V. Dynamic Programming and Multistage Optimization

1. Optimization problems consist in selecting from among the feasible alternatives one which is economically optimal. A problem of this nature is solved by formulating a mathematical model of the problem, typically a maximization model in which a preference function is to be maximized subject to a number of side conditions, and applying a method of solution tailored to the particular kind of problem. The variables of the model, interdependent through the side relations, are determined *simultaneously* in the solution.

Consider, for example, the linear programming problem

$$\begin{aligned} f = 8x_1 + 10x_2 &= \max \\ 4x_1 + \ 2x_2 &\leqslant 12 \\ x_1, x_2 &\geqslant 0, \end{aligned} \tag{1}$$

to be interpreted as a problem of optimal capacity utilization. x_1 and x_2 are quantities produced per period of two commodities which require 4 and 2 machine hours per unit, and the right-hand side of the side condition is maximum available machine time per period; the coefficients in the preference function are unit profits. Solving by the simplex method (or, what is simpler in such a trivial case, geometrically or by numerical inspection) we get

$$x_1 = 0, \quad x_2 = 6; \quad f = 60. \tag{2}$$

The optimal values of the two decision variables are found simultaneously in the solution procedure.

2. An alternative approach is to determine the variables *one at a time* (sequentially), *decomposing* the problem into a series of *stages* each corresponding to a subproblem in only one variable, and solving the two single-variable subproblems separately instead of solving the two-variable problem (1). This is the basic idea underlying *dynamic programming* (DP).

The decomposition of the problem (1) can be illustrated as shown in Fig. 8.

S. Danø, *Nonlinear and Dynamic Programming*
© Springer-Verlag/Wien 1975

Let us assume for convenience that commodity no. 1 is produced "first" (stage 1). We might as well have started with the second commodity; the order in which they are arranged is purely formal in a case like this where the decomposition into stages does not reflect a sequence in time.

Now, for the production of the first commodity, 12 units of the capacity factor (machine hours) are available as shown in the flow diagram. If x_1 units are produced, $12 - 4x_1$ machine hours are available as input for the second stage. After producing x_2 units of commodity

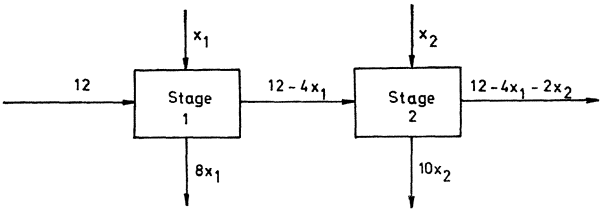

Fig. 8

no. 2, we are left with $12 - 4x_1 - 2x_2$ machine hours, corresponding to the slack variable in (1) which represents unutilized capacity. The two stages contribute $8x_1$ and $10x_2$ respectively to total profit.

We can now solve the problem *backwards*, treating x_1 as a *parameter* and optimizing *stage 2* with respect to the variable x_2. For parametric $x_1 = \bar{x}_1$ the maximal capacity left to stage 2 is $12 - 4\bar{x}_1$ machine hours so that the (parametric) subproblem of stage 2 is

$$\begin{aligned} f_1 = 10x_2 &= \max_{x_2} \\ 2x_2 &\leqslant 12 - 4\bar{x}_1 \\ x_2 &\geqslant 0, \end{aligned} \tag{3}$$

which is a linear programming problem like the total problem (1), only it is a single-variable problem. The solution is obviously

$$x_2 = 6 - 2x_1; \quad f_1^{\max} = 60 - 20\bar{x}_1 \tag{4}$$

where $\bar{x}_1$ is a parameter.

