

Contents

10 Dynamic Programming (DP)	511
10.1 Sequential Decision Processes	511
10.2 Backwards Recursion, a Generalization of Back Substitution	521
10.3 State Space, Stages, Recursive Equations	524
10.4 To Find Shortest Routes in a Staged Acyclic Network	530
10.5 Shortest Routes - 2	534
10.6 Solving the Nonnegative Integer Knapsack Problem By DP	539
10.7 Solving the 0–1 Knapsack Problem by DP	542
10.8 A Discrete Resource Allocation Problem	547
10.9 Exercises	553
10.10References	563

Chapter 10

Dynamic Programming (DP)

This is Chapter 10 of “Junior Level Web-Book *Optimization Models for decision Making*” by Katta G. Murty.

10.1 Sequential Decision Processes

So far, we have discussed methods for solving **single stage** or **static models**; i.e., we find a solution at one time for the model and we are done. But in many applications we need to make a **sequence of decisions** one after the other. These applications deal with a process or system that is observed at the beginning of a period to be in a particular **state**. That point of time may be a decision point where, one out of a possible finite set of **decisions** or **actions** is to be taken to move the system towards some goal. Two things happen, both depend on the present state of the system, and the decision taken:

- (i): an immediate cost is incurred (or reward earned)
- (ii): the action moves the system to another state in the next period.

And the same process is repeated over a finite number of periods, n say. Thus, a sequence of decisions are taken at discrete points of time. The aim is to optimize an objective function that is additive over time, to get the system to a desired final state. The objective may be to

minimize the sum of the costs incurred at the various decision points, or to maximize the sum of the rewards earned if the problem is posed that way. The important feature in such a sequential decision process is that the various decisions cannot be treated in isolation, since one must balance a desirable low cost at the time of a decision with the possibility of higher costs in later decisions.

Here we have a multistage problem involving a finite number of stages, n . The system may be in several possible states. As time passes, the state of the system changes depending on the sequence of decisions taken and the initial state at the beginning. Because of these changing states of the system, the approach for optimizing the performance of such a system is called **dynamic programming (DP)**.

A selection that specifies the action to take at each decision point is called a **policy**. The aim of DP is to determine an optimal policy that minimizes the total costs in all the stages (or maximizes the total reward if the problem is posed that way). DP solves such problems recursively in the number of stages n . At each decision point it selects an action that minimizes the sum of the current cost and the best future cost. We will now illustrate these basic concepts with some examples.

Example 10.1.1

Consider a driver in his car, starting at his office in the evening, to get home as quickly as possible. The problem of finding an optimal route for this driver through the street network of the city, is known as a **shortest route problem** or **shortest chain problem** or in some books as a **shortest path problem**. The street network is represented by a directed network in which nodes correspond to major traffic centers or street intersections, and directed arcs joining pairs of nodes correspond to street segments joining the corresponding traffic centers; the orientation of the arc being specified by the segment's orientation if it is a one way street segment, or otherwise the direction in which our driver would normally travel that segment on his way home from work. For a picture of such a network, see Figure 10.8 in Section 10.4 later on.

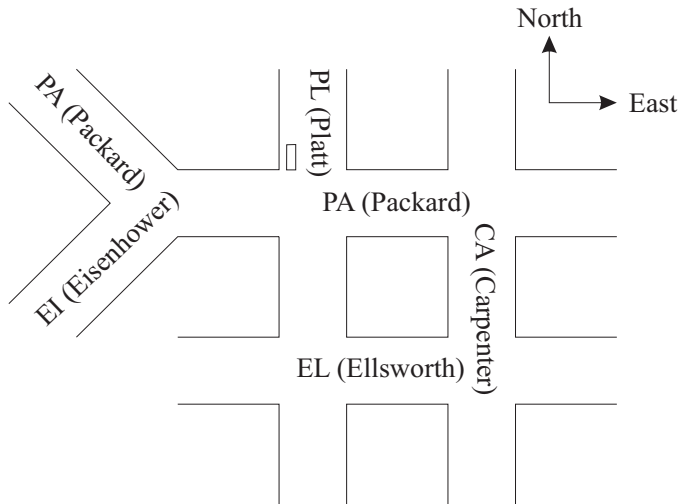


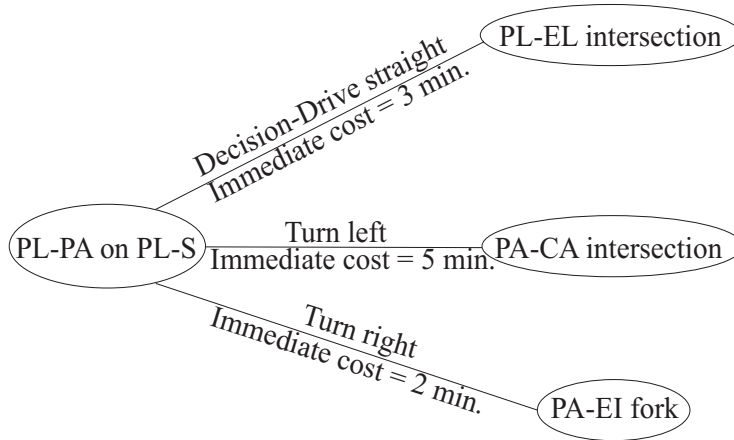
Figure 10.1: The car (system) at state “PL-PA on PL-S”.

In this problem, the system is the car with the driver sitting behind the steering wheel. The states of the system are the various street intersections or nodes. We show some of the streets in Figure 10.1, and suppose at some stage the driver has just arrived at the Platt-Packard intersection on Platt South (called PL-PA on PL-S in Figure 10.2). So, the present state of the system is PL-PA on PL-S.

There are 3 possible actions the driver can take now, they are: (a) to continue driving straight on Platt, (b) turn left onto Packard East, or (c) turn right onto Packard West. The result of each of these actions is to cause a transition of the system to the state which is the next intersection on the street along which the car continues to travel by that action; and the immediate cost incurred as a result of this action is the driving time in minutes it takes the car to reach that intersection. See Figure 10.2. The objective is to minimize the total driving time before reaching the “home” state.

The information needed to apply DP to solve a sequential decision problem such as the shortest route problem discussed in Example 10.1.1 is:

- the set of all possible states of the system (assumed to be finite);



Present state

Next state that the
decision leads to

Figure 10.2: Choice between 3 possible decisions at present state. The outcome of each is a transition to the next state shown on the right. Immediate cost of taking the decision is the driving time incurred before reaching the next state. \times

- the set of all decisions that can be taken in each state (one of these decisions has to be taken when the system reaches this state);
- and the immediate cost incurred and the next state that the system will reach under each of these decisions.

With this information we have total knowledge of the dynamics of the system. Since state transitions occur at discrete points of time, such a system is called a **discrete-time dynamic system**, and we assume that the cost function is **additive over time**. With this information, the problem of finding an optimum policy (one that specifies the optimum decision to be taken in each possible state of the system) can be solved by the DP approach. We will discuss this approach in the next section, but first we present some more examples to illustrate the basic concepts.

Important applications of DP arise in continuous time problems, but these problems are beyond the scope of this book. We restrict our discussion to discrete DP.

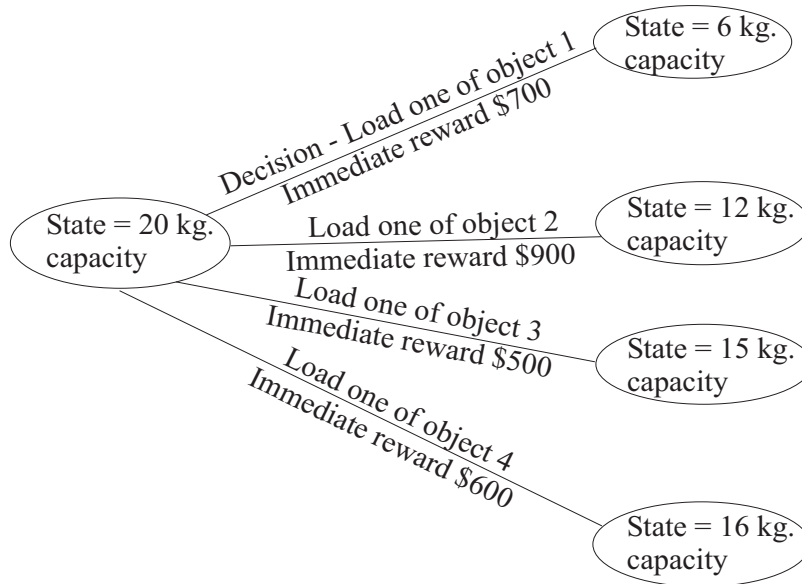
Example 10.1.2: Solving a nonnegative integer knapsack problem by DP

Consider the nonnegative integer knapsack problem discussed in Chapter 7. In this problem there are a set of n objects available to be loaded into a knapsack with a weight capacity of w_0 , a positive integer. The aim is to determine how many copies of each object to load into the knapsack, to maximize the total value of all the objects loaded subject to the knapsack's weight capacity constraint. We assume that the weights of all the objects are positive integers.

Data for the nonnegative integer knapsack problem		
Object	Weight kg.	Value \$
1	14	700
2	8	900
3	5	500
4	4	600
5	22	2700
6	25	3500

Knapsack's remaining weight capacity 20 kg.

This problem can be posed in a sequential decision format by considering the loading process as a sequential process loading one object at a time. In this format, the state of the system at any point of time in the loading process can be represented by the knapsack's remaining weight capacity. So, there are $w_0 + 1$ possible states of the system. At any stage, an object is considered available for loading into the knapsack iff its weight is \leq the knapsack's remaining weight capacity (i.e., the state of the system) at that stage. And the decisions that can be taken at that stage are to load one of the available objects into the knapsack. There is an immediate reward from that decision in the form of the value of the object loaded. This decision will reduce the



Present state

Next state that the
decision leads to

Figure 10.3: State transitions in a nonnegative integer knapsack problem.

knapsack's remaining weight capacity by the weight of the object loaded, and the next state of the system is determined from this.

As a numerical example, consider a point of time at which the knapsack's remaining weight capacity is 20 kg., and there are $n = 6$ objects according to the data given above.

Objects 5, 6 have weight $>$ the knapsack's remaining weight capacity at this time, so they are not available for loading at this time; the other objects 1 to 4 are available now. Thus there are 4 possible decisions that can be taken in the present state, corresponding to loading one of objects 1 to 4. The results of these decisions are depicted in Figure 10.3

In this problem the aim is to maximize the total value loaded, which is the sum of the rewards obtained over the entire process before it terminates. The DP approach for solving the nonnegative integer knap-

sack problem using this format is discussed later on.

Example 10.1.3: Solving a 0–1 knapsack problem by DP

Here we consider the 0–1 knapsack problem discussed in Chapters 7, 8, 9. As in Example 10.1.2, there are n objects available to be loaded into a knapsack of weight capacity w_0 , a positive integer; *but in this problem, only one copy of each object is available*. The aim of the problem here is to determine the subset of objects to be loaded so as to maximize the total value of the objects loaded subject to the knapsack's weight capacity.

In Example 10.1.2, any nonnegative integer number of copies of any of the objects could be loaded into the knapsack subject to its weight capacity and we were able to represent the state of the system by the knapsack's remaining weight capacity. Here we can include only one copy of any object in the knapsack (that is why this is the 0–1 knapsack problem), and in this problem, the knapsack's remaining weight capacity does not include enough information to fully represent the state of the system and to decide what possible decisions can be taken in a state.

For example, let the weight of object 1 be 14 kg. and at some stage, let the knapsack's remaining weight capacity be 20 kg. Because this is a 0–1 problem, at this stage object 1 is available for loading into the knapsack only if it is not already loaded into the knapsack. Thus in this problem, at any stage, an object is considered available for loading into the knapsack iff:

- (i) its weight is \leq knapsack's remaining weight capacity at this stage, and
- (ii) the copy of the object is not already loaded into the knapsack.

In this format, the state of the system can be represented by the knapsack's remaining weight capacity and the subset of objects still available for loading into the knapsack at this stage by the above definition. And the decisions that can be taken in this state are to load one of the available objects into the knapsack. And the system moves

forward.

