DYNAMIC PROGRAMMING

Dynamic programming is a mathematical technique for optimizing a multistage decision process. It is an approach to optimization, rather than a technique for actually optimizing the objective function. It transforms a problem into a different form composed of a series of recursive *subproblems* frequently more suitable for optimization. The task of breaking down a given problem is a creative step for which no general method is known. Once the problem is transformed, the actual optimization is carried out recursively by introducing the notation of *state* which couples the underlying subproblems.

Historically, the first known use of dynamic programming is traced back to Newton, who used this technique to solve a problem proposed by one of the Bernoulli brothers. This technique was developed in the early 1950s by Richard Bellman who also chose the name "dynamic programming" (1). Since then, dynamic programming has been applied to a variety of optimization problems, including *optimal control* (2), *neural networks* (3), and *communications* (4).

The objective in dynamic programming is to minimize a certain cost function which is a mathematical expression for a desirable outcome. In this technique, decisions regarding a certain problem are typically optimized in *stages* rather than simultaneously. This generally signifies that the original decision problem is divided into a sequence of small subproblems (stages) which then are handled more efficiently from the computational viewpoint. In this process, we need to determine how to break our problem down into a sequence of subproblems, and we also need to know how to solve a specific subproblem in the sequence, given that the solutions to all previous subproblems are known. The main point is that decisions can not be viewed in isolation because the desire for low present cost must be balanced with the inevitability of high future costs. At each stage, one selects a decision to minimize the sum of the current stage cost and the best cost that can be expected from future stages. The dependence between different stages is reflected through a set of states which connect subsequent stages.

The big skill in dynamic programming and the art involved is taking a problem and determining stages and states for an efficient solution. To identify the stages we must imagine how the problem can be analyzed sequentially. To carry out a stagewise analysis, the state variables are introduced which summarize the previous decisions compactly. The decision-making process at each stage involves selecting one of the alternatives of the stage. This is usually referred to as a stage decision. Associated with each stage decision is a return function which evaluates the alternative selected by this decision in terms of its contribution to the returns of the entire problem. By selecting an optimal feasible alternative for each stage, then the selected set of alternatives comprises an optimal policy for the entire problem. The solution is obtained in an orderly manner by going from one stage to the next and is completed after the final stage is reached. The computational efficiency of dynamic programming stems from the fact that the optimum solution is obtained by considering one stage at a time.

PRINCIPLE OF DYNAMIC PROGRAMMING

Any problem which requires identifying the optimum of a function in N variables can be expressed in a standard form: *minimize* or *maximize* $f(x_1, x_2, \ldots, x_N)$ subject to given constraints on the values of $x_1, x_2, \ldots, x_N$. The list of functions and types of problems which give rise to this format is quite extensive: linear programming, geometric programming, network optimization, and so on.

We shall alter this from one problem with N variables, whose values we try to find simultaneously, to a succession of problems each associated with one of N stages. A stage here signifies a portion of the decision problem for which a separate decision can be made. The resulting decision must also be meaningful in the sense that, if it is optimal for the stage it represents, then it can be used directly as part of the optimal solution to the entire problem.

As already mentioned, separation between successive stages is achieved in dynamic programming by the concept of the state. In each of these stages, we have problems to be solved with only one variable. Then, we try to find the best value of a particular decision variable for that stage. Because it is not possible to know the consequences of the other $N - 1$ decisions, it is generally essential to find the best value for decision variables for several different states at each stage. Thus, at any stage, a state summarizes the current "status" of the system which permits a feasible decision for the current stage without having to "look back" and study the effect of this decision on the stages previously considered. This usually means that we have to solve more than N subproblems with one variable. When solving each of these subproblems, we must assume that the other variables have taken different possible values or that different amounts of some resource are available when the decision is made at the stage being looked at.