Next we optimize *stage 1* with respect to its decision variable, x_1. The capacity available is 12 machine hours. Production of x_1 units contributes $8x_1$ to total profit, but against this we have to consider that machine hours left over and used by stage 2 also affect total profit,

contributing $f_1^{\max} = 60 - 20x_1$ which also depends on x_1. The optimization problem of stage 1 therefore becomes

$$\begin{aligned} f_2 = 8x_1 + f_1^{\max} = 60 - 12x_1 &= \max_{x_1} \\ 4x_1 &\leqslant 12 \\ x_1 &\geqslant 0, \end{aligned} \tag{5}$$

which is also a linear problem. Because f_2—which expresses profit contributed by the first stage plus the (parametric) maximum profit earned by the second—is a decreasing function of x_1, the solution obviously is

$$x_1 = 0;\ f_2^{\max} = 60. \tag{6}$$

Having thus found the optimal value of x_1, which is also a parameter in the solution for x_2, we substitute it into (4) to get

$$x_2 = 6 - 2x_1 = 6. \tag{7}$$

The solutions which we have found for the two variables are seen to agree with (2), and total profit $f = 60$ is seen to be equal to $f_2^{\max}$—as it should be since $f_2^{\max}$ was calculated as the total of stage contributions to profit. What we did in solving (5) was to maximize the profit of stage 1—a function of x_1—plus the maximum profit earned by stage 2 for any given value of x_1.

In this way we have solved an optimization problem in two variables by transforming it into a series, or sequence, of two single-variable problems. This is an example of *dynamic programming*. The *subproblems* corresponding to the individual stages are of the *same type as the total problem* (1)—in the present case, a linear programming problem—and they are *solved by the same method* as that applied in the simultaneous solving of the total problem (e.g. the simplex method). In other words, "dynamic programming" does not refer to a particular class of optimization problems (e.g. linear programming problems) or to a specific method of solution (like the simplex method); rather, it indicates a general procedure for *decomposing* a problem into a series of subproblems involving fewer variables, and combining their solutions to get the solution of the original problem.

3. When an optimization problem is formulated as a multistage problem to be solved by dynamic programming, it is convenient to introduce *state variables* y_n associated with the individual stages (numbered $n = 1, 2, \ldots, N$). The production process of Fig. 8 may be thought of as starting in a (given) *initial state* where $y_0 = 12$ machine hours are available; this is the input state of stage 1. Producing x_1 units of the first commodity, each of which requires 4 machine hours,

changes the state of the system: available capacity is reduced by $4x_1$ machine hours so that the *output state* of stage 1—which is also the *input state* to stage 2—becomes $y_1 = y_0 - 4x_1 = 12 - 4x_1$. Producing x_2 units of the second product, available capacity is further reduced to $y_2 = y_1 - 2x_2 = y_0 - 4x_1 - 2x_2$, which in this case represents the *final state*. ($y_2 \geqslant 0$ by definition but otherwise unknown.)

Thus, the input state of stage no. n, y_{n-1}, is transformed into an output state y_n, the change being brought about by the *decision variable*

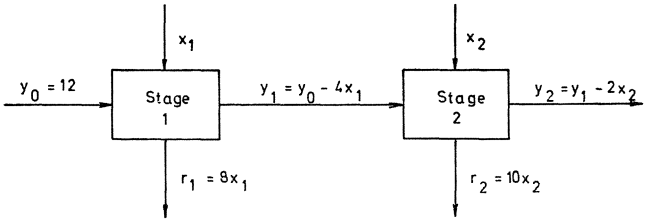

Fig. 9

of the stage, x_n. The successive changes of the state of the system can formally be described by *transformation equations* of the form

$$y_n = t_n(y_{n-1}, x_n) \quad (n = 1, 2, \ldots, N).$$

In the example they have the form

$$\begin{aligned} y_1 &= y_0 - 4x_1 \\ y_2 &= y_1 - 2x_2; \end{aligned} \tag{8}$$

together with the nonnegativity requirements $x_1, x_2, y_1, y_2 \geqslant 0$ they are equivalent to the restrictions of the original problem (1), y_0 being $= 12$ and y_2 representing the slack variable.

The *stage returns*, i.e., the contributions of the individual stages to the preference function f, will in the general case depend on the input state and the decision variable:

$$r_n = r_n(y_{n-1}, x_n) \quad (n = 1, 2, \ldots, N);$$

in the present example these *return functions* are of the simple form

$$\begin{aligned} r_1 &= 8x_1 \\ r_2 &= 10x_2. \end{aligned} \tag{9}$$

Introducing these symbols into Fig. 8, the flow diagram of the two-stage problem has the form of Fig. 9. The backward solution now proceeds as follows.