As a numerical example, consider a point of time at which the knapsack's remaining weight capacity is 20 kg., and there are 6 objects not yet loaded into the knapsack, according to the following data.

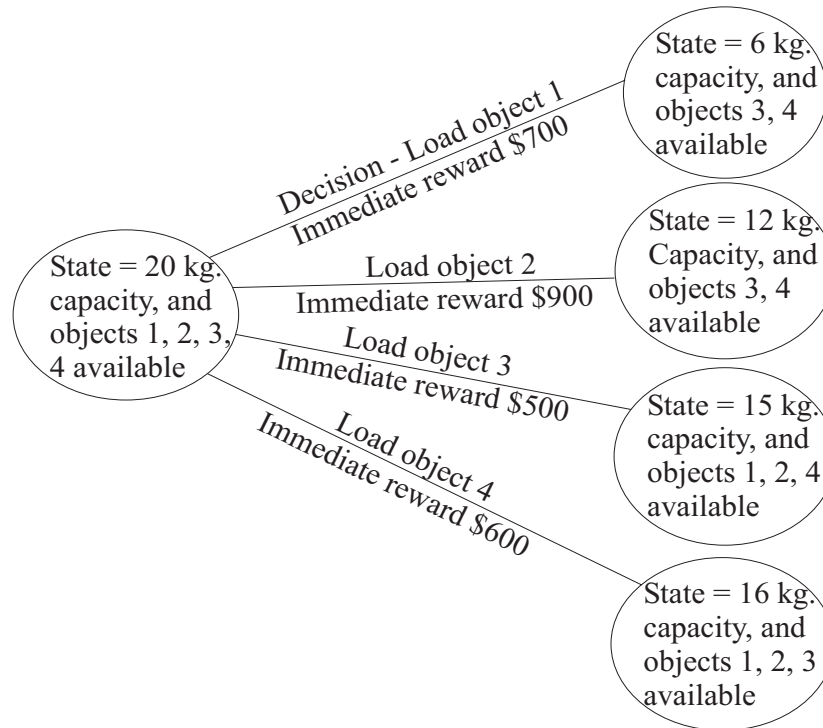
Data on objects not yet loaded		
Object	Weight kg.	Value \$
1	14	700
2	8	900
3	5	500
4	4	600
5	22	2700
6	25	3500

Knapsack's remaining weight capacity 20 kg.

Objects 5, 6 have weight greater than the knapsack's remaining weight capacity at this time, so they are not available for loading at this time; the other unloading objects 1 to 4 are available now. Thus, there are 4 possible decisions that can be taken in the present state. They correspond to loading one of objects 1 to 4. The results of these decisions are depicted in Figure 10.4

In this problem also, the objective is to maximize the total value loaded into the knapsack. Using the definition of states given here (characterized by the knapsack's remaining weight capacity and the subset of objects available for loading), the 0 – 1 knapsack problem can be solved by the DP approach. This is discussed later in Section 12.6.

The reader should pay careful attention to the difference in the definition of states in Example 10.1.2 and this example. To represent a nonnegative integer knapsack problem (any number of copies of any object could go into the knapsack subject only to its weight capacity) with n objects and knapsack's weight capacity w_0 , in a sequential decision format, we needed $w_0 + 1$ states. The 0 – 1 knapsack problem (only one copy of any object is available) with the same data may



Present state

Next state that the
decision leads to

Figure 10.4: State transitions in a 0–1 knapsack problem.

need $n(w_0 + 1)$ states to be represented in a sequential decision format, because here we need to carry the subset of objects not yet loaded into the knapsack in the definition of the state.

Ingenuity Needed to Model a Problem for Solution by DP

Thus, in posing a problem for solution by DP, one should formulate the definition of states very carefully taking the structure of the problem into account. The definition of states should always carry enough information so that the set of all possible decisions in any state can be

determined unambiguously to continue the process till the end.

That's why even when a decision problem can be posed as a sequential decision problem, formulating it for solution by DP algorithms requires a lot of ingenuity (much more so than for solving problems by techniques like LP, integer programming, discussed earlier, when those techniques are appropriate for modeling the problem) and very careful thought. Facility in applying DP comes with experience and practice, our goal in this chapter is to expose you to the basic idea of recursion that is the fundamental technique behind DP algorithms, and to illustrate it with a few very simple examples. To gain mastery of DP you have to follow this up with additional reading.

Deterministic and Stochastic DP

So far, we assumed that the result of an action taken in a state is an immediate reward which is known with certainty and transition to a known state. The branch of DP dealing with models in which there is no uncertainty, and we have perfect information about the effect of every possible action in every state of the system, is called **deterministic dynamic programming**.

In some applications the effects of actions may not be known with certainty. As an example, suppose the unemployment in the country is running around 7.5%, and the President is considering investing some federal money in public works programs to stimulate employment. Assume that the President has two possible options, to invest either \$100 billion or \$200 billion, over the next two years. The effect of either of these actions on the unemployment percentage cannot be predicted with certainty, but government economists have come up with the following estimates of the results from these investments.

Option	Estimated probability of unemployment % decreasing to		
	7.0	6.7	6.4
Invest \$100 bil.	0.70	0.20	0.10
Invest \$200 bil.	0.60	0.25	0.15

Here the state of the system is measured by the unemployment percentage. For each possible action we do not know with certainty to which state the system will move as a result of that action, but we have its probability distribution. Each of these actions may contribute some amount to the already high national debt; these contributions may not also be known with certainty, but we can estimate their probability distributions. The President's goal may be to bring the unemployment percentage to a desirable level over the next 5 years, while minimizing the total expected contribution of the actions taken in this regard to the national debt.

In this situation, the total contribution incurred to the national debt to bring the unemployment percent to a desirable level is a random variable not completely under our control, and we can only hope to minimize its expected value.

The branch of DP which deals with models based on such probabilistic data to minimize total expected cost, is called **stochastic dynamic programming**.

In this chapter we treat only deterministic DP, but the interested reader should consult the references at the end of this chapter for discussion of stochastic DP.

10.2 Backwards Recursion, a Generalization of Back Substitution

In a process that involves many steps to solve a problem, or to reach a desired goal; each step either makes the problem simpler, or brings the system closer to the desired goal. So, the remaining problem in the final step is going to be a simple one for which the solution can be obtained very easily.

Backwards recursion is the mathematical technique that starts with the simple solution for the final step, and by working backwards one step at a time, finally obtains the solution for the original problem in the initial step. It is a very important technique with many applications.

Example 10.2.1

We will illustrate backwards recursion by showing its application on a puzzle problem taken from [R. Smullyan, 1997] cited in Chapter 7. This puzzle was posed by Scheherazade to her husband, the King, on the 1003rd of their married life. Here is how she related it:

“Your Majesty, one night a thief stole into Abdul’s jewelery shop. He joyfully came across a pile of diamonds. His first thought was to take them all, but then his conscience bothered him, so he took only half the number of diamonds in the pile and started to leave. But then temptation made him take just one more, and he left the shop.

Strangely enough, a few minutes later a 2nd thief entered the shop and took half the number of remaining diamonds and one more. Then later a 3rd thief entered the shop and took half the number of remaining diamonds and one more. Then a 4th thief entered the shop and took half the number of remaining diamonds and one more.

Then Abdul entered the shop and found that all the diamonds in the pile were gone.

The problem is to determine how many diamonds were in the pile to start with.”

We will now show how this problem can be solved by backwards recursion. Remember that five persons entered the shop one after the other, Thiefs 1 to 4 and finally Abdul himself, in this order. For $i = 1$ to 5, let x_i denote the number of diamonds in the pile when the i th person in this sequence just entered the shop. Our problem is to find x_1 .

We are told that Abdul, the 5th person in the sequence, found no diamonds left in the pile. So, $x_5 = 0$. We will work backwards one step at a time, from this known information, to determine the desired quantity, value of x_1 .

Going back to the 4th person in the sequence, Thief 4, he finds x_4 diamonds in the pile when he enters the shop, and takes half of them plus one (i.e., $(x_4/2)+1$ diamonds) and hence leaves $(x_4/2)-1 = x_5 = 0$ diamonds, which yields $x_4 = 2$. This is exactly what recursion is,

knowing the value of x_5 we have found the value of x_4 in this step.

Going back to the 3rd person in the sequence, Thief 3, he finds x_3 diamonds in the pile when he enters the shop, takes $(x_3/2) + 1$ of them and leaves $(x_3/2) - 1 = x_4 = 2$ diamonds, which yields $x_3 = 6$.

Now going to the 2nd person in the sequence, Thief 2, he finds x_2 diamonds in the pile when he enters the shop, takes $(x_2/2) + 1$ of them and leaves $(x_2/2) - 1 = x_3 = 6$ diamonds, which yields $x_2 = 14$.

Now coming to the 1st person in the sequence, Thief 1, he finds x_1 diamonds in the pile when he enters the shop, takes $(x_1/2) + 1$ of them and leaves $(x_1/2) - 1 = x_2 = 14$ diamonds, which yields $x_1 = 30$. Now our original problem is solved. ∞

The reader can easily see that what we solved here is a triangular system of linear equations in variables x_i , $i = 1$ to 5 , to find the value of the variable x_i in the solution in the order $i = 5$ to 1 ; and that the method that we used is exactly the **back substitution method** of linear algebra. Back substitution is a method for solving a system of linear equations with triangular structure, backwards recursion is a generalization of it to solve more general functional equations defined in the next section.

Exercises

10.2.1: (From [R. Smullyan, 1997] cited in Chapter 7) Consider the problem solved by backwards recursion in Example 10.2.1. Suppose each of the first four persons to enter the shop (Thiefs 1 to 4) take half the number of diamonds he found plus two (instead of half plus one as in the example problem), and the 5th person (Abdul) finds none as in the example problem. Determine how many diamonds did Thief 1 find in the pile when he entered the shop.

10.2.2: (From [R. Smullyan, 1997] cited in Chapter 7) Consider the problem which is the same as that in Exercise 10.2.1, except that the 5th person (Abdul) finds one diamond when he enters. How many diamonds did Thief 1 find in the pile when he entered the shop?

10.3 State Space, Stages, Recursive Equations

States of the System, State Space

The set of all possible states of the system during the entire sequential decision process is called the **state space**. The definition of each state should contain all necessary information so that the set of decisions that can be taken when the system is in that state can be easily identified. Associated with each state s is the **decision set** $D(s)$ of decisions that can be taken at s .

States play a key role in DP. Transitions always occur from one state to another. In our deterministic DP models we assume that the definition of states is so formulated that the immediate cost or reward, and the next state that the system moves to after a decision, depend only on the current state and the decision, and not on the path of past states through which the system arrived at the current state. This property is known as the **Markovian property**.

Since states are the points where decisions are made; the sequential decision process evolves from one state to the next. The sequence of states visited by the system before the process ends, forms a path known as a **realization**; it depends on the initial state and the policy adopted (i.e., the decisions made at the various states along the path). This path can be represented as in Figure 10.5. Nodes in it represent states, and arcs correspond to decisions. There is an immediate cost incurred for each arc; the objective value of this realization is the sum of the costs incurred over all the arcs on the path.

To formulate an optimization problem for solution by DP requires the identification of all the states in the state space. This usually takes a lot of ingenuity. We will illustrate it with many examples to give the reader some experience.

Stages in Some DP Models

Every DP model has states, and it can be solved using them. But

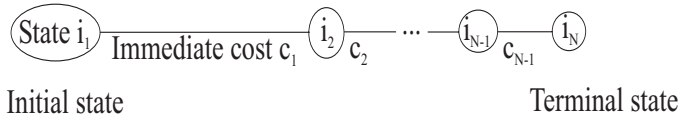


Figure 10.5: A realization of a sequential decision process. The total cost of this realization is $c_1 + c_2 + \dots + c_{N-1}$.

in some DP models there are also the so-called **stages**. These are sequential decision models in which the states form groups called **stages** that appear in some order. In these models transitions always occur from a state in some stage, to a state in the next stage. The process always begins in some state in stage 1, then it moves to stage 2, and then to stage 3, etc. Such models usually arise in situations where decisions are taken on a periodic basis, say once every time period at the beginning of the period.