For descriptive ease, it is often convenient to consider all dynamic programming problems as sequential in time. Then, each variable corresponds to a decision made at a specific epoch or moment. A simple example of a sequential decision problem is How should production of an item be managed, where the costs of production and storage vary, and the demand is random? We assume that each stage of the problem corresponds to a period of one week and the states for different stages correspond to the number of items in storage at the beginning of the corresponding week. Now assume that at the start of the problem, the "system" had four items. A decision had to be made about the number (say, x_1) to be produced during the first week, and the random demand d_1 during that week means that the state of the system at the beginning of the second week is $4 + x_1 - d_1$. Then, the second decision is made. The random demand of the second week means that the decision maker starts the third week in a new state, and similarly in the fourth week. The straightforward approach to solving the problem is to calculate the total cost of managing this small company for four weeks with all of the possible decisions that might be made at the start of each week and all the possible random demands that could occur. However, instead of such brute force methods of total enumeration, dynamic programming relies on a principle, the so-called principle of optimality, which facilitates identification of optimal policies.

The principle of optimality (also known as Bellman's principle) says that, if we want to know the best decision which can be made from a given state and stage of the problem, we must consider each decision and each state to which that decision would lead (at the next stage). However, one does not need to go any further than the next stage. After that an optimal policy is followed which can be found by comparing the outcomes from each decision and the optimal policy from the resulting states.

One of the variant forms of the principle of optimality implies that an optimal policy is independent of the past and looks only to the future. This is an essential part of dynamic programming, because it allows calculating policies recursively. It means that the identification of the state must be sufficient to fully describe the system, so that this independence may be observed and so that the problem is effectively decomposed into a series of one-dimensional problems each of which depends on the solution to later problems but not on the solution to earlier ones.

A common example of the application of dynamic programming is in solving the so-called shortest path problem which is explained following by an example (5).

SHORTEST PATH PROBLEM

Consider the graph shown in Fig. 1 where the nodes correspond to a set of cities connected through some paths. There is a travel cost for each path. The total cost of a journey is obtained by adding the costs of its constituent paths. Assume that a traveler wishes to travel from city A to city J. We are looking for the route from A to J with the minimum overall cost. The problem appears as a series of decision. In each city visited, the traveler has a choice of several paths to take and must decide on one of them. The dynamic programming formulation is composed of four stages in which the states correspond to the cities. The recurrence relationship which defines the cost of a policy is as follows: If $f_n^*(i_n)$ is the cost of an optimal policy when there are n stages remaining and the decision is made in state i_n, then

$$f_n^*(i_n) = \min_{i_{n-1}}[p(i_n, i_{n-1}) + f_{n-1}^*(i_{n-1})] \tag{1}$$

where $p(a, b)$ is the cost of the path from a to b and $f_o^*(J) = 0$ (J is the destination). We solve the problem starting from J and moving backward to the starting point. At the last stage, there are two possible states for the traveler, H and I. In each of these the traveler has no choice of destinations, but must go to J directly. Thus,

$$f_1^*(H) = 23 + f_0^*(J) = 23 + 0 = 23 \tag{2}$$

and similarly

$$f_1^*(I) = 29 + f_0^*(J) = 29 + 0 = 29 \tag{3}$$

When there are two stages left, the traveler can be in any of the three states E, F and G. In each one, there is a choice of

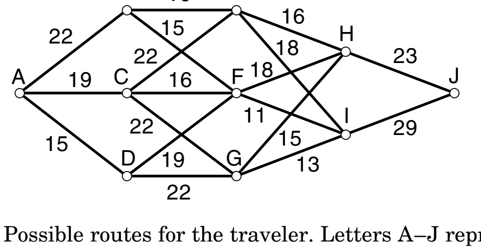

Figure 1. Possible routes for the traveler. Letters A–J represent the cities; the journey starts at A and ends at J.

two destinations, H and I. If the traveler goes from E to H, the cost of this stage is 16 and then the traveler follows the optimal policy from H to J. If the traveler goes from E to I, then the traveler will pay 18 for the single stage, followed by the cost of the optimal policy from I to J. So, we can calculate as follows:

$$\begin{aligned} f_2^*(E) &= \min[16 + f_1^*(H), 18 + f_1^*(I)] \\ &= \min[16 + 23, 18 + 29] \\ &= 39 \text{ (corresponding to deciding to go to} H) \end{aligned}$$

Similarly

$$\begin{aligned} f_2^*(F) &= \min[18 + f_1^*(H), 11 + f_1^*(I)] \\ &= \min[18 + 23, 11 + 29] \\ &= 40 \text{ (corresponding to deciding to go to} I) \end{aligned}$$

and

$$\begin{aligned} f_2^*(G) &= \min[15 + f_1^*(H), 13 + f_1^*(I)] \\ &= \min[15 + 23, 13 + 29] \\ &= 38 \text{ (corresponding to deciding to go to} H) \end{aligned}$$