At the *first* stage of the calculations—corresponding to the last stage in the production system, $n = N = 2$—the input state y_1 is

considered as a parameter, "inherited" from the previous stage of the system. The stage is optimized by maximizing its *decision function*, f_1—here equal to the stage return, $r_2(x_2)$—subject to the parametric capacity restriction $2x_2 \leqslant y_1$[1]:

$$\begin{aligned} f_1 = r_2(x_2) &= 10x_2 = \max_{x_2} \\ 2x_2 &\leqslant y_1 \\ x_2 &\geqslant 0. \end{aligned} \tag{10}$$

The solution to this parametric single-variable LP problem is

$$\begin{aligned} x_2(y_1) &= 0.5y_1 \\ F_1(y_1) &= 5y_1 \end{aligned} \tag{11}$$

where F_1 denotes the maximum value of the stage decision function, $F_1 = f_1^{\max}$.

At the *second* stage of the computations (production stage 1) we maximize the decision function $f_2 = r_1(x_1) + F_1(y_1)$ subject to the capacity limitation $4x_1 \leqslant y_0$ (i.e., $y_1 = y_0 - 4x_1 \geqslant 0$); substituting the stage transformation $y_1 = y_0 - 4x_1$, f_2 becomes a function of x_1 and y_0 so that we have the LP problem

$$\begin{aligned} f_2 = r_1(x_1) + F_1(y_1) &= 8x_1 + 5y_1 \\ = 8x_1 + 5(y_0 - 4x_1) &= 5y_0 - 12x_1 = \max_{x_1} \\ 4x_1 &\leqslant y_0 \\ x_1 &\geqslant 0. \end{aligned} \tag{12}$$

The solution is

$$\begin{aligned} x_1(y_0) &= 0 \\ F_2(y_0) &= 5y_0 \end{aligned} \tag{13}$$

when $F_2 = f_2^{\max}$.

The solution to the complete problem (1)—called *the optimal policy*—can now be determined by solving the *recursive equation system* formed by the *parametric optimum solutions* (11) and (13) and the *transformation equations* (8), starting from the initial state $y_0 = 12$:

Transformation equations $y_n = t_n(y_{n-1}, x_n)$	Parametric optimal solutions $x_n = x_n(y_{n-1})$	Maximum of decision function
$y_0 = 12$		
$y_1 = y_0 - 4x_1 = 12$	$x_1(y_0) = 0$	$F_2(y_0) = 5y_0 = 60$
$y_2 = y_1 - 2x_2 = 0$	$x_2(y_1) = 0.5y_1 = 6$	$F_1(y_1) = 5y_1 = 60$

[1] This restriction is equivalent to $y_2 = y_1 - 2x_2 \geq 0$, that is, the output state—as represented by the slack variable—must not be negative.

where the direction of the computations is the opposite of that followed above in the optimization of stages. The optimal policy emerges as $x_1 = 0$, $x_2 = 6$ and total profit is $f = F_2 = 60$. If y_0 had been a parameter, the solution—now parametric—would have been $x_1 = 0$, $x_2 = 0.5y_0$, $f = 5y_0$[1].

The decomposition by which we solved problem (1) can be described as follows. Replacing the side conditions by the equivalent formulation

$$y_1 = y_0 - 4x_1, \; y_1 \geqslant 0$$
$$y_2 = y_1 - 2x_1, \; y_2 \geqslant 0,$$

this together with the nonnegativity requirements implies

$$0 \leqslant x_1 \leqslant \frac{1}{4} y_0 \tag{14}$$

$$0 \leqslant x_2 \leqslant \frac{1}{2} y_1 . \tag{15}$$

Then we can write (1) in the form

$$f^{\max} = \max_{x_1, x_2} (8x_1 + 10x_2) = \max_{x_1} (8x_1 + \max_{x_2} 10x_2)$$