Thus, if a sequential decision process has a natural organization into stages, the state space \mathbf{S} can be partitioned as $\mathbf{S}_1 \cup \mathbf{S}_2 \cup \dots \cup \mathbf{S}_N$, and the system always moves from \mathbf{S}_1 to \mathbf{S}_2 , from \mathbf{S}_2 to \mathbf{S}_3 , etc. and finally from \mathbf{S}_{N-1} to \mathbf{S}_N . All the states in \mathbf{S}_N (stage N) are terminal states, i.e., the process terminates when a state in \mathbf{S}_N is reached. In this case it is convenient to represent states so that the number of the stage to which they belong is apparent, since these stage numbers can be used to simplify the DP algorithm.

We will consider sequential decision processes that have definite ends, those at which decision making begins and ends; i.e., models with finite planning horizons. If the model is a staged model with N stages, it begins in stage 1 and terminates after $N - 1$ transitions by reaching stage N . If the model is not a staged model, it terminates whenever a desired terminal state is reached, or after some specified number of transitions take place.

Decision Sets and Policies

Consider a general model in which the state space is the set \mathbf{S} . To

apply DP we need to know, for each state $s \in \mathbf{S}$, the **decision set** $D(s)$, the set of decisions (or actions) that can be taken at s , and the immediate cost (or reward) and the next state of the system that comes up as a consequence of selecting each of these decisions. Since a solution must specify the decision to be selected from $D(s)$ for each $s \in \mathbf{S}$, it is called a **policy**. A policy completely specifies the sequence of decisions to be taken after each transition in every possible realization.

Optimum Value Function

For each state $s \in \mathbf{S}$ define

$$f(s) = \text{minimum total cost that is incurred (or maximum reward that is obtained) by pursuing an optimum policy beginning with } s \text{ as the initial state.}$$

This function $f(s)$ defined over the state space \mathbf{S} is called the **optimum value function** or OVF.

Suppose the problem specifies desired **terminal states**, i.e., the process terminates whenever the system reaches one of the states in this terminal set. No more decisions will be taken when a terminal state is reached, and the future cost (or reward) is 0. Hence, if the system is initiated in one of these terminal states, the optimum cost (or reward) is 0, i.e.,

$$f(s) = 0 \quad \text{if } s \text{ is a terminal state} \quad (10.3.1)$$

(10.3.1) is called the **boundary condition** that the OVF satisfies.

Principle of Optimality

The DP technique rests on a very simple principle called the **principle of optimality**, that is a simple consequence of the additivity property of the objective function to be optimized, and the Markovian property. We give several equivalent versions of it.

Principle of Optimality - Version 1 An optimum policy has

the property that if s is a state encountered in an optimum realization obtained by pursuing an optimum policy beginning with an initial state s_0 ; then the portion of this realization from s till the end constitutes an optimum realization if the process is initiated with the system in s .

Principle of Optimality - Version 2 Given the current state at some point of time, the optimal decisions at each of the states encountered in the future do not depend on past states or past decisions made at them.

Principle of Optimality - Version 3 An optimum policy has the property that whatever the initial state and the initial decision are, the remaining decisions must constitute an optimum policy with regards to the state resulting from the first transition.

In other words, given the current state on an optimum realization at some time, an optimum strategy for the remaining time is independent of the policy adopted in the past. So, knowledge of the current state of the system conveys all the information about its previous behavior necessary for determining the optimum sequence of decisions henceforth. This is the consequence of the Markovian property mentioned above, and the additivity of the objective function over the various transitions of the system.

Explanation of Principle of Optimality in Terms of Shortest Route Problem

To explain the principle of optimality in terms of the shortest route problem, suppose we found the shortest route, call it P , from (DE)Detroit to (SE)Seattle, and it passes through (CH)Chicago. Then the principle of optimality states that the Chicago to Seattle portion of this route, call it P_1 , is a shortest route from Chicago to Seattle. For, if P_1 is not a shortest route from Chicago to Seattle, let P_2 be a shorter route than P_1 from Chicago to Seattle. Then by following the route P from Detroit until we reach Chicago, and then following the route P_2 from

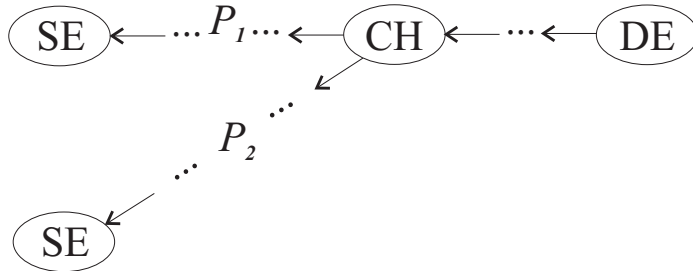


Figure 10.6: Top path P from DE to SE passes thro' CH. If CH to SE portion, P_1 of P has length 100 units, & another path P_2 from CH to SE has length < 100 (80 say), there is contradiction; since replacing P_1 on P by P_2 gives a shorter path than P from DE to SE.

Chicago to Seattle, we will get a route from Detroit to Seattle which is shorter than P , contradicting that P is a shortest route from Detroit to Seattle. See Figure 10.6.

The principle of optimality is a direct and simple consequence of the Markovian property and assumption that the objective function is the sum of the immediate costs incurred at each state along the optimal path (the additivity of the objective function over the transitions of the system).

The Functional Equation for the OVF

Consider the formulation in which the total cost is to be minimized. Let s_0 be the current state of the system at some time. Suppose there are k possible decisions available at this state, of which one must be chosen at this time. Suppose the immediate cost incurred is c_t and the system transits to state s_t if decision t is chosen at this time, for $t = 1$ to k . As defined earlier, for $t = 0, 1, \dots, k$

$$f(s_t) = \text{minimum total cost incurred by pursuing an optimum policy beginning with } s_t \text{ as the initial state.}$$

For $t = 1$ to k , if we select decision t now, but follow an optimum policy from the next state onwards, the total cost from this point of

time will be $c_t + f(s_t)$. The reason for this is the following: c_t is the immediate cost incurred as a result of the decision now, and this decision moves the system to state s_t . And $f(s_t)$ is the cost incurred by beginning with state s_t and following an optimum policy into the future. By the additivity hypothesis, the total cost from now till termination is the sum of these two costs, which is $c_t + f(s_t)$.

Hence, an optimum decision in the current state s_0 is the t between 1 to k which minimizes $c_t + f(s_t)$. Thus we have the equation

$$f(s_0) = \min\{c_t + f(s_t) : t = 1 \text{ to } k\} \quad (10.3.2)$$

and an optimum decision at the current state s_0 is the decision t which attains the minimum in (10.3.2). Clearly (10.3.2) is a direct consequence of the additivity hypothesis through the principle of optimality. (10.3.2) is intimately related to version 3 of the principle of optimality because the sum $c_t + f(s_t)$ in it is the cost of the path that selects decision t now, and thereafter uses decisions dictated by an optimal policy.

(10.3.2) is known as a **functional equation** because it gives an expression for the value of the OVF at state s_0 in terms of the values of the same function at other states s_1, \dots, s_k that can be reached from s_0 by a single decision. It is also known as the **optimality equation** in the literature.

Backwards Recursion

If the values of $f(s_1), \dots, f(s_k)$ are all known, (10.3.2) can be used to determine the value of $f(s_0)$ and the optimum decision at s_0 . This is called **recursive fixing** since it fixes the value of $f(s_0)$ from the known values of $f(s_1), \dots, f(s_k)$.

By (10.3.1), $f(s) = 0$ for every terminal state s . Starting from the known values of $f(s)$ at terminal states s (obtained from the boundary conditions), we can compute the values of the OVF at all the states, using (10.3.2), by moving backward one state at a time. This method of evaluating the values of OVF at all the states is called the **recursive technique** or **backwards recursion** (because it starts at the terminal states and moves backward one state at a time), or **recursive**

fixing (because it consists of evaluating the functional equations for the various states in a predetermined sequence). Some writers refer to the recursive technique itself as dynamic programming. DP finds an optimum policy by recursion.

If the problem is stated as one of maximizing the total reward, the OVF is defined as the total reward, and we get a functional equation similar to (10.3.2) with “maximum” replacing the “minimum.”

If the states are grouped into stages in the problem, the boundary conditions state that $f(s) = 0$ for all states s in the terminal stage, stage N , say. In such staged problems, the recursive approach begins in stage N and moves backward stage by stage, each time finding the OVF value and the optimum decision for each state in that stage.

To solve a problem by DP, the functional equations have to be developed for it individually. It takes ingenuity and insight to recognize whether a problem can be solved by DP and how to solve it actually.

The final output from DP would be a list of values of the OVF and an optimum decision for each possible state of the system, an optimum policy. One should remember that all the states may not materialize in a particular realization, but an optimum policy provides complete information on what to do if any state in the state space were to materialize.

10.4 To Find Shortest Routes in a Staged Acyclic Network

A **directed network** $G = (\mathcal{N}, \mathcal{A})$ consists of a finite set of nodes \mathcal{N} , and set of directed arcs \mathcal{A} , each arc joining a pair of nodes, with its orientation indicated by an arrow on it. The arc joining node i to node j is denoted by the ordered pair (i, j) , node i is its **tail**, and j is its **head**. In routing problems (like the one we are discussing) nodes usually represent cities, traffic centers, road crossings, etc., and arcs represent transportation channels that can be travelled only from the tail node to the head node.

A **simple circuit** in the directed network G consists of a sequence of arcs of the form $(i_1, i_2), (i_2, i_3), (i_3, i_4), \dots, (i_{k-1}, i_k), (i_k, i_1)$ along which

one can go around all the nodes i_1, i_2, \dots, i_k in it and return to the starting node, with all these nodes i_1, \dots, i_k being distinct. On the left side of Figure 10.7, there is a simple circuit with three nodes 1, 2, 3.

Top path P from DE to SE passes thro' CH. If CH to SE portion, P_1 of P has length 100 units, & another path P_2 from CH to SE has length < 100 (80 say), there is contradiction; since replacing P_1 on P by P_2 gives shorter path than P from DE to SE.

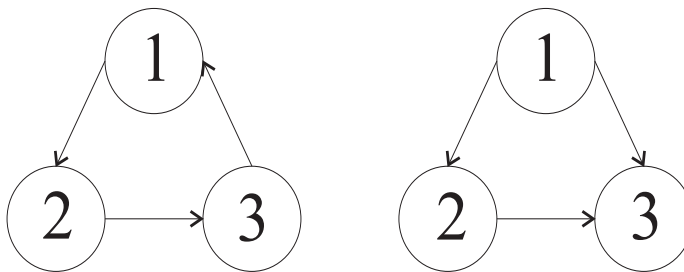


Figure 10.7: On the left we have a simple circuit. On the right we have a simple cycle that is not a simple circuit, because arc orientations are not compatible.

A simple cycle in a directed network is exactly like a simple circuit, but at least one arc in it has a reverse orientation that makes it impossible to travel around it. On the right of Figure 10.7 is a simple cycle (that is not a simple circuit), the arcs in it are $(1, 2)$, $(2, 3)$, $(1, 3)$; here arc $(1, 3)$ has the reverse orientation.

A directed network is said to be **acyclic** if it does not have any simple circuits. In the next section we will describe a simple procedure to check whether a given directed network is acyclic.

In this section, we consider special type of acyclic networks called **staged acyclic networks**. These are directed networks in which nodes are formed into groups (also called **stages**) numbered serially as Stage 1, 2, \dots ; and every arc in the network goes from a node in one stage to a node in the next stage. Because of this property, it is impossible to have a simple circuit in such a network, and so it is acyclic.

In drawing a staged acyclic network, one follows the usual convention that all the nodes in any stage are aligned vertically.

We now consider the problem of finding a shortest route from an origin node to a destination node in a directed staged acyclic network. The length (or the driving time) of each arc is given (in the example network in Figure 10.8 it is entered on the arc itself). We will illustrate the application of DP to solve this problem by backwards recursion on the network in Figure 10.8. As we move from the origin node, node 1 in stage 1, towards the destination node, node 14 in stage 6, we always move from a node in a stage, to a node in the next stage.