One stage earlier, there are three states in which the traveler could be found, B, C, and D. We find that

$$\begin{aligned} f_3^*(B) &= \min[16 + f_2^*(E), 15 + f_2^*(F)] \\ &= \min[16 + 39, 15 + 40] = \min(55, 55) \\ &= 55 \text{ (it does not matter which decision is made)} \\ f_3^*(C) &= \min[22 + f_2^*(E), 16 + f_2^*(F), 22 + f_2^*(G)] \\ &= \min[22 + 39, 16 + 40, 22 + 38] \\ &= 56 \text{ (corresponding to deciding to go to} F) \\ f_3^*(D) &= \min[19 + f_2^*(F), 22 + f_2^*(G)] \\ &= \min[19 + 40, 22 + 38] \\ &= 59 \text{ (corresponding to deciding to go to} F) \end{aligned}$$

One stage earlier, the traveler has a choice of three routes. These lead to a recurrence relationship:

$$\begin{aligned} f_4^*(A) &= \min[22 + f_3^*(B), 19 + f_3^*(C), 15 + f_3^*(D)] \\ &= \min[22 + 55, 19 + 56, 15 + 59] \\ &= 74 \text{ (corresponding to deciding to go to} D) \end{aligned}$$

So we have found the optimum policy for the whole journey which is the route A, D, F, I, J with a total cost of 74 units. We continue our discussion with a more advanced example (5).

RIVER CROSSING

One of the popular puzzles frequently appears in the form of arranging groups to cross a river. Usually, it is posed as a story such as the following. Three explorers are traveling in a jungle with three cannibals. They reach a wide river and build a raft big enough for two people. All six people want to cross the river. However, there should never be more cannibals than explorers on either bank. How should the party cross the river safely in the shortest possible time?

The relationship of such puzzles to dynamic programming is evident as one considers that there is a sequence of decisions to be made after each crossing. Suppose that we define the state of the system using the number of people (E_1, C_1) on the first bank of the river where the two components denote the number of explorers and cannibals, respectively. With the empty raft there, there are ($3 - E_1$, $3 - C_1$) on the opposite bank. Assume that (e_1, c_1) and (e_2, c_2) denote the number of (explorers, cannibals) on a forward and return journey, respectively. Clearly, there is a restricted set of values for the four numbers (e_1, c_1, e_2, c_2) because the raft cannot hold more than two people and the number of explorers must not be exceeded by the number of cannibals on either bank. The double crossing is regarded as a stage in the sequential problem. We want to minimize the number of stages needed to achieve the target state (0, 0) under an optimal policy:

$$f^*(E_1, C_1) = 1 + \min[f^*(E_1 - e_1 + e_2,\ C_1 - c_1 + c_2)] \quad (4)$$

with the minimum taken over all the permissible sets of values of the four variables (e_1, c_1) and (e_2, c_2). There are only a limited number of feasible states for the problem, namely,

$$(0,0), (0,1), (0,2), (0,3), (1,1), (2,2), (3,1), (3,2), (3,3) \quad (5)$$

and the recurrence relationship yields the following:

$$\begin{aligned} f^*(0,0) &= 0 \\ f^*(0,1) &= 1 \\ f^*(0,2) &= 1 \\ f^*(0,3) &= 1 + \min[f^*(2,2), f^*(0,2), f^*(1,1)] \\ f^*(1,1) &= 1 \\ f^*(2,2) &= 1 + f^*(0,3) \\ f^*(3,1) &= 1 + \min[f^*(3,2), f^*(2,2)] \\ f^*(3,2) &= 1 + \min[f^*(3,1), f^*(3,3)] \\ f^*(3,3) &= 1 + f^*(3,2) \end{aligned}$$

Solving these recursive subproblems results in $f^*(3, 3) = 6$ where the sequence of states (traced backward) is as follows: (3, 3), (3, 2), (3, 1), (2, 2), (0, 3), (0, 2), (0, 0). This means that the raft must cross the river 11 times in total (five double crossings plus a single final trip).