subject to (14)–(15); clearly this maximization problem can be decomposed into two single-variable problems, corresponding to (10) and (12):

$$F_1(y_1) = \max_{x_2} \; 10x_2 \left(0 \leqslant x_2 \leqslant \frac{1}{2} y_1\right)$$

$$F_2(y_0) = \max_{x_1} \; (8x_1 + F_1(y_1))$$
$$= \max_{x_1} \{8x_1 + F_1(y_0 - 4x_1)\} \left(0 \leqslant x_1 \leqslant \frac{1}{4} y_0\right),$$

where $f^{\max} = F_2(y_0)$. In a more general formulation the decomposition of a two-stage problem

$$f^{\max} = \max_{x_1, x_2} [r_1(y_0, x_1) + r_2(y_1, x_2)]$$

can be expressed in the *recursion equations*

$$F_1(y_1) = \max_{x_2} [r_2(y_1, x_2)] \tag{16}$$

$$F_2(y_0) = \max_{x_1} [r_1(y_0, x_1) + F_1\{t_1(y_0, x_1)\}]. \tag{17}$$

---

[1] For $f^{\max} = 5y_0$, where y_0 is the total capacity of the system, we have $df/dy_0 = 5$. **Exercise 17:** Interpret this result.

4. This procedure of solving a dynamic programming problem by *backward recursion* can be generalized to any number of variables. A flow diagram for an N-stage system is shown in Fig. 10.

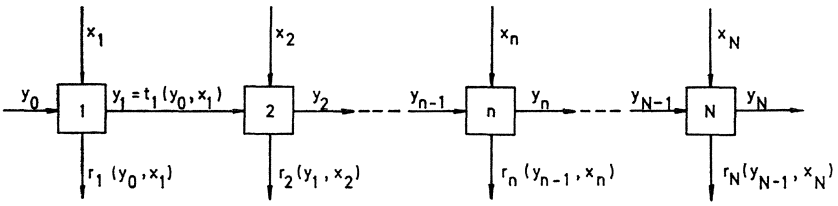

Fig. 10

The decision functions of stages $N, N-1, \ldots, 2, 1$ are respectively

$$f_1 = r_N(y_{N-1}, x_N)$$

$$f_2 = r_{N-1}(y_{N-2}, x_{N-1}) + F_1(y_{N-1}) \text{ where } y_{N-1} = t_{N-1}(y_{N-2}, x_{N-1}) \quad (18)$$

$$\ldots$$

$$f_N = r_1(y_0, x_1) + F_{N-1}(y_1) \qquad \text{where } y_1 = t_1(y_0, x_1),$$

F_j being the maximum of f_j $(j = 1, 2, \ldots, N)$. Maximizing the decision function of each stage with respect to its decision variable, treating the input state as a parameter, we get the parametric stage solutions

$$x_n = x_n(y_{n-1}) \quad (n = N, N-1, \ldots, 1) \qquad (19)$$

which can be "sewn together" by means of the transformation equations so that we get the parameters determined. y_0 now determines x_1 and together they determine y_1; this gives x_2, which with y_1 determines y_2; and so forth as illustrated by Fig. 11.

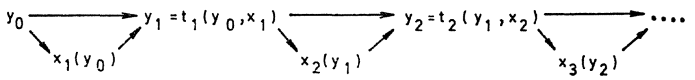

Fig. 11

It follows from (18) that the maximum of f_N represents the accumulated value of the stage returns, i.e., $F_N = f^{\max}$.

5. If the variables of a DP problem are allowed to take *discrete* values only, or if the stage returns and/or the transformation functions

are given in *tabular form* for discrete values of the variables, the problem will have to be solved by *tabular computations*.