Nodes in the network correspond to the states of the system. So in this problem there are 14 states in all, which are grouped into 6 stages. At each node, the decisions correspond to which of the arcs incident out of it to travel next. The immediate cost of a decision is the length of the arc traveled, and this decision moves the system to the head node of that arc. For example, when at node 3 there are three decisions to choose from, they are: travel along arc (3, 6) (immediate cost 10, transit to node 6 next), or travel along arc (3, 7) (immediate cost 4, transit to node 7 next), or travel along arc (3, 8) (immediate cost 5, transit to node 8 next).

Now we define the OVF. For each $i = 1$ to 14, it is

$$f(i) = \text{length of the shortest route from node } i \text{ to the destination node 14.}$$

Since the destination node 14 represents the terminal state, the boundary condition in this problem is $f(14) = 0$. Moving backward one stage at a time, we determine the OVF and optimum decisions at the various nodes as shown below.

Stage 5

$$f(12) = \min\{14 + f(14)\} = \min\{14 + 0\} = 14. \text{ Opt. decision, travel along arc (12, 14).}$$

$$f(13) = \min\{13 + f(14)\} = \min\{13 + 0\} = 13. \text{ Opt. decision, travel along arc (13, 14).}$$

Stage 4

$$f(9) = \min\{19 + f(12), 8 + f(13)\} = \min\{19+14, 8+13\} \\ = 21. \text{ Opt. decision, travel along arc (9, 13).}$$

$$f(10) = \min\{16+f(12), 14+f(13)\} = \min\{16+14, 14+13\} \\ = 27. \text{ Opt. decision, travel along arc (10, 13).}$$

$$f(11) = \min\{12+f(13)\} = \min\{12+13\} = 25. \text{ Opt. deci-} \\ \text{sion, travel along arc (11, 13).}$$

Stage 3

$$f(5) = \min\{12+f(9)\} = \min\{12+21\} = 33. \text{ Opt. deci-} \\ \text{sion, travel along arc (5, 9).}$$

$$f(6) = \min\{6+f(9), 4+f(10)\} = \min\{6+21, 4+27\} = 27. \\ \text{Opt. decision, travel along arc (6, 9).}$$

$$f(7) = \min\{3+f(10), 9+f(11)\} = \min\{3+27, 9+25\} = \\ 30. \text{ Opt. decision, travel along arc (7, 10).}$$

$$f(8) = \min\{7+f(11)\} = \min\{7+25\} = 32. \text{ Opt. deci-} \\ \text{sion, travel along arc (8, 11).}$$

Stage 2

$$f(2) = \min\{6+f(5), 4+f(6)\} = \min\{6+33, 4+27\} = \\ 31. \text{ Opt. decision, travel along arc (2, 6).}$$

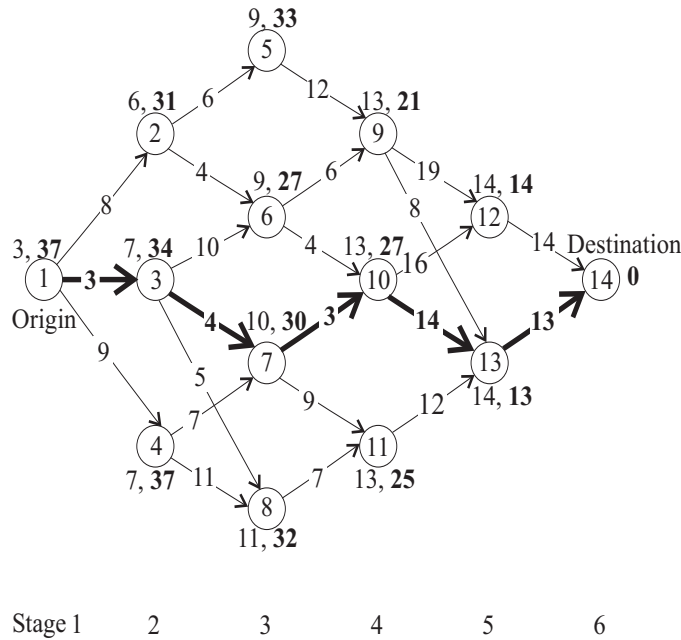
$$f(3) = \min\{10+f(6), 4+f(7), 5+f(8)\} = \min\{10+27, \\ 4+30, 5+32\} = 34. \text{ Opt. decision, travel along} \\ \text{arc (3, 7).}$$

$$f(4) = \min\{7+f(7), 11+f(8)\} = \min\{7+30, 11+32\} = \\ 37. \text{ Opt. decision, travel along arc (4, 7).}$$

Stage 1

$$f(1) = \min\{8+f(2), 3+f(3), 9+f(4)\} = \min\{8+31, \\ 3+34, 9+37\} = 37. \text{ Opt. decision, travel along} \\ \text{arc (1, 3).}$$

The optimum decisions, and the OVF values are shown on Figure 10.8. The shortest route from the origin node 1 to the destination node 14 of length 37 is marked with thick lines there. By following the optimum decisions determined at the nodes, we can also obtain the shortest route from any node in the network, to node 14.



Stage 1 2 3 4 5 6

Figure 10.8: The staged acyclic network. Node numbers are entered inside them. Arc lengths are marked on them. By the side of each node we marked the next node to go to (to reach the destination by a shortest route from that node), and in bold the length of the shortest route from that node to the destination. The shortest route from the origin to the destination is marked in thick lines.

10.5 Shortest Routes in an Acyclic Network That is Not Staged

Let $G = (\mathcal{N}, \mathcal{A})$ be a directed network with \mathcal{N} as the set of nodes, and \mathcal{A} as the set of arcs. Let $|\mathcal{N}| = n$.

Here we first discuss how to check whether G is acyclic. Assuming that it is, we then show how to find a shortest route from an origin node to a destination node in it, using DP treating the nodes as states. Even though there are no stages, backwards recursion solves the functional equations beginning with the destination node, and moving backward one node at a time.

How to Check Whether G is Acyclic, and Develop an Acyclic Numbering for its Nodes

A special property of acyclic networks is that its node can be numbered in such a way that with this numbering, on every arc (i, j) of the network, $i < j$ (i.e., on every arc the number of the tail node is $<$ the number of the head node). See [K. G. Murty, 1992, of Chapter 5] for a proof. A numbering of nodes of the network satisfying this property is called **acyclic numbering of the nodes**. Such a numbering can be found by using the following procedure.

Initially this procedure begins with the original network G . During the procedure, which may take several steps, in each step some nodes are numbered, they and all the arcs containing them are considered deleted from the network for the remaining part of the procedure, and the process continues with the remaining network; until all the nodes in the network are numbered.

- 1 If there are nodes that have not been numbered in the procedure, look for nodes which have no arcs incident into them in the remaining part of the network. If there are no nodes satisfying this property, the network is not acyclic, terminate. Otherwise, number all these nodes serially in some order beginning with 1 if this is the first step, or beginning with the next unused integer if some nodes are numbered already. Go to 2.
- 2 If all the nodes are now numbered, we have the desired node numbering, terminate. Otherwise, consider all the newly numbered nodes and arcs incident at them as deleted, and go back to 1 with the remaining part of the network.

As an example, consider the network on the left in Figure 10.9. In this network, the leftmost pair of nodes have no arcs incident into them. So, they are numbered 1, 2 first. Continuing this way, nodes get numbered by the above procedure from left to right, leading to the acyclic numbering of the nodes on the right in Figure 10.9.

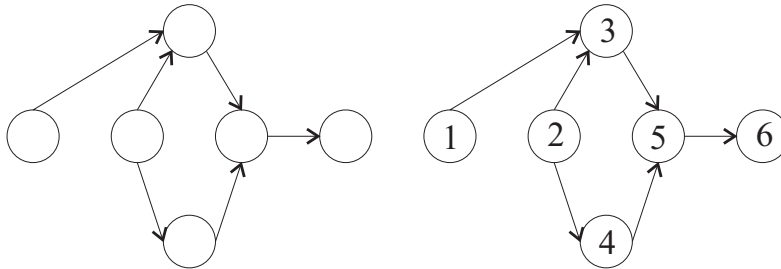


Figure 10.9: Acyclic numbering of nodes in an acyclic network.

As another example, consider the network in Figure 10.10. The one node in this network with no arc incident into it is numbered as node 1. In the remaining network after node 1 and the thick arcs incident at it are deleted, every node has an arc incident into it. So, we terminate with the conclusion that the network in Figure 10.10 is not acyclic.

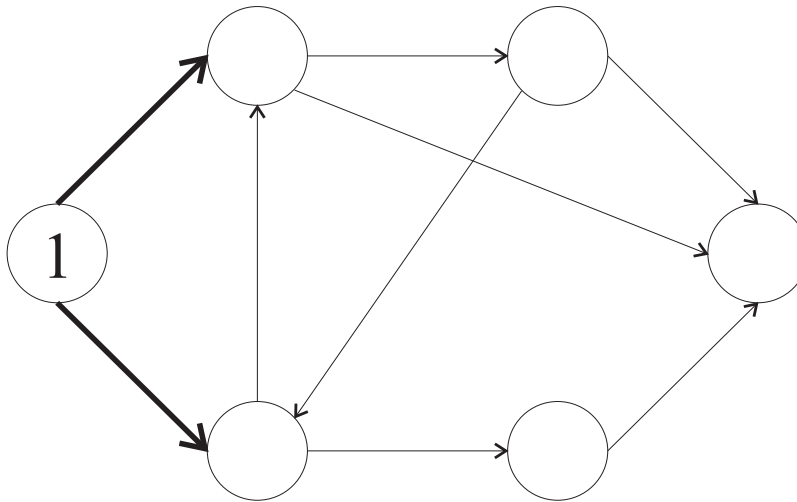


Figure 10.10: A network that is not acyclic.

In the original network, G itself, nodes may have some numbers or labels. However, the DP algorithm for finding the shortest route operates with the acyclic numbering of the nodes just obtained.

The DP Algorithm for Finding Shortest Routes

This algorithm works with the acyclic numbering of the nodes, and hence only applies to acyclic networks. If the network is not acyclic, a different algorithm (not discussed in this book) has to be used for finding shortest routes in it. So, the node numbers in the discussion here refer to those in the acyclic numbering.

We assume that nodes in our network G are numbered serially using an acyclic numbering. Let nodes 1, n be the origin, destination, respectively. For each arc (i, j) in the network G let c_{ij} be its length. As before, define the OVF

$$f(i) = \text{length of the shortest route from node } i \text{ to the destination node } n.$$

The boundary condition is $f(n) = 0$. Beginning with this, backwards recursion computes the values of the OVF in the order $f(n-1), f(n-2), \dots, f(1)$, using the functional equation

$$f(i) = \min\{c_{ij} + f(j) : i+1 \leq j \leq n \text{ s. th. } (i, j) \text{ arc in } G\} \quad (10.5.1)$$

in the order $i = n-1$ to 1. The j that attains the minimum on the right in (10.5.1) defines the next node to go to from node i .

As an example, consider the network in Figure 10.11, with arc lengths entered on the arcs, and nodes with an acyclic numbering. Clearly, this is an acyclic network, but not a staged one as defined in Section 10.4.

Here is how backwards recursion proceeds on this network. The boundary condition is $f(9) = 0$ since 9 is the destination node.