PRINCIPLE OF DECOMPOSITION

As already mentioned, the first step in every dynamic programming problem is to transform the original problem into some small subproblems. In mathematical notations, our objective is to decompose the problem

$$\begin{aligned} f_N(x_N) &= \min_{d_N,\ldots,d_1} g[r_N(x_N, d_N), \ldots, r_1(x_1, d_1)] \\ \text{subject to}\quad x_{n-1} &= t_n(x_n, d_n), \quad n = 1, \ldots, N \end{aligned}$$

into N equivalent subproblems each containing only one state variable ($x_i's$) and one decision variable ($d_i's$). To achieve this decomposition, the function $g(.)$ should have a special form which is explained in the following (6). Let

$$\begin{aligned} g[r_N(x_n, d_N), r_{N-1}(x_{N-1}, d_{N-1}), \ldots, r_1(x_1, d_1)] \\ = r_N(x_N, d_N) + r_{N-1}(x_{N-1}, d_{N-1}) + \ldots + r_1(x_1, d_1) \end{aligned}$$

which results in

$$f_N(x_N) = \min_{d_N,\ldots,d_1} [r_N(x_N, d_N) + r_{N-1}(x_{N-1}, d_{N-1}) + \ldots + r_1(x_1, d_1)]$$
$$\text{subject to} \quad x_{n-1} = t_n(x_n, d_n), \quad n = 1, \ldots, N$$

Noting that (1) the Nth stage return does not depend on $d_{N-1}, \ldots, d_1$, and (2) for arbitrary real-valued functions $h_1(u_1)$ and $h_2(u_1, u_2)$,

$$\min_{u_1,u_2} [h_1(u_1) + h_2(u_1, u_2)] = \min_{u_1} [h_1(u_1) + \min_{u_2} h_2(u_1, u_2)]$$

We can rewrite the objective function in the following form:

$$f_N(x_N) = \min_{d_N} \{r_N(x_N, d_N) + \min_{d_{N-1},\ldots,d_1} [r_{N-1}(x_{N-1}, d_{N-1}) + \ldots + r_1(x_1, d_1)]\}$$
$$\text{subject to} \quad x_{n-1} = t_n(x_n, d_n), \quad n = 1, \ldots, N$$

Starting with the minimization over $d_{N-1}, \ldots, d_1$ is the crucial step in the decomposition. The minimum with respect to d_N, however, is still over $r_{N-1}, \ldots, r_1$, because x_{N-1}, depends on d_N through the stage transformation t_N. From the definition of $f_N(x_N)$, it follows that

$$f_{N-1}(x_{N-1}) = \min_{d_{N-1},\ldots,d_1} [r_{N-1}(x_{N-1}, d_{N-1}) + \ldots + r_1(x_1, d_1)]$$

So, there is a new form for $f_N(x_N)$:

$$f_N(x_N) = \min_{d_N} [r_N(x_N, d_N) + f_{N-1}(x_{N-1})]$$
$$\text{subject to} \quad x_{N-1} = t_N(x_N, d_N)$$

We consider $Q_N(x_N, d_N) = r_N(x_N, d_N) + f_{N-1}[t_N(x_N, d_N)]$ as the return function. Finally, in a recursive format,

$$\begin{aligned} f_n(x_n) &= \min_{d_n} Q_n(x_n, d_n), && n = 1, \ldots, N \\ Q_n(x_n, d_n) &= r_n(x_n, d_n), && n = 1 \\ &= r_n(x_n, d_n) + f_{n-1}[t_n(x_n, d_n)], && n = 2, \ldots, N \end{aligned}$$

We continue the discussion with an example. Consider the optimization problem

$$\begin{aligned} &\min && d_1^2 + d_2^2 + d_3^2, \\ &\text{subject to} && d_1 + d_2 + d_3 \geq k, \quad k > 0, \quad d_1, d_2, d_3 \geq 0 \end{aligned} \tag{6}$$