For example, the decision variables may be required to have *integral* values ($x_n = 0, 1, 2, \ldots$) because the interpretation of the problem is such that fractional values would be meaningless. This is so in problem (1): strictly speaking it is impossible to produce a fractional number of units of a commodity, e.g. $x_1 = 2.6$. Thus, although in this case the analytical shape of the *return functions* is known, r_1 and r_2 are defined only for integral values of x_1 ($\leqslant 3$) and x_2 ($\leqslant 6$):

x_1	0	1	2	3
$r_1 (= 8x_1)$	0	8	16	24

x_2	0	1	2	3	4	5	6
$r_2 (= 10x_2)$	0	10	20	30	40	50	60

The *transformation functions* in tabular form are as follows:

	$y_1 (= y_0 - 4x_1)$			
$y_0 \backslash x_1$	0	1	2	3
12	12	8	4	0

	$y_2 (= y_1 - 2x_2)$						
$y_1 \backslash x_2$	0	1	2	3	4	5	6
0	0						
4	4	2	0				
8	8	6	4	2	0		
12	12	10	8	6	4	2	0

where y_1 is confined to the values 0, 4, 8, and 12 resulting from the first table. The blank cells in the last table correspond to combinations of values of y_1 and x_2 which are not feasible because they would imply a negative value of y_2 (cf. the sign restriction $y_2 = y_1 - 2x_2 \geqslant 0$).

The solution procedure, using backward recursion, now proceeds as follows. For the *last stage* ($n = N = 2$) we have:

Stage 2	$f_1 = r_2(x_2)$ $(= 10x_2)$									
$y_1 \diagdown x_2$	0	1	2	3	4	5	6	$F_1(y_1)$	$x_2(y_1)$	$y_2(y_1)$
0	**0**							0	0	0
4	0	10	**20**					20	2	0
8	0	10	20	30	**40**			40	4	0
12	0	10	20	30	40	50	**60**	60	*6*	*0*

where the maximal value of the decision function f_1 for each of the possible input states (values of y_1) is shown in bold-faced type[1]. These parametric optima and the optimal values of the decision variable are listed in the F_1 and x_2 columns to the right. The last column gives the resulting values of the output state y_2, computed from the transformation function (or table).

For *stage 1* we have the decision function $f_2 = r_1(x_1) + F_1(y_1)$ where the transformation gives y_1 for each value of x_1; for example, $x_1 = 2$ implies $y_1 = y_0 - 4x_1 = 4$, and the preceding table then gives $F_1(y_1) = 20$. The stage computations are done in the following table:

Stage 1	$f_2 = r_1(x_1) + F_1(y_1)$ $\{= 8x_1 + F_1(y_1)\}$					
$y_0 \diagdown x_1$	0	1	2	3	$F_2(y_0)$	$x_1(y_0)$	$y_1(y_0)$
12	**0 + 60**	8 + 40	16 + 20	24 + 0	60	*0*	*12*

These tables correspond to (10)–(11) and (12)–(13) respectively, and the optimal solution can be found similarly, starting with the last table. $y_0 = 12$ gives $x_1 = 0$ (the optimal stage solution) which leads to $y_1 = 12$ (by the transformation equation). Proceeding to the first table, $y_1 = 12$ (as just determined) gives the stage optimum $x_2 = 6$ and the output state $y_2 = 0$. These values are indicated in italics[2]. The maximal total return is $F_2(y_0) = 60$.

6. It will now be clear why it was expedient to solve the decomposed version of problem (1) *backwards*, starting with the optimization of the last stage. The procedure led to a recursive system which has the *initial state* y_0 as its starting point—as shown in Fig. 11—and it was y_0 that was given ($y_0 = 12$). This suggests that problems in which the

[1] For solution by hand these values may be put in circles.

[2] When solving such a problem by hand it is practical to underline these values in the tables, in the order in which they are determined (y_0, x_1, . . .).

final state y_N is given may be solved in the opposite direction, proceeding *forwards* from the first stage.

To show how this is done, let us redefine the state variables y_n in the example so that the final state y_2 now represents total accumulated "use" of capacity, including idle capacity; the latter is put first as the input state y_0 of the first stage of the production system[1]. Then y_1 represents the accumulated "use" of capacity, including capacity not utilized, after the first commodity has been produced. This leads to the transformations

$$\begin{aligned} y_1 &= y_0 + 4x_1 \\ y_2 &= y_1 + 2x_2 \end{aligned} \tag{20}$$

where $y_0 \geqslant 0$ and $y_2 = 12$. The stage return functions are the same as above, (9).