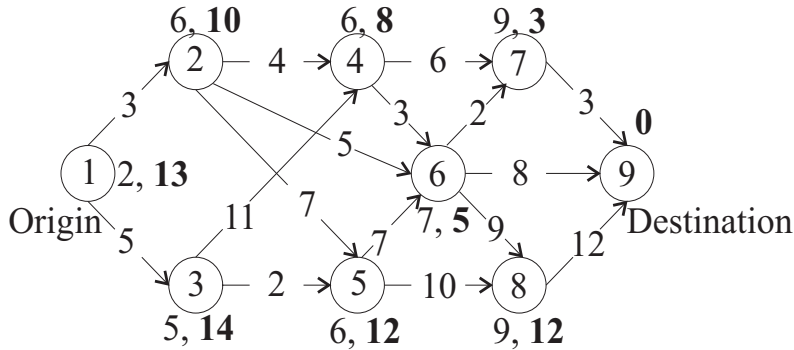


Figure 10.11: The length of each arc is entered on it. By the side of each node we marked the next node to go to on the shortest route from that node to the destination, and the length of that shortest route in bold face.

$$\begin{aligned}
 f(8) &= \min\{12 + f(9)\} = \min\{12 + 0\} = 12. \text{ Opt. decision, travel along arc } (8, 9). \\
 f(7) &= \min\{3 + f(9)\} = \min\{3 + 0\} = 3. \text{ Opt. decision, travel along arc } (7, 9). \\
 f(6) &= \min\{2 + f(7), 8 + f(9), 9 + f(8)\} = \min\{2+3, 8+0, 9+12\} = 5. \text{ Opt. decision, travel along arc } (6, 7). \\
 f(5) &= \min\{7 + f(6), 10 + f(8)\} = \min\{7+5, 10+12\} = 12. \text{ Opt. decision, travel along arc } (5, 6). \\
 f(4) &= \min\{6 + f(7), 3 + f(6)\} = \min\{6+3, 3+5\} = 8. \text{ Opt. decision, travel along arc } (4, 6). \\
 f(3) &= \min\{11 + f(4), 2 + f(5)\} = \min\{11+8, 2+12\} = 14. \text{ Opt. decision, travel along arc } (3, 5). \\
 f(2) &= \min\{4 + f(4), 5 + f(6), 7 + f(5)\} = \min\{4+8, 5+5, 7+12\} = 10. \text{ Opt. decision, travel along arc } (2, 6). \\
 f(1) &= \min\{3 + f(2), 5 + f(3)\} = \min\{3+10, 5+14\} = 13. \text{ Opt. decision, travel along arc } (1, 2).
 \end{aligned}$$

The optimum decisions and the OVF values at the various nodes are

shown on Figure 10.11. The OVF of node 1 is 13; it is the length of the shortest route from node 1 to node 9 in this network. By following the optimum decisions determined at the various nodes, we can also obtain the shortest route from any node in the network to the destination node 9.

DP can also be applied to find shortest routes in directed networks that are not acyclic. We refer the reader to [K. G. Murty, 1992] for a discussion of DP based shortest route algorithms in non-acyclic directed networks.

10.6 Solving the Nonnegative Integer Knapsack Problem By DP

Consider the nonnegative integer knapsack problem in which there are n objects available to load into the knapsack, with w_i, v_i being the weight and value of the i th object, for $i = 1$ to n . Let w_0 be the knapsack's weight capacity. All w_0, w_1, \dots, w_n are assumed to be positive integers. The problem is to determine the number of copies of each object to load into the knapsack to maximize the total value of all objects loaded, subject to the knapsack's weight capacity.

As discussed in Example 10.1.2, to solve this problem by DP we consider the loading process as a sequential process loading one object at a time, and represent the state of the system at any point of time in this process by the knapsack's remaining weight capacity. We define the OVF in state w to be

$$f(w) = \text{maximum possible value that can be loaded into the knapsack if its weight capacity is } w$$

When the knapsack's weight capacity is w , only objects i satisfying $w_i \leq w$, are available for loading into it. So, the functional equation satisfied by the OVF in this problem is

$$f(w) = \max\{v_i + f(w - w_i) : i \text{ s. th. } w_i \leq w\} \quad (10.6.1)$$

The operation in (10.6.1) is "max" instead of the usual "min" be-

cause our aim here is to maximize the total reward.

Clearly $f(0) = 0$ is the boundary condition satisfied by the OVF in this problem. Beginning with this, we evaluate $f(w)$ for $w = 1, 2, \dots, w_0$ in this order recursively using (10.6.1). The i attaining the maximum in (10.6.1) is the number of the object to be loaded into the knapsack when in state w , in an optimal policy.

As an example consider the problem with $n = 6$, and the following data.

Data for a nonnegative integer knapsack problem		
Object i	Weight w_i	Value v_i
1	3	12
2	4	12
3	3	9
4	3	15
5	7	42
6	9	18

Knapsack's weight capacity, $w_0 = 12$

Since all objects have weights ≥ 3 , we have $f(0) = f(1) = f(2) = 0$ in this problem, these are the boundary conditions here.

So when the state of the system is 0, 1, or 2 (i.e., the remaining weight capacity of the knapsack is 0, 1, or 2) we just terminate, since no more objects can be loaded into the knapsack. When the state of the system is 3, objects 1, 3, 4 become available to be loaded into the knapsack, leading to the following equation for $f(3)$. Continuing in this way we evaluate $f(w)$ for higher values of w until $w_0 = 12$. As you can see, to evaluate an $f(W)$ say, the functional equation for $f(W)$ uses the values of $f(w)$ for $w < W$. That's why the procedure computes values of $f(w)$ in order of increasing w beginning with the known values of $f(0), f(1), f(2)$ given by the boundary conditions. This is the recursive feature of the DP algorithm.

$$\begin{aligned}
f(0) &= f(1) = f(2) = 0. \text{ Opt. decision - terminate.} \\
f(3) &= \text{Max}\{12+f(0), 9+f(0), 15+f(0)\} = \max\{12+0, 9+0, 15+0\} = 15. \text{ Opt. decision - load one of object 4 and continue as in state 0.} \\
f(4) &= \text{Max}\{12+f(1), 12+f(0), 9+f(1), 15+f(1)\} = \max\{12+0, 12+0, 9+0, 15+0\} = 15. \text{ Opt. decision - load one of object 4 and continue as in state 1.} \\
f(5) &= \text{Max}\{12+f(2), 12+f(1), 9+f(2), 15+f(2)\} = \max\{12+0, 12+0, 9+0, 15+0\} = 15. \text{ Opt. decision - load one of object 4 and continue as in state 2.} \\
f(6) &= \text{Max}\{12+f(3), 12+f(2), 9+f(3), 15+f(3)\} = \max\{12+15, 12+0, 9+15, 15+15\} = 30. \text{ Opt. decision - load one of object 4 and continue as in state 3.} \\
f(7) &= \text{Max}\{12+f(4), 12+f(3), 9+f(4), 15+f(4), 42+f(0)\} = \max\{12+15, 12+15, 9+15, 15+15, 42+0\} = 42. \text{ Opt. decision - load one of object 5 and continue as in state 0.} \\
f(8) &= \text{Max}\{12+f(5), 12+f(4), 9+f(5), 15+f(5), 42+f(1)\} = \max\{12+15, 12+15, 9+15, 15+15, 42+0\} = 42. \text{ Opt. decision - load one of object 5 and continue as in state 1.} \\
f(9) &= \text{Max}\{12+f(6), 12+f(5), 9+f(6), 15+f(6), 42+f(2), 18+f(0)\} = \max\{12+30, 12+15, 9+30, 15+30, 42+0, 18+0\} = 45. \text{ Opt. decision - load one of object 4 and continue as in state 6.} \\
f(10) &= \text{Max}\{12+f(7), 12+f(6), 9+f(7), 15+f(7), 42+f(3), 18+f(1)\} = \max\{12+42, 12+30, 9+42, 15+42, 42+15, 18+0\} = 57. \text{ Opt. decision - load one of object 4 and continue as in state 7.} \\
f(11) &= \text{Max}\{12+f(8), 12+f(7), 9+f(8), 15+f(8), 42+f(4), 18+f(2)\} = \max\{12+42, 12+42, 9+42, 15+42, 42+15, 18+0\} = 57. \text{ Opt. decision - load one of object 4 and continue as in state 8.}
\end{aligned}$$

$$\begin{aligned}
 f(12) = \text{Max}\{12+f(9), 12+f(8), 9+f(9), 15+f(9), \\
 42+f(5), 18+f(3)\} = \max\{12+45, 12+42, 9+45, \\
 15+45, 42+15, 18+15\} = 60. \text{ Opt. decision -} \\
 \text{load one of object 4 and continue as in state 9.}
 \end{aligned}$$

By following the optimum decisions beginning with state 12, we see that an optimum strategy to maximize the value loaded when the weight capacity of the knapsack is 12, is to load four copies of object 4 into it, giving a total value of 60 for the objects loaded.

Since the value of $f(w)$ has to be computed for all $0 \leq w \leq w_0$ in this algorithm, it is not efficient for solving this problem, in comparison to B&B methods discussed in Chapter 8. We discussed this method mainly to illustrate an application of DP.

10.7 Solving the 0–1 Knapsack Problem by DP

Consider the 0–1 knapsack problem involving n objects. Let w_0 be the capacity of the knapsack by weight, and let w_i, v_i be the weight and value of the i th object, $i = 1$ to n . Here, only one copy of each object is available. The problem is to determine the subset of objects to be loaded into the knapsack to maximize the value loaded subject to its weight capacity. We assume that w_0, w_1, \dots, w_n are all positive integers.

To solve this problem by DP we consider the loading process as a sequential process loading one object at a time. However, as pointed out in Example 10.1.3, since there is only one copy of each object available, the definition of the state of the system at any point of time in this process must contain information on the remaining weight capacity of the knapsack at that time, and the set of objects not yet loaded.

For this it is convenient to represent the process as a staged process with n stages. For each $k = 1$ to n , states in stage k will be denoted by the ordered pair (\mathbf{k}, w) where $0 \leq w \leq w_0$ represents the knapsack's remaining weight capacity at that stage. In any state (\mathbf{k}, w) in stage k , there are at most two possible decisions that can be taken, and they are:

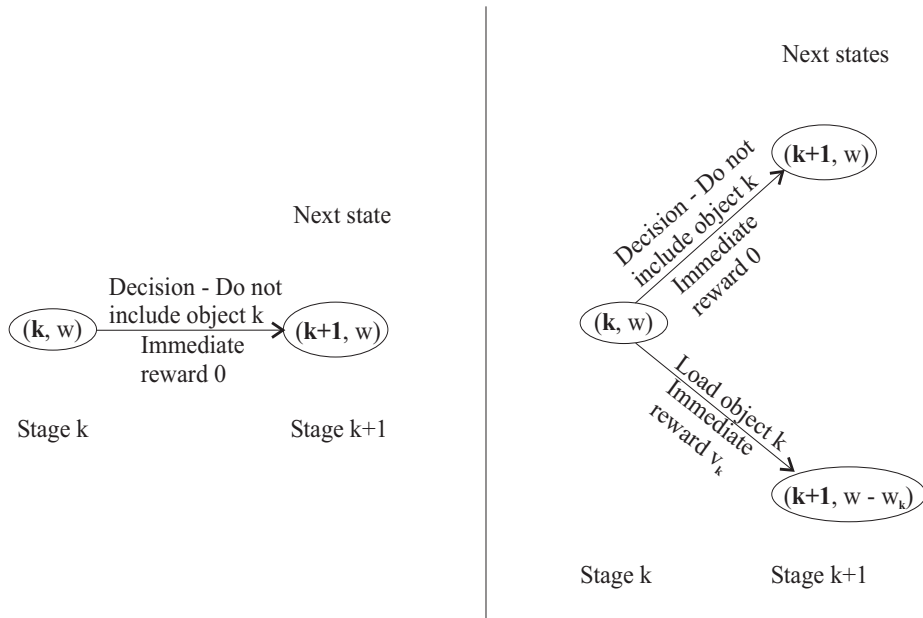


Figure 10.12: On the left is displayed the unique choice in state (\mathbf{k}, w) if $w < w_k$. On the right are displayed the two available choices at state (\mathbf{k}, w) if $w \geq w_k$.

(i): to decide not to include object k in the knapsack (in fact this is the only decision available if $w_k > w$) with an immediate reward of 0 and transition to state $(\mathbf{k} + 1, w)$ in stage $k + 1$; or

(ii): to load object k into the knapsack (this decision is only available if $w \geq w_k$) with an immediate reward of v_k and transition to state $(\mathbf{k} + 1, w - w_k)$ in stage $k + 1$.

This creates an artificial stage structure with n stages, with the decision in stage k relating only to the inclusion or exclusion of object k , for each $k = 1$ to n . Thus each object's fate is considered in a unique stage, and there can be no confusion in any state what the available decisions in that state are. The available decisions at state (\mathbf{k}, w) and the resulting state transitions are displayed in Figure 10.12.