To put this problem into the appropriate form of

$$\min \quad r_1(x_1, d_1) + r_2(x_2, d_2) + r_3(x_3, d_3)$$
$$\text{subject to} \quad x_{n-1} = t_n(x_n, d_n), \quad n = 1, 2, 3$$

we introduce the state variables (x_0, x_1, x_2, x_3) and replace $d_1 + d_2 + d_3 \geq k$ by $x_3 \geq k$, $x_2 = x_3 - d_3$, $x_1 = x_2 - d_2$, and $x_0 = x_1 - d_1$. This is legitimate, because by adding these four equations, we obtain $d_1 + d_2 + d_3 \geq k - x_0$. So that $d_1 + d_2 + d_3 \geq k$, it is sufficient that $x_0 = 0$, or equivalently, $d_1 = x_1$. Because $d_1 = x_1 \geq 0$, $d_2 \leq x_2$, and similarly $d_3 \leq x_3$. We can restate the problem as follows:

$$\begin{aligned} &\min && d_1^2 + d_2^2 + d_3^2 \\ &\text{subject to} && x_1 = x_2 - d_2, \quad d_1 = x_1 \geq 0 \\ &&& x_2 = x_3 - d_3, \quad 0 \leq d_2 \leq x_2 \\ &&& x_3 \geq k, \quad 0 \leq d_3 \leq x_3 \end{aligned}$$

This is the appropriate form, because

$$r_n(x_n, d_n) = d_n^2$$
$$x_{n-1} = t_n(x_n, d_n) = x_n - d_n, \quad n = 1, 2, 3$$

The remaining restriction on the decision variables simply limits the feasible combinations of (x_n, d_n) and, in that sense, acts to our advantage. Having determined the appropriate definitions for r_n and t_n, we can state the problem in terms of the recursive equations of dynamic programming:

$$f_1(x_1) = \min_{d_1 = x_1} d_1^2,$$
$$f_n(x_n) = \min_{0 \leq d_n \leq x_n} [d_n^2 + f_{n-1}(x_n - d_n)] \quad n = 2, 3, \text{with } x_3 \geq k$$

The critical step is the proper interpretation of stages, decisions, returns, and transformations. We imagine that a nonnegative quantity x_3, $x_3 \geq k$, is divided into three quantities. Each is placed into a separate box marked 3, 2, and 1, respectively. Associated with each box is a decision, the quantity put in the box, and a return, the square of the quantity in the box. The total return is also determined by adding the returns from each of the boxes.

We have identified the stages (the boxes), the decisions, and the returns at each stage. We have used all information in the original problem except that the total quantity placed in all three boxes equals x_3. This constraint determines the relationships among stages and consequently among the stage transformations. We imagine that the division of the quantity x_3 is sequential. First, an amount d_3, $0 \leq d_3 \leq x_3$, is placed in box three. The quantity remaining to be divided between stages two and one is $x_2 = x_3 - d_3$. Likewise, d_2, $0 \leq d_2 \leq x_2$ is placed in box two, and $x_1 = x_2 - d_2$ remains. This remainder most be allocated to box one, so $d_1 = x_1$.

The solution procedure begins by finding $d_1(x_1)$ and $f_1(x_1)$. In terms of the multistage model, $d_1(x_1)$ is the optimal allocation at stage one as a function of x_1, and $f_1(x_1)$ is the optimal return from stage one that results from an allocation of $d_1(x_1)$. Because we have already established that $d_1(x_1) = x_1$, there is no optimization, and $f_1(x_1) = x_1^2$. The next step is to express the optimal one-stage return as a function of x_2 and d_2. Because

$$x_1 = x_2 - d_2$$
$$f_1(x_1) = (x_2 - d_2)^2$$

the return from stage two is d_2^2. Thus, the total return from stages two and one, given that stage one is operated optimally as a function of its input, is given by

$$Q_2(x_2, d_2) = d_2^2 + (x_2 - d_2)^2 \tag{7}$$

The optimal return from two stages as a function of x_2 is given by

$$f_2(x_2) = \min_{0 \le d_2 \le x_2} [d_2^2 + (x_2 - d_2)^2] \tag{8}$$

Setting the partial derivative of Q_2 with respect to d_2 to zero, the necessary condition for a minimum is given by

$$\frac{\partial Q_2}{\partial d_2} = 2d_2 - 2(x_2 - d_2) = 0 \tag{9}$$

This condition is also sufficient because the second derivative is positive. The unique solution is $d_2 = x_2/2$ and $f_2(x_2) = x_2^2/2 = (x_3 - d_3)^2/2$. Continuing for $n = 3$, in exactly the same manner,

$$f_3(x_3) = \min_{0 \le d_3 \le x_3} [d_3^2 + (x_3 - d_3)^2/2] \tag{10}$$

By partially differentiating, we obtain the solution $d_3 = x_3/3$ which yields $f_3(x_3) = x_3^2/3$. Clearly f_3 is minimum when $x_3 = k$, so

$$\begin{aligned} f_3(k) = k^2/3, d_3^* = k/3, x_2^* = k - k/3 = 2k/3, \\ d_2^* = x_2^*/2 = k/3, \text{ and } x_1^* = k/3 = d_1^* \end{aligned} \tag{11}$$

