Dynamic Programming

Many modern optimization problems involve large number of variables and/or constraints. Solutions of such problems often involve rather lengthy computations, and some of the solution techniques may even turn out to be computationally infeasible. *Dynamic Programming* (DP) is an approach that is designed to economize the computational requirements for solving large problems. The basic idea in using DP to solve a problem is to split up the problem into a number of stages. Each stage is associated with one subproblem, and the subproblems are linked together by some form of recurrence relations. The solution of the whole problem is obtained by solving these subproblems using recursive computations.

The development of DP in its early stages was largely due to Richard Bellman, whose book "*Dynamic Programming*" was published in 1957. Since then, DP has been found to be applicable to a whole variety of problems and is especially suited for the solution of the class of problems requiring interrelated decisions, i.e. decisions which must be made in a sequence and which influence future decisions in that sequence. DP is very simple in concept; the main difficulty in applying this approach, however, is the lack of a clear-cut formulation and solution algorithm. Consequently, we shall try to acquaint ourselves with this approach through the study of a wide variety of examples.

1 A Simple Path Problem

We begin with a problem seeking the shortest path from one physical location to another.

Example 1. Suppose that we have to find the shortest path from A to B given in Figure 1 (the numbers in Figure 1 are the lengths of the arcs). We could, of course, solve this problem by enumerating all possible paths from A to B, adding up the lengths of each, and then choosing the smallest such sum. Brute-force methods like this are usually referred to as *enumeration methods*. Alternatively, let us consider the more efficient DP approach:

Let us define $S(A)$ to be the length of the shortest path from A to B. Similarly, $S(i)$ is the length of the shortest path from node i to B for every node i. We try to establish relations between $S(A)$ and $S(i)$ by making the following observation. If the shortest path from a node i to B passes through a node j, then the section of this path from node j to B must also be a shortest path from node j to B. In fact, if it is not, then we can replace the section by the shortest path from node j to B to get a short path from node i to B.

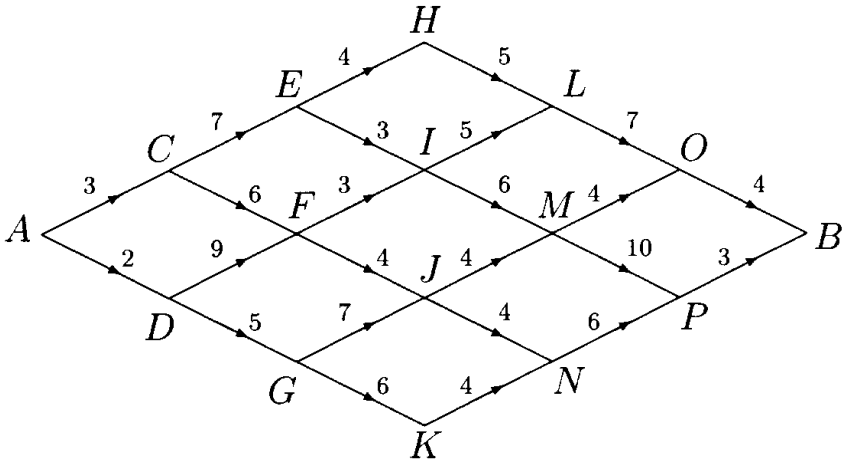

Fig. 1. Network for Example 1

With this observation, we can assert that

$$S(A) = \min\{a_{AC} + S(C), a_{AD} + S(D)\}, \tag{1}$$

where a_{AC} and a_{AD} are the arclengths from node A to C and to D respectively. This is because the shortest path from A to B must either pass through node C or D. If the shortest path passes through C, then $S(A) = a_{AC} + S(C)$ by the above observation; otherwise $S(A) = a_{AD} + S(D)$. Hence $S(A)$ must be equal to the minimum of $a_{AC} + S(C)$ and $a_{AD} + S(D)$.

Repeating this argument, we can set up a recurrence relation as follows:

$$\begin{aligned}
S(A) &= \min\{3 + S(C), 2 + S(D)\} \\
S(C) &= \min\{7 + S(E), 6 + S(F)\} \\
S(D) &= \min\{9 + S(F), 5 + S(G)\} \\
&\vdots \\
S(O) &= 4 + S(B) \\
S(P) &= 3 + S(B).
\end{aligned}$$

Clearly we can see that $S(B) = 0$. We can compute the values of $S(i)$ recursively by considering nodes further and further away from B:

$$\begin{aligned}
&S(O) = 4 + S(B) = 4;\ S(P) = 3 + S(B) = 3 \\
&S(L) = 7 + S(O) = 11;\ S(M) = \min\{4 + S(O), 10 + S(P)\} = 8; \\
&S(N) = 6 + S(P) = 9;\ \cdots \\
&S(A) = \min\{3 + S(C),\ 2 + S(D)\} = 25.
\end{aligned}$$

In the process of calculating the values of the function S, say $S(M)$, the value $S(M)$ is obtained from the term $4+S(O)$ since $4+S(O)<10+S(P)$. Consequently we know that the next node on the shortest path from M to B must be O. To keep track of this, we define a function $P(i)$ to be the next node on the shortest path from node i to B. Thus $P(M)=O$. Similarly we can find $P(i)$ at every node i when we are calculating $S(i)$.