We now define the OVF in state (\mathbf{k}, w) to be

$$f(\mathbf{k}, w) = \text{maximum possible value that can be loaded into the knapsack if its weight capacity is } w, \text{ and choice of objects restricted to only those in the set } \{k, k+1, \dots, n\}. \quad (10.7.1)$$

Since only one copy of each object is available, for $k = n$ we clearly have

$$f(\mathbf{n}, w) = \begin{cases} 0 & \text{if } w < w_n \\ v_n & \text{if } w \geq w_n \end{cases} \quad (10.7.2)$$

(10.7.2) is the boundary condition that the OVF $f(\mathbf{k}, w)$ satisfies in this problem. From the decisions available at state (\mathbf{k}, w) displayed in Figure 10.12, we get the functional equations satisfied by the OVF to be

$$f(\mathbf{k}, w) = \begin{cases} f(\mathbf{k} + \mathbf{1}, w) & \text{if } w < w_k \\ \max\{f(\mathbf{k} + \mathbf{1}, w), v_k + f(\mathbf{k} + \mathbf{1}, w - w_k)\} & \text{if } w \geq w_k \end{cases} \quad (10.7.3)$$

Using the boundary conditions in (10.7.2) and the functional equations in (10.7.3), the OVF at all states can be evaluated by moving forward one stage at a time beginning with stage $n - 1$. We compute the OVF at all states in a stage when we deal with that stage and then move forward to the adjacent stage. At state (\mathbf{k}, w) , the decision is to exclude object k from the knapsack if it happens that $f(\mathbf{k}, w) = f(\mathbf{k} + \mathbf{1}, w)$ in (10.7.3); or to load object k into the knapsack if $f(\mathbf{k}, w) = v_k + f(\mathbf{k} + \mathbf{1}, w - w_k)$.

As an example consider the 0–1 knapsack problem with $n = 5$ and the following data.

Data for a 0–1 knapsack problem		
Object i	Weight w_i	Value v_i
1	3	12
2	4	12
3	3	15
4	7	42
5	9	18

Knapsack's weight capacity, $w_0 = 12$

Stage 5: Boundary conditions

$$f(5,0) \text{ to } f(5,8) = 0. \text{ Opt. decision - terminate.}$$

$$f(5,9) \text{ to } f(5,12) = 18. \text{ Opt. decision - load object 5 and terminate.}$$

Stage 4

$$f(4,w) = f(5,w) \text{ for } w = 0 \text{ to } 6. \text{ Opt. decision - exclude object 4 and continue as in state } (5,w).$$

$$f(4,7) = \text{Max}\{0+f(5,7), 42+f(5,0)\} = \max\{0+0, 42+0\} = 42. \text{ Opt. decision - load object 4 and continue as in } (5,0).$$

$$f(4,8) = \text{Max}\{0+f(5,8), 42+f(5,1)\} = \max\{0+0, 42+0\} = 42. \text{ Opt. decision - load object 4 and continue as in state } (5,1).$$

$$f(4,9) = \text{Max}\{0+f(5,9), 42+f(5,2)\} = \max\{0+18, 42+0\} = 42. \text{ Opt. decision - load object 4 and continue as in state } (5,2).$$

$$f(4,10) = \text{Max}\{0+f(5,10), 42+f(5,3)\} = \max\{0+18, 42+0\} = 42. \text{ Opt. decision - load object 4 and continue as in state } (5,3).$$

$$f(4,11) = \text{Max}\{0+f(5,11), 42+f(5,4)\} = \max\{0+18, 42+0\} = 42. \text{ Opt. decision - load object 4 and continue as in state } (5,4).$$

$$f(4,12) = \text{Max}\{0+f(5,12), 42+f(5,5)\} = \max\{0+18, 42+0\} = 42. \text{ Opt. decision - load object 4 and continue as in state } (5,5).$$

Continuing the same way, we get the following OVF values and optimum decisions at states in stages 3, 2.

w	Stage 3		Stage 2	
	OVF $f(3, w)$	Opt. decision	OVF $f(2, w)$	Opt. decision
0, 1, 2	$f(4, w)$	Exclude obj. 3 Cont. as in $(4, w)$	$f(3, w)$	Exclude obj. 2 Cont. as in $(3, w)$
3	15	Load obj. 3 Cont. as in $(4,0)$	15	Exclude obj. 2 Cont. as in $(3,3)$

w	Stage 3		Stage 2	
	OVF $f(\mathbf{3}, w)$	Opt. decision	OVF $f(\mathbf{2}, w)$	Opt. decision
4	15	Load obj. 3 Cont. as in (4,1)	15	Exclude obj. 2 Cont. as in (3,4)
5	15	Load obj. 3 Cont. as in (4,2)	15	Exclude obj. 2 Cont. as in (3,5)
6	15	Load obj. 3 Cont. as in (4,3)	15	Exclude obj. 2 Cont. as in (3,6)
7	42	Exclude obj. 3 Cont. as in (4,7)	42	Exclude obj. 2 Cont. as in (3,7)
8	42	Exclude obj. 3 Cont. as in (4,8)	42	Exclude obj. 2 Cont. as in (3,8)
9	42	Exclude obj. 3 Cont. as in (4,9)	42	Exclude obj. 2 Cont. as in (3,9)
10	57	Load obj. 3 Cont. as in (4,7)	57	Exclude obj. 2 Cont. as in (3,10)
11	57	Load obj. 3 Cont. as in (4,8)	57	Exclude obj. 2 Cont. as in (3,11)
12	57	Load obj. 3 Cont. as in (4,9)	57	Exclude obj. 2 Cont. as in (3,12)

And finally, we have $f(\mathbf{1}, 12) = \max\{0+f(\mathbf{2}, 12), 12+f(\mathbf{2}, 9)\} = \max\{0+57, 12+42\} = 57$, with the optimum decision in state (1,12) to be to exclude object 1, and continue as in state (2,12). Following the decisions in the various stages, we see that an optimum strategy in the original problem to maximize the value loaded in the knapsack is to load objects 3 and 4 into it. This leads to the maximum value loaded of 57.

Since the value of $f(\mathbf{k}, w)$ has to be evaluated for all $1 \leq k \leq n$ and $0 \leq w \leq w_0$ in this algorithm, it is not efficient to solve the 0–1 knapsack problem, in comparison to B&B methods discussed in Chapter 8 when w_0 is large. Our main interest in discussing this algorithm here is to illustrate another application of DP.

10.8 A Discrete Resource Allocation Problem

There are K units of a single resource available, which can be distributed among n different activities. K is a positive integer. The problem is to allocate the resource units most profitably among the activities. Assume that resource units can only be allocated to activities in non-negative integer quantities. Define for $i = 1$ to n

$$x_i = \text{number of units of resource allotted to activity } i \quad (10.8.1)$$

Let $r_i(x_i)$ denote the profit or reward realized from an allocation of x_i units of resource to activity i . A table giving the values of $r_i(x_i)$ for $x_i = 0$ to K , $i = 1$ to n , is the data for this problem. We assume that $r_i(x_i) \geq 0$ for all $x_i \geq 0$, $i = 1$ to n . The problem is to choose a nonnegative integer vector $x = (x_1, \dots, x_n)^T$ so as to maximize the total reward subject to the constraint $x_1 + \dots + x_n \leq K =$ the units of resource available.

This problem is not really dynamic, but can be posed as a staged sequential decision problem involving n stages, based on the technique used for the 0–1 knapsack problem. It views this problem as a sequential decision process in which at the i th stage only the value of the variable x_i (the number of units of resource to be allotted to activity i) is determined, $i = 1$ to n . The states of the system in stage i are (\mathbf{i}, k) , $0 \leq k \leq K$, where k denotes the number of unallotted units of resource available at this stage. The possible decisions available in state (\mathbf{i}, k) are to select an integral value for the variable x_i between 0 and k , leading to an immediate reward of $r_i(x_i)$ and a transition to state $(\mathbf{i} + \mathbf{1}, k - x_i)$ in stage $i + 1$. These state transitions are illustrated in Figure 10.13.

We now define the OVF in this process to be

$$f(\mathbf{i}, k) = \text{maximum total reward that can be obtained from activities } i \text{ to } n, \text{ with } k \text{ units of resource that can be allotted among them} \quad (10.8.2)$$

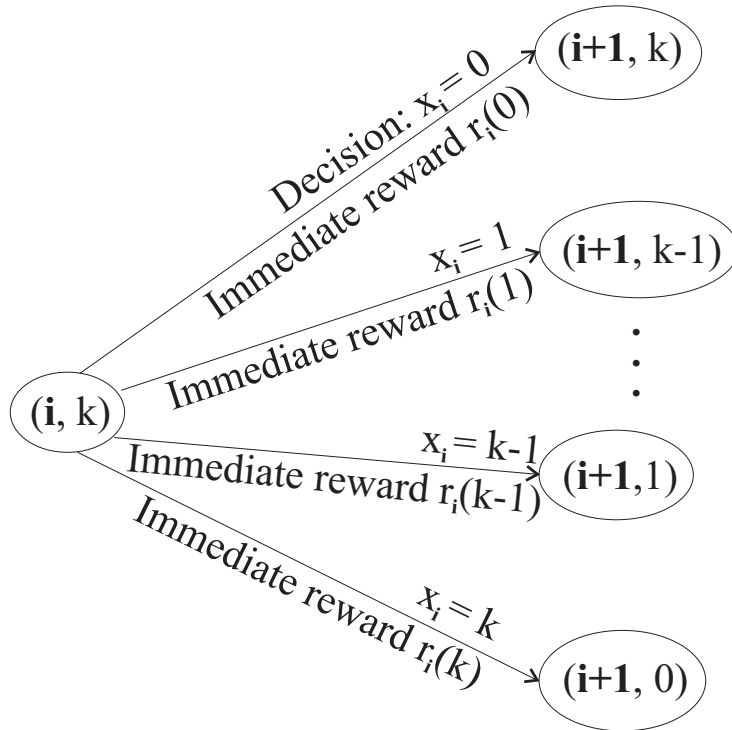


Figure 10.13: State transitions from a state (i, k) in stage i to stage $i + 1$.

This OVF clearly satisfies the following boundary condition

$$f(\mathbf{n}, k) = \max\{r_n(t) : 0 \leq t \leq k\} \quad (10.8.3)$$

and if p attains the maximum in (10.8.3), the optimum decision in state (\mathbf{n}, k) is to allot p units of resource to activity n and leave the other $k - p$ units of resource unallotted.

Normally the reward function $r_i(k)$ will be monotonic increasing in k for all i (in most real world applications this will be the case since the return is usually an increasing function of the resources committed). In this case, (10.8.3) becomes $f(\mathbf{n}, k) = r_n(k)$ for all $0 \leq k \leq K$ and the optimum decision in state (\mathbf{n}, k) is to allot all k units of resource to activity n .

From the state transitions illustrated in Figure 10.13, and the principle of optimality, it is clear that

$$f(\mathbf{i}, k) = \max\{r_i(x_i) + f(\mathbf{i} + \mathbf{1}, k - x_i) : 0 \leq x_i \leq k\} \quad (10.8.4)$$

for $1 \leq i \leq n$ and $0 \leq k \leq K$. (10.8.4) is the functional equation satisfied by the OVF in this problem. The optimum decision in state (\mathbf{i}, k) is to make the variable x_i equal to the argument attaining the maximum in (10.8.4).

Beginning with the known values of $f(\mathbf{n}, k)$, $0 \leq k \leq K$ given by the boundary conditions (10.8.3), the values of the OVF can be evaluated and the optimum decisions at all states in other stages determined, in the order: stage $n - 1$, $n - 2$, \dots , 1.

As an example, consider the problem faced by a politician running for reelection for his position in city administration. He has $K = 5$ volunteers who have agreed to help his campaign by distributing posters door to door and talking to residents in his district. We give below estimates of additional votes that would result from assigning these volunteers to 4 different precincts.

No. of volunteers assigned k	$r_i(k) =$ expected additional votes (in 100s) gained by assigning k volunteers to precinct			
	$i = 1$	2	3	4
0	0	0	0	0
1	35	79	130	86
2	42	110	160	120
3	56	130	170	130
4	50	140	180	130
5	50	125	175	125

If too many volunteers knock on the doors people may get irritated and react negatively; that's why in this problem $r_i(k)$ increases as k increases up to a value, and then begins to decrease.

Here the volunteers are the resource and we have 5 of them. The problem is to determine the optimum number of volunteers to allot to

the various precincts in order to maximize the total expected additional votes gained by their effort.