In the problem just considered, we actually solved

$$\begin{aligned} &\min \quad \textstyle\sum_{n=1}^{N} d_n^2 \\ \text{subject to} \quad &\textstyle\sum_{n=1}^{N} d_n \ge k, \quad d_n \ge 0 \end{aligned}$$

In view of these results, it would be plausible to guess that, for any positive integer n, $d_n(x_n) = x_n/n$ and $f_n(x_n) = x_n^2/n$. To show this, we proceed by induction:

$$f_{n+1}(x_{n+1}) = \min_{0 \le d_{n+1} \le x_{n+1}} [d_{n+1}^2 + (x_{n+1} - d_{n+1})^2/n] \tag{12}$$

Setting the partial derivative equal to zero,

$$d_{n+1} - (x_{n+1} - d_{n+1})/n = 0 \tag{13}$$

which is satisfied only by $d_{n+1} = x_{n+1}/(n + 1)$, resulting in $f_{n+1}(x_{n+1}) = x_{n+1}^2/(n + 1)$. We have proved that the minimum of the sum of squares of N variables whose sum is equal to or greater than a constant k is k^2/N.

STOCHASTIC MULTISTAGE DECISION PROCESS

One of the beauties of the dynamic programming method is that stochastic multistage decision processes are often treated similarly to the way we deal with deterministic processes (7). The basic assumptions are not changed significantly, beyond the introduction of randomness. In all multistage processes considered so far, the consequences of any particular decision that might be made were assumed to be known explicitly. However, in many cases there are decision processes in which we do not explicitly know the results of our actions until after we have taken them. A very simple example illustrates this. Suppose that you are gambling against a generous opponent who describes a game with a fair die as follows: You can roll the die up to three times. When you have seen the value on the die after the first roll, x_1, you decide whether to roll it again or not. If you decide to stop, then you are paid x_1 dollars. Similarly, if you roll the die a second time, scoring x_2, you are paid x_2 dollars if you stop. However, if you roll the die for the third time, scoring x_3, you receive $x_3 - 3$. You want to play in the best way possible, which is interpreted as an objective of maximizing your "expected" income from the game.

After the first roll (and the second if you continue), you are faced with a decision: stop or continue. This decision corresponds to a stage in a sequential process, whose states are defined in terms of the score in front of you when you make a decision. If you continue, the state which you reach at the next stage depends on the roll of the die, which is random (all six states have the same probability of occurring). In this case, one can approach the problem by replacing the return values corresponding to each *event* (a probabilistic consequence of a decision) by its probability times the corresponding deterministic value, and progressing recursively, similar to the case of the standard dynamic programming.

The previous examples of multistage decision processes, incorporating uncertainty in the behavior of the state variable and/or the criterion function, are *random decision processes* or, as they are more commonly called, *stochastic decision processes.* The procedure of solving such problems is further explained in the following example (8).

STOCHASTIC SHORTEST PATH

A map of the city, together with the costs of various arcs, is shown in Fig. 2. We imagine that we have been hired as a consultant to a forgetful traveler who wishes to go from A to B at minimum cost. If we instruct the traveler to go diagonally up, the traveler remembers our advice and does so with probability $\frac{3}{4}$. With probability $\frac{1}{4}$, the traveler does the opposite (takes the downward arc). Likewise, if our instruction is to move diagonally downward, the traveler complies with probability $\frac{3}{4}$, but moves upward with probability $\frac{1}{4}$. The traveler behaves this way at each vertex. Consequently, no matter