The solution of the simple shortest path problem is now readily seen. The length of the shortest path from A to B is given by $S(A)=25$. The shortest path is obtained by following the direction given by $P(i)$. $P(A)=C$; $P(C)=F$; $P(F)=J$; $P(J)=M$; $P(M)=O$; $P(O)=B$. So the shortest path is $A-C-F-J-M-O-B$. □

2 Terminology and Comments

The procedure described in the previous section is quite a typical DP solution procedure. Let us now introduce some terminology and comments associated with this procedure.

(i) Optimal value function: The function $S(i)$ in Example 1 is called an *optimal value function*, and i is called the *argument* of the function. To seek the value of $S(i)$ at some node i is then a subproblem of the original problem. There is no fixed rule as to the setting up of these optimal value functions. Different ways of defining these optimal value functions may mean different ways of splitting up the given problem into subproblems. As in the above example, we could have defined the optimal value function $S(i)$ to be the length of the shortest path from node A to i, see Exercise 2 below.

(ii) Principle of optimality: In Example 1, we define an optimum path as the shortest path. If we had defined an optimum path as the path of longest length (lowest cost, highest cost, etc.), we could still use the same technique to find the optimum path. The key feature here is that "any subpath of an optimum path must be optimum itself from the current state to the final stage". This seemingly obvious statement can be generalized to many other problems, such as multistage decision problems, allocation processes, and so forth. Instead of using the word "path", we use the word "policy" which is more suitable to decision problems. This idea has been called the "*principle of optimality*" by people using DP and it says:

ANY SUBPOLICY OF AN OPTIMUM POLICY FROM ANY GIVEN STATE MUST ITSELF BE AN OPTIMUM POLICY FROM THAT STATE TO THE TERMINAL STATE(S).

In other words, the subpolicy from a given state to the terminal state(s) is independent of the policy adopted previously.

(iii) Recurrence relation (Functional equation): If an optimal value function has been suitably chosen, then using the principle of optimality one will be able to define a *recurrence relation* between the values of the optimal value function. The existence of such a recurrence relation makes DP well suited to computer solutions, as Richard Bellman mentioned in his book *Applied Dynamic Programming* that DP is "a new approach based on the use of functional equations and the principle of optimality, with one eye on the potentialities of the burgeoning field of digital computers". In Example 1, the recurrence relation at the node A takes the form

$$S(A) = \min\{a_{AC} + S(C), a_{AD} + S(D)\}.$$

At this node, one can decide to go "up" along the arc AC or go "down" along the arc AD. Should the decision be "up", then the length of the path is the sum of a_{AC} and the optimal length of the remaining path. In DP terminology, a_{AC} denotes the *immediate return* of the *decision* "up". The optimal value (e.g. $S(A)$) is therefore given by choosing the decision that optimizes the sum of the immediate return and the optimal value of the remaining process.

(iv) Boundary conditions: Using the recurrence relation, the values of the optimal value function at different arguments can be found one by one. Of course, the process must start with arguments at which the values of the optimal value function are obvious, and these obvious values are called the *boundary conditions*, e.g. $S(B) = 0$ is obvious in Example 1.

(v) Optimal policy function: The rule that associates the best first decision with each subproblem—the function P in Example 1—is called the *optimal policy function*.

Using these jargons, to solve a problem by means of DP can be described simply as follows:

1. Define an optimal value function.
2. Using the principle of optimality, determine a recurrence relation.
3. Identify the boundary conditions. Starting with the boundary conditions, and using the recurrence relation, determine concurrently the optimal value and policy functions.
4. Determine the solution of the problem by using the optimal value and policy functions.

This procedure seems very simple. However, the main difficulty when using DP is in choosing a suitable optimal value function for which a recurrence relation can be determined; there is no fixed rule to follow and it really depends on one's ingenuity to apply this technique successfully.

Let us point out one major difference between DP and LP: LP refers to a specific mathematical model (i.e. an optimization problem with linear objective function and linear constraints) that can be solved by a variety of

techniques, most notably being the simplex method. DP deals with a particular analytic approach, which can be applied to a variety of mathematical models.

Exercise 2. Notice that for a given problem, one can have different ways to define the optimal value function. Solve the simple shortest path problem in Figure 1 with the optimal value function $T(i)$ defined to be the length of the shortest path from node A to i.

Example 3. DP is a versatile approach. Once we grasp the main idea, we can solve many similar but more complicated problems that are seemingly difficult to solve. Consider the shortest path problem in Figure 1 again. But let us redefine the cost of a path to be the largest cost between two nodes on the path. If we let $S(i)$ be the length of the shortest path from node i to B for every node i, where the length is measured using the new definition, then instead of (1), we have

$$S(A) = \min\{\max\{a_{AC}, S(C)\}, \max\{a_{AD}, S(D)\}\}.$$

Similarly, we can establish the recurrence relation for other nodes. Obviously, the boundary condition is still $S(B) = 0$. From that we can get the optimal solution $S(A) = 6$ and the optimal path is $A - C - F - I - M - O - B$ and $A - D - G - K - N - P - B$.