So, we define 4 stages, with stage i dealing with the decision variable x_i = number of volunteers allotted to precinct i , $i = 1$ to 4. For $i = 1$ to 4, the symbol (\mathbf{i}, k) defines the state in stage i of having k volunteers to assign in precincts i to 4. The OVF here is

$$f(\mathbf{i}, k) = \text{maximum expected additional votes gained by allotting } k \text{ volunteers in precincts } i \text{ to 4 optimally.}$$

The boundary conditions for stage 4 are given below.

$$f(\mathbf{4}, 0) = \text{Max}\{0\} = 0. \text{ Opt. decision - allot 0 volunteers to precinct 4 and terminate.}$$

$$f(\mathbf{4}, 1) = \text{Max}\{0, 86\} = 86. \text{ Opt. decision - allot 1 volunteer to precinct 4 and terminate.}$$

$$f(\mathbf{4}, 2) = \text{Max}\{0, 86, 120\} = 120. \text{ Opt. decision - allot 2 volunteers to precinct 4 and terminate.}$$

$$f(\mathbf{4}, 3) = \text{Max}\{0, 86, 120, 130\} = 130. \text{ Opt. decision - allot 3 volunteers to precinct 4 and terminate.}$$

$$f(\mathbf{4}, 4) = \text{Max}\{0, 86, 120, 130, 130\} = 130. \text{ Opt. decision - allot 3 volunteers to precinct 4 and terminate.}$$

$$f(\mathbf{4}, 5) = \text{Max}\{0, 86, 120, 130, 130, 125\} = 130. \text{ Opt. decision - allot 3 volunteers to precinct 4 and terminate.}$$

We now compute the OVF and the optimum decision in each state in other stages, in the order stage 3, 2, 1, recursively.

Stage 3

$$\begin{aligned}
f(\mathbf{3}, 0) &= \text{Max}\{0 + f(\mathbf{4}, 0)\} = \max\{0+0\} = 0. \text{ Opt. decision - allot 0 volunteers to precinct 3 and continue as in state } (\mathbf{4}, 0). \\
f(\mathbf{3}, 1) &= \text{Max}\{0 + f(\mathbf{4}, 1), 130+f(\mathbf{4}, 0)\} = \max\{0+86, 130+0\} = 130. \text{ Opt. decision - allot 1 volunteer to precinct 3 and continue as in state } (\mathbf{4}, 0). \\
f(\mathbf{3}, 2) &= \text{Max}\{0 + f(\mathbf{4}, 2), 130+f(\mathbf{4}, 1), 160+f(\mathbf{4}, 0)\} = \max\{0+120, 130+86, 160+0\} = 216. \text{ Opt. decision - allot 1 volunteer to precinct 3 and continue as in state } (\mathbf{4}, 1). \\
f(\mathbf{3}, 3) &= \text{Max}\{0 + f(\mathbf{4}, 3), 130+f(\mathbf{4}, 2), 160+f(\mathbf{4}, 1), 170+f(\mathbf{4}, 0)\} = \max\{0+130, 130+120, 160+86, 170+0\} = 250. \text{ Opt. decision - allot 1 volunteer to precinct 3 and continue as in state } (\mathbf{4}, 2). \\
f(\mathbf{3}, 4) &= \text{Max}\{0 + f(\mathbf{4}, 4), 130+f(\mathbf{4}, 3), 160+f(\mathbf{4}, 2), 170+f(\mathbf{4}, 1), 180+f(\mathbf{4}, 0)\} = \max\{0+130, 130+130, 160+120, 170+86, 180+0\} = 280. \text{ Opt. decision - allot 2 volunteers to precinct 3 and continue as in state } (\mathbf{4}, 2). \\
f(\mathbf{3}, 5) &= \text{Max}\{0 + f(\mathbf{4}, 5), 130+f(\mathbf{4}, 4), 160+f(\mathbf{4}, 3), 170+f(\mathbf{4}, 2), 180+f(\mathbf{4}, 1), 175+f(\mathbf{4}, 0)\} = \max\{0+130, 130+130, 160+130, 170+120, 180+86, 175+0\} = 290. \text{ Opt. decision - allot 3 volunteers to precinct 3 and continue as in state } (\mathbf{4}, 2).
\end{aligned}$$

Stage 2

$$\begin{aligned}
f(\mathbf{2}, 0) &= \text{Max}\{0 + f(\mathbf{3}, 0)\} = \max\{0+0\} = 0. \text{ Opt. decision - allot 0 volunteers to precinct 2 and continue as in state } (\mathbf{3}, 0). \\
f(\mathbf{2}, 1) &= \text{Max}\{0 + f(\mathbf{3}, 1), 79+f(\mathbf{3}, 0)\} = \max\{0+130, 79+0\} = 130. \text{ Opt. decision - allot 0 volunteers to precinct 2 and continue as in state } (\mathbf{3}, 1). \\
f(\mathbf{2}, 2) &= \text{Max}\{0 + f(\mathbf{3}, 2), 79+f(\mathbf{3}, 1), 110+f(\mathbf{3}, 0)\} = \max\{0+216, 79+130, 110+0\} = 216. \text{ Opt. decision - allot 0 volunteers to precinct 2 and continue as in state } (\mathbf{3}, 2).
\end{aligned}$$

Stage 2 contd.

$$f(2, 3) = \text{Max}\{0 + f(3, 3), 79+f(3, 2), 110+f(3, 1), 130+f(4, 0)\} = \max\{0+250, 79+216, 110+130, 130+0\} = 295. \text{ Opt. decision - allot 1 volunteer to precinct 2 and continue as in state } (3, 2).$$

$$f(2, 4) = \text{Max}\{0 + f(3, 4), 79+f(3, 3), 110+f(3, 2), 130+f(4, 1), 140+f(3, 0)\} = \max\{0+280, 79+250, 110+216, 130+130, 140+0\} = 329. \text{ Opt. decision - allot 1 volunteer to precinct 2 and continue as in state } (3, 3).$$

$$f(2, 5) = \text{Max}\{0 + f(3, 5), 79+f(3, 4), 110+f(3, 3), 130+f(3, 2), 140+f(3, 1), 125+f(3, 0)\} = \max\{0+290, 79+280, 110+250, 130+216, 140+130, 125+0\} = 360. \text{ Opt. decision - allot 2 volunteers to precinct 2 and continue as in state } (3, 3).$$

Stage 1

$$f(1, 5) = \text{Max}\{0 + f(2, 5), 35+f(2, 4), 42+f(2, 3), 56+f(3, 2), 50+f(2, 1), 50+f(2, 0)\} = \max\{0+360, 35+329, 42+295, 56+216, 50+130, 50+0\} = 364. \text{ Opt. decision - allot 1 volunteer to precinct 1 and continue as in state } (2, 4).$$

By following the optimum decisions beginning with state (1, 5), we see that an optimum strategy is to allot 1 volunteer each to precincts 1, 2, 3, and the remaining 2 volunteers to precinct 4. This yields the maximum expected additional votes of 364 (in units of hundreds).

In this section we discussed a family of simple allocation models involving the distribution of a single discrete resource among various activities. These models can be generalized to encompass situations in which activities require two or more resources, but the number of states needed to represent multiple resource allocation problems for solution by DP grows very rapidly with the number of resources. This unfortunate aspect of DP is called the **curse of dimensionality**.

Summary

In this chapter we introduced the recursive technique of dynamic programming, and illustrated its application to several discrete deterministic optimization problems that can be posed in a sequential decision format. The basic principles behind DP have been in use for many years, but it was R. Bellman who in the 1950s developed it into a systematic tool and pointed out its broad scope. Now, dynamic programming is a powerful technique with many applications in production planning and control, optimization and control of chemical and pharmaceutical batch and continuous processes, cargo loading, inventory control, equipment replacement and maintenance, and in finding optimal trajectories for rockets and satellites. Our treatment of the subject has been very elementary since our aim is mainly to introduce the concepts of systems and their states, optimum value functions, functional equations and the recursive technique for solving them, which are fundamental to DP. The books referenced at the end of this chapter should be consulted for advanced treatments of the subject.

10.9 Exercises

10.1: The US government is worried about increasing unemployment in states on the west coast due to rapid decline of timber-lands in those states

Year	New jobs created if \$ r mil. are spent in year								
	$r = 0$	1	2	3	4	5	6	7	8
1	0	5	15	40	80	90	95	98	100
2	0	5	15	40	60	70	73	74	75
3	0	4	26	40	45	50	51	52	53

by excessive lumbering activity. So, they recently authorized spending an additional \$8 mil. over the next 3 calendar years to create new jobs in alternate industries in those states. Because of programs going on already, the effectiveness of additional funds depends on when they are spent. Funds can only be released in integral multiples of \$1 mil. for

any year. The following table provides important data estimated by a panel of economists, with new jobs measured in units of 100.

Find an optimum policy for spending the funds over the planning horizon, which maximizes the total number of additional jobs created, using DP.

10.2: There are 4 types of investments. Each accepts investments only in integer multiples of certificates. We have 30 units of money to invest (1 unit = \$1000). Following table provides data on rewards obtained from investments in the different types.

Investment type	Cost (units/certificate)	Reward for buying r certificates				
		$r = 1$	2	3	4	5
1	3	2	3	8	16	23
2	2	1	2	4	7	12
3	4	4	8	15	24	30
4	6	4	9	23	36	42

At least one certificate of each type must be purchased. Use DP to determine the optimum number of certificates of each type to buy to maximize total reward.

10.3: A production process is available for 3 periods. In each period it can produce an integer number of units of a commodity between 0 to 4. A total of 6 units of the commodity must be produced by the end of period 3.

Period	Production cost (in \$100s) if r units produced				
	$r = 0$	1	2	3	4
1	0	4	8	9	12
2	0	7	10	11	15
3	0	8	11	15	16

Units produced in period 1 (period 2) have to be stored at a cost of \$2/unit (\$1/unit) till the end of period 3. Those produced in period 3 incur no storage cost. Other data is given above. Determine an

optimum production plan to minimize the total cost of production and storage for meeting the requirement.

10.4: A batch of a chemical consisting of 6 tons, contains the chemical in particle sizes 1, 2, and 3 in equal proportion (size 1 is smaller than size 2 which in turn is smaller than size 3).

The company has 2 sieves. Sieve 1 transfers particles of size 1 to the bottom and leaves everything else on top. Sieve 2 leaves particles of size 3 at the top, but transfers everything else to the bottom. To use either sieve, a minimum of 2 tons of material must be fed. Each use of either sieve costs \$10. Data on the selling price of the chemical is given below. Determine the maximum amount of money that can be made with the existing batch of the chemical.

No.	Chemical containing particle sizes	Price/ton
I	1, 2, 3	\$40
II	1,2 only	\$55
III	1 only	\$60
IV	2, 3 only	\$50
V	3 only	\$70
VI	2 only	\$45

10.5: The major highways in Michigan are US-23, I-94, I-96 and I-75. The state highway department is concerned about the ever increasing number of speed limit violators on these highways. To control the problem they have decided to put 7 new patrol cars on these highways. The following data represents the best estimates of the number of violators ticketed per day.

Highway	Expected no. ticketed/day if r new patrol cars assigned			
	$r = 0$	1	2	3
I-94	30	70	100	140
US-23	20	45	80	115
I-75	10	20	40	65
I-96	20	40	90	110

Determine an optimum allocation of new patrol cars to the various highways (no more than 3 for any highway) using DP.

10.6: A hi-tech company has perfected a process of growing crystalline silicon rods in 10 inch lengths. Profit obtained by selling a silicon rod depends on its length as given below.

Length (in.)	1	2	3	4	5	6	7	8	9	10
Profit (\$)	60	125	185	235	260	340	360	400	440	475

The cutting tool only accepts rods whose length in inches is an integer ≥ 2 , and it cuts the rod into two pieces whose lengths in inches are integers. Each use of the cutting tool costs \$10. The pieces obtained from a cut can be cut again if they satisfy the conditions mentioned above. Determine an optimum cutting policy for each 10 inch rod, to maximize the net profit from it.

10.7: A company has 5 identical machines that it uses to make four products A, B, C, and D. Each machine can make any product, and when it is set up to make a product, a production run of one week is scheduled. The following table gives a forecast for the coming week's profit depending on how many machines are scheduled to produce each product.