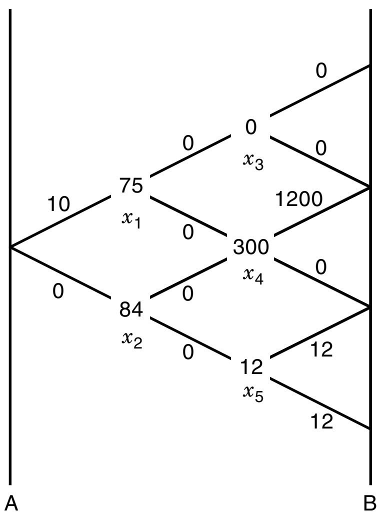

Figure 2. The possible paths from city A to B with different probabilities and costs.

what our instructions are, we cannot be sure of the path our employer will follow, but our advice will certainly determine the probabilities of various results. We wish to minimize the expected cost of the trip, assuring us that if the traveler repeats the journey a great many times, encountering different costs on different trips, the average cost is minimized.

To determine the best path, we consider all eight possible sequences of three decisions each and choose the one with the minimum expected cost. Note that the decision sequence D, U, D (Downward, Upward, Downward) that optimizes the deterministic version of this problem has probability $\frac{27}{64}$ of actually yielding the path consisting of a D, U, D transition. This path has cost 0. There is a probability of $\frac{9}{64}$ of an U, U, D path of cost 10, and a probability of $\frac{9}{64}$ of a D, U, U path of cost 1200, and soon. Multiplying each of the eight appropriate costs by their respective probabilities and adding, we obtain an expected cost E_{DUD} given by

$$E_{DUD} = \frac{27}{64} \times 0 + \frac{9}{64} \times (10 + 12 + 1200) + \frac{3}{64} \times (12 + 10 + 10) + \frac{1}{64} \times 1210 = 192\frac{1}{4} \qquad (14)$$

where the first, second, third, and fourth terms correspond to obeying all three instructions, obeying two of them (occurring in three ways), obeying one of them (again occurring in three ways), and obeying none of them.

To find the sequence of decisions which results in the optimum expected value of the cost, we proceed as follows: First we assign a cost to each branch equal to the probability of taking that branch (given a specific decision) times the value of the deterministic cost of the branch. In this case, if we are at state x_3, the expected cost is equal to zero for both decisions. If we are at state x_4, we instruct the traveler to do downward and the expected value of the corresponding cost is equal to: $1200 \times (\frac{1}{4}) + 0 \times (\frac{3}{4}) = 300$. Similarly, if we are at state x_5, the expected cost is equal to 12 for both decisions. Proceeding one stage backward, we conclude that the optimal decision at states x_1, x_2 is to move upward, downward, respectively. These decisions result in an average cost of $300 \times (\frac{1}{4}) + 0 \times (\frac{3}{4}) = 75$ and $300 \times (\frac{1}{4}) + 12 \times (\frac{3}{4}) = 84$, for the states x_1 and x_2, respectively. Finally, the best decision at point A is to move downward resulting in an average cost of $(75 + 10) \times (\frac{1}{4}) + 84 \times (\frac{3}{4}) = 84.25$.

BIBLIOGRAPHY

1. R. E. Bellman, *Dynamic Programming,* Princeton, NJ: Princeton University Press, 1957.
2. D. P. Bertsekas, *Dynamic Programming and Optimal Control,* Belmont, MA: Athena Scientific, 1995.
3. D. P. Bertsekas and J. Tsitsiklis, *Neuro-Dynamic Programming,* Belmont, MA: Athena Scientific, 1996.
4. A. J. Viterbi, Error bounds for convolutional codes and an asymptotically optimum decoding algorithm, *IEEE Trans. Inf. Theory,* **IT-13**: 260–269, 1967.
5. D. K. Smith, *Dynamic Programming,* New York: Ellis Horwood Limited, 1991.
6. G. L. Nemhauser, *Dynamic Programming,* New York: Wiley, 1967.
7. S. M. Ross, *"Introduction to Stochastic Dynamic Programming,"* New York: Academic Press, 1983.
8. S. E. Dreyfus and A. M. Law, *The Art and Theory of Dynamic Programming,* New York: Academic Press, 1977.

S. NIKNESHAN
A. K. KHANDANI
University of Waterloo

DYNAMIC RECONFIGURATION. See INTEGRATED SOFTWARE.

DYNAMICS OF MAGNETIC PARTICLES. See MAGNETIC PARTICLES.

DYNAMICS, ROBOT. See ROBOT DYNAMICS.