This dynamic programming problem is another decomposed version of problem (1). To solve it by *forward recursion* we reverse the direction

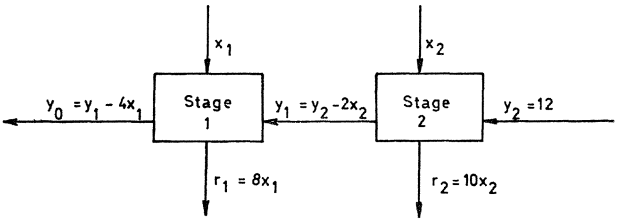

Fig. 12

of the system as shown in Fig. 12, where y_n is now to be formally treated as the input state of stage n ($n = 1, 2$) whereas the y_{n-1} become output states; we therefore write the transformation functions (20) in the inverse form

$$\begin{aligned} y_0 &= y_1 - 4x_1 \\ y_1 &= y_2 - 2x_2 \end{aligned} \tag{21}$$

(more generally,

$$y_{n-1} = t_n^* (y_n, x_n)$$

where t_n^* is the inverse transformation equation of stage n).

The procedure starts with the optimization of *stage 1*. The decision function $f_1 = r_1 (x_1)$ is to be maximized subject to $y_0 = y_1 - 4x_1 \geqslant 0$,

---

[1] y_0 may be thought of as the output of a fictitious (slack) stage which "absorbs" part of the total capacity.

$x_1 \geqslant 0$, where y_1—now the input state of the stage—is a parameter:

$$\begin{aligned} f_1 = r_1(x_1) = 8x_1 &= \max_{x_1} \\ 4x_1 &\leqslant y_1 \\ x_1 &\geqslant 0; \end{aligned} \tag{22}$$

the solution depends on the parameter,

$$\begin{aligned} x_1(y_1) &= 0.25y_1 \\ F_1(y_1) &= 2y_1 \end{aligned} \tag{23}$$

where $F_1 = f_1^{\max}$. At the *second stage* we have the decision function $f_2 = r_2(x_2) + F_1(y_1)$ where $y_1 = y_2 - 2x_2 \geqslant 0$, $x_2 \geqslant 0$ so that the stage optimization problem becomes

$$\begin{aligned} f_2 = 10x_2 + 2y_1 = 6x_2 + 2y_2 &= \max_{x_2} \\ 2x_2 &\leqslant y_2 \\ x_2 &\geqslant 0. \end{aligned} \tag{24}$$

The solution is

$$\begin{aligned} x_2(y_2) &= 0.5y_2 \\ F_2(y_2) &= 5y_2. \end{aligned} \tag{25}$$

Working backwards from y_2 $(= 12)$ through the recursive system (21), (23), (25) we obtain the optimal solution:

$$\begin{aligned} y_2 &= 12 & x_2(y_2) &= 0.5y_2 = 6 \\ y_1 &= y_2 - 2x_2 = 0 & x_1(y_1) &= 0.25y_1 = 0, \\ y_0 &= y_1 - 4x_1 = 0, \end{aligned}$$

and $f^{\max} = F_2(y_2) = 5y_2 = 60$.

In the general case of N stages the decision functions are[1]

$$\begin{aligned} f_1 &= r_1(y_0, x_1) && \text{where } y_0 = t_1^*(y_1, x_1) \\ f_2 &= r_2(y_1, x_2) + F_1(y_1) && \text{where } y_1 = t_2^*(y_2, x_2) \\ &\cdots \\ f_N &= r_N(y_{N-1}, x_N) + F_{N-1}(y_{N-1}) && \text{where } y_{N-1} = t_N^*(y_N, x_N). \end{aligned}$$

The parametric stage solutions

$$x_n = x_n(y_n) \qquad (n = 1, 2, \ldots, N)$$

and the inverse transformation equations determine the optimal policy for given y_N as shown in Fig. 13.

[1] In the example above, r_n depends only on x_n.

It is often possible to solve by forward recursion when the initial state y_0 is given, or to apply backward recursion to DP problems with given final state y_N, using a slightly modified procedure[1]. However, when the order of the stages is arbitrary and the transformations can be inverted so that we are free to choose the direction, *backward recursion* is generally a more efficient procedure for *given* y_0, and *forward recursion* for *given* y_N (which amounts to the same thing, the only difference being the numbering of stages and variables)[2].

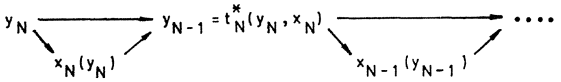

Fig. 13

7. In order for an optimization problem to be solvable by dynamic programming, the "*technical*" *structure* of the problem (as represented by the *restrictions*) must be such that it can be described by a series of successive changes of the state of the system, from the initial state y_0 to the final state y_N, each change being effected by a particular decision variable. The two-stage system of Fig. 9 above is an example of this; the transformation equations (8) together with x_1, y_1, x_2, $y_2 \geqslant 0$ and $y_0 = 12$ are an equivalent reformulation of the restrictions in problem (1), i.e., $4x_1 + 2x_2 \leqslant 12$ and $x_1, x_2 \geqslant 0$, so that the set of feasible solutions is preserved. In some cases the multistage structure represents a sequence in time—hence the name "dynamic" programming—where the stages correspond to actual processes and the direction indicates the order in which the transformations take place. This would be so in our example (1) if the two commodities were produced in separate processes and commodity no. 1 had to be made first. In many applications, however, the sequence of stages and the order in which the system passes through them are an artificial device, introduced in order to make the problem solvable by DP methods. In either case we can choose between forward and backward recursion if the direction is mathematically arbitrary.

Decomposition of a problem also requires that the *objective function* satisfies certain conditions. In general, the function to be maximized is some function of the stage returns r_n,

$$f = \varphi\{r_1(y_0, x_1), \ldots, r_N(y_{N-1}, x_N)\}. \tag{26}$$

---

[1] Using forward recursion for given y_0, there is no maximization at the first stage because the inverse transformation equation determines x_1 uniquely as a function of the parameter y_1 for given y_0. See Ch. X below.

[2] A survey of these recursion procedures is given in the Appendix (Ch. X).

It can be shown that two conditions on the function—separability and monotonicity[1]—together are sufficient for decomposition, i.e., for solution by means of a system of recursive equations. These conditions are automatically satisfied by a class of functions including the case of *additive returns*,

$$f = r_1(y_0, x_1) + \ldots + r_N(y_{N-1}, x_N).$$

In this case, as we have seen above, the objective function is obviously decomposable and the recursion equations have the form (16)–(17) for a two-stage problem, readily generalized to any number of stages [cf. (18)].

8. The recursion equations can be thought of as a mathematical expression of an intuitive principle known as the "*principle of optimality*"[2]. Consider the two-stage problem shown in Fig. 9, and let $(x_1, x_2) = (\bar{x}_1, \bar{x}_2)$ be the optimal policy. The first decision, $x_1 = \bar{x}_1$, changes the state of the system from the initial state $y_0 = 12$ to $y_1 = y_0 - 4\bar{x}_1 = \bar{y}_1$. The principle of optimality now says that the remaining decision, $x_2 = \bar{x}_2$, must represent an optimal policy with respect to the state $\bar{y}_1$, i.e., it must be an optimal solution to the remaining one-stage DP problem with the initial state $\bar{y}_1$. The proof is simple: if this were not so, $(x_1, x_2) = (\bar{x}_1, \bar{x}_2)$ could not be an optimal policy.

The backward recursive procedure, as expressed in (16)–(17), by which we solved the problem is based directly on this principle. Starting by optimizing the last stage ($n = 2$), we do not know its input stage y_1, but we do know that whatever it is—i.e., whatever the first decision is—x_2 must be optimal with regard to y_1. Hence we optimize $f_2 = r_2(y_1, x_2)$ for parametric y_1 as expressed in (16). Proceeding backwards to stