R.E., Multiperiod Airline Overbooking with a Single Fare Class," *Operations Research*, Vol. 46, No. 6, pp. 805-819 1998.

Denardo, E.V., *Dynamic Programming: Models and Applications*, Prentice Hall, Engelwood Cliffs, NJ, 1982.

Dreyfus, S.E. and A.M. Law, *The Art and Theory of Dynamic Programming*, Academic Press, New York, 1977.

Edwards, D.M., R.D. Shachter and D.K. Owens, "A Dynamic HIV-Transmission Model for Evaluating the Costs and Benefits of Vaccine Programs," *Interfaces*, Vol. 28, No. 3, pp. 144-166, 1998.

Hark, H., and U.S. Ji, "Production Sequencing Problem with Reentrant Work Flows and Sequence Dependent Setup Times," *Computers & Industrial Engineering*, Vol. 33, No. 3/4, pp. 773-776, 1997.

Hodgson, T.J., G. Ge, R.E. King and H. Said, "Dynamic Lot Size/Sequencing Policies in a Multi-Product, Single-Machine System," *IIE Transactions on Scheduling & Logistics*, Vol. 29, No. 2, pp. 127-137, 1997.

Johnson, L.A. and D.C. Montgomery, *Operations Research in Production, Planning, Scheduling, and Inventory Control*, John Wiley & Sons, New York, 1974.

Khater, M., "Optimal Packing of Transformer Coil via Dynamic Programming," *Computers & Industrial Engineering*, Vol. 35, No. 3/4, pp. 447-450, 1998.

Peters, E.R., "A Dynamic Programming Model for the Expansion of Electric Power Systems," *Management Science*, Vol. 10, pp. 656-664, 1973.

Wagner, H. and T. Whitin, "Dynamic Version of the Economic Lot Size Model," *Management Science*, Vol. 5, pp. 89-96, 1958.