No. of machines	Week's forecasted profit (\$10,000 units) from product			
	A	B	C	D
0	0	0	0	0
1	12	17	5	8
2	17	30	12	14
3	25	49	22	25
4	35	64	34	35
5	45	76	48	43

Determine the optimum number of machines to allot to each product for the coming week to maximize the total profit.

10.8: The EPA got into a lot of bad publicity recently about lax monitoring of dioxin contamination of Michigan rivers. EPA divides the state into 3 regions. The following table gives data on the number of tests that can be conducted in each region by allotting some inspectors. EPA is willing to appoint 5 inspectors. Determine how many of these to allot to each region (this should be a nonnegative integer for each region) so as to maximize the total number of tests conducted over the whole state per month.

Region	No. of tests/month if r inspectors allotted				
	$r = 1$	2	3	4	5
1	25	50	80	117	125
2	20	70	130	150	160
3	10	20	35	40	45

10.9: There are 4 objects which can be packed in a vessel. Objects 1, 2, 3 are available in unlimited number of copies; but only four copies of object 4 are available. The weight of each object and the capacity of the vessel are expressed in weight units, and values in money units, in the following table.

object	Wt. per copy	Value/copy if no. included is				
		1	2	3	4	5
1	3	2	3	8	16	23
2	2	1	2	4	7	12
3	4	4	8	15	24	30
4	6	4	9	23	36	

The vessel's weight capacity is 30 weight units. The objective function is total value, and it is additive over objective types. Find the maximum objective value, subject to the constraint that at least one copy of each object must be included.

10.10: A company has four salesmen to allocate to three marketing regions. Their objective is to maximize the total sales volume generated. The sales growth in each region is expected to go up as more salesmen are allocated there, but not linearly. The company's estimates of the sales volume as a function of the number of salesmen allocated to each region are given below.

Region	Sales volume if r salesmen allotted				
	$r = 1$	2	3	4	5
1	25	50	60	80	100
2	20	70	90	100	100
3	10	20	30	50	60

Each salesman has to be allotted to one region exclusively, or his employment can be terminated. Formulate the problem of determining how many salesmen to allot to each region so as to maximize the total sales volume as a DP and solve it.

10.11: A resource may be used on either or both of two processes. Each unit of resource generates \$4, \$3 when used for a day on the first process, second process, respectively. The resource can be recycled, but in recycling, a fraction is lost owing to usage and wastage. Thus, of the units used on the first process (second process) only half (two-thirds) remain for use the following day. 100 units of the resource are available at the start of a 10-day period, at the end of which, any units remaining will have no value. Determine how many units should be used on each process (fractions of units are allowed) on each of the ten days in order to maximize the total return. ([P. Dixon and J.M. Norman, 1984])

10.12: A student has final examinations in 3 courses, X , Y , and Z , each worth the same number of credits. There are only 3 days available for study. Assume that the student has to devote a nonnegative integer number of the available days for studying for each course, i.e., a day cannot be split between two courses. Estimates of expected grades based upon various numbers of days devoted for studying for each

course are given below.

Course	Expected grade if days of study is				
	0	1	2	3	4
<i>X</i>	0	1	1	3	4
<i>Y</i>	1	1	3	4	4
<i>Z</i>	0	1	3	3	4

(a) Determine the number of available days that the student should devote to each course in order to maximize the sum of all the grades using DP.

(b) How does the strategy in (a) change if the student has 4 days available to study before the examinations? What is the increase in the optimum objective value?

(c) How do the strategies in (a), (b) change if a new course *W* is added, with expected grade of 0, 0, 2, 3, and 4 when the number of days devoted to studying it is 0, 1, 2, 3, and 4 respectively? (**S. M. Pollock**).

10.13: A spaceship is on its way to landing on the moon. At some point during its descent near the moon, it has ϕ units of fuel, a downwards velocity of v towards the surface of the moon, and an altitude z above the surface of the moon. Time is measured in discrete units, and actions are only taken at integer values of time until the spaceship touches down.

At each integer value of time t , you can select an amount y of fuel to use, which will result in new variable values at time $t + 1$ of

$$\begin{aligned}\phi' &= \phi - y && \text{(depleted by } y \text{ units of fuel)} \\ v' &= v + 5 - y && \text{(the force of gravity is "5")} \\ z' &= z - v' && \text{(altitude decreases by } v')\end{aligned}$$

If z ever becomes negative, or if $v > 0$ when z becomes 0, the spaceship is fully destroyed.

Given initial (i.e., at time point 0 in this portion of the spaceship's trajectory) fuel, velocity, and altitude values of Φ , V , and Z , solve using

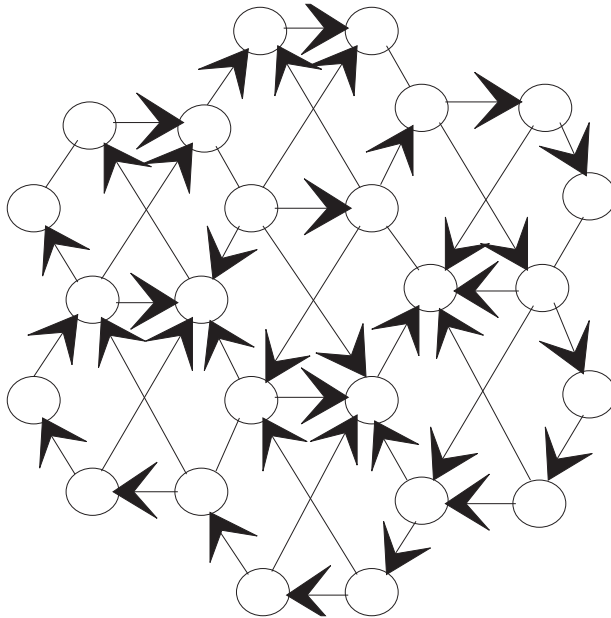


Figure 10.14:

DP the problem of reaching the point $(v = 0, z = 0)$ safely, using
(a) the OVF $f(\phi, v, z)$; **(b)** the OVF $g(v, z)$ defined below.

$f(\phi, v, z)$ = maximum amount of fuel remaining when the space-
 ship safely lands at $(v = 0, z = 0)$, given it is at
 (ϕ, v, z) at time point 0.

$g(v, z)$ = minimum amount of fuel required to safely reach
 $(v = 0, z = 0)$, given it is at (v, z) at time point
 0.

(c) Assuming that all variables are integer valued, what is the com-
 putational effort involved in solving (a)? Solve the problem numerically
 when $\Phi = 100$, $V = 20$, $Z = 300$. (S. M. Pollock)

10.14: Check whether the network in Figure 10.14 is acyclic. Find
 the acyclic numbering of its nodes if it is.

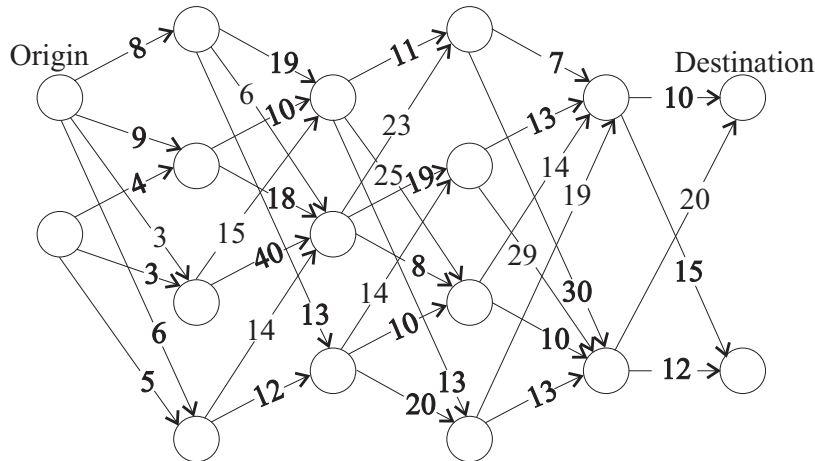


Figure 10.15:

10.15: Find the shortest route from the origin to the destination in the network in Figure 10.15.

10.16: There are 4 objects available for loading into a knapsack of unlimited weight capacity. Data on the objects is given below.

Object i	1	2	3	4
Value v_i	7	16	19	15
Weight w_i	3	6	7	5

An unlimited number of copies of each object are available for loading into the knapsack. Define

$$g(t) = \text{the minimum total weight of items needed in order to achieve a total value of at least } t \text{ in the knapsack.}$$

Find $g(t)$ and the associated (complete) optimal policy for nonnegative integers $t = 0$ to 100. (S. M. Pollock)

10.17: Find the shortest route from the origin to the destination

in the network in Figure 10.16.

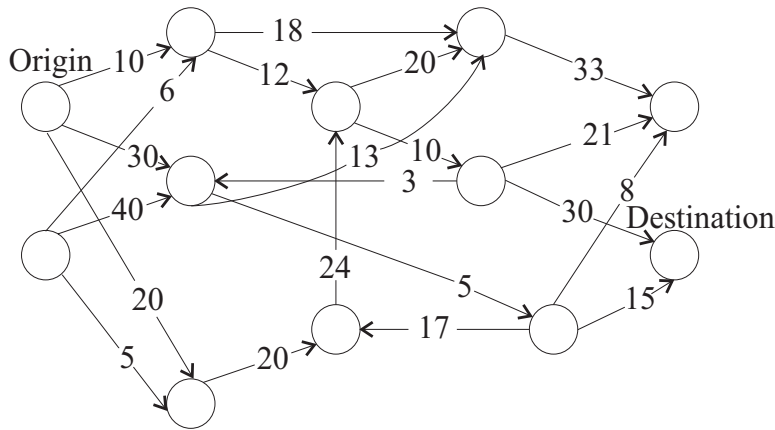


Figure 10.16:

10.18: A person wants to cross an uninhabited desert that is 100 miles wide in a jeep. The jeep is heavy and the sand is soft, so he gets only 3 miles/gallon. Gasoline can be purchased in unlimited quantities at the beginning of the desert, but once the jeep enters the desert no gasoline can be purchased until the desert is completely crossed. The jeep has a carrying capacity of 20 gallons of gasoline which includes gasoline consumed while travelling.

The driver plans to cross the desert by using the following procedure. Fill up the jeep at a depot at the beginning of the desert, and drive into the desert to a spot (call it the first “temporary gas dump”) where some gasoline is unloaded and stored, and then drive back to the depot to load up again. Continue this process until there is enough gasoline stored up at the first temporary gas dump so that the driver can use this as a new “depot” to continue past it into the desert.

Formulate the problem of minimizing the total quantity of gasoline needed to cross the desert by this procedure, as a DP. Solve your formulation and find the minimum amount of gasoline needed, and the policy that attains it. ([D. Gale, 1970]).

10.10 References

E. V. DENARDO, 1982, “*Dynamic Programming Models and Applications*”, Prentice Hall, Englewood Cliffs, NJ 07632.

S. E. DREYFUS and A. M. LAW, 1977, “*The Art and Theory of Dynamic Programming*”, Academic Press, NY.

P. DIXON and J. M. NORMAN, 1984, “An Instructive Exercise in Dynamic Programming”, *IIE Transactions*, 16, no. 3, 292-294.

D. GALE, 1970, “The Jeep Once More or Jeeper By the Dozen”, *American Math Monthly*, 77, 493-501. Correction published in *American Math Monthly*, 78 (1971) 644-645.

K. G. MURTY, 1992, *Network Programming*, Prentice Hall, Englewood Cliffs, NJ.

Index

For each index entry we provide the section number where it is defined or discussed first.

Acyclic network 10.4

Not staged 10.4, 10.5

Staged 10.4

Additivity over time 10.1

Back substitution 10.2

Backwards recursion 10.2, 10.3

Decision sets 10.3

Dynamic programming (DP) 10.1

Deterministic 10.1

Stochastic 10.1

Functional equations 10.3

Jeep problem 10.9

Knapsack problem 10.1

Nonnegative integer 10.1, 10.6

0-1; 10.1, 10.7

Markovian property 10.3

Models 10.1

Dynamic 10.1

Multistage 10.1

Single stage 10.1

Multistage 10.1

Optimum value function (OVF) 10.3

Policies 10.3

Principle of optimality 10.3

Recursive equations 10.3

Resource allocation problems 10.8

Sequential decision processes 10.1

Shortest routes 10.4

State space 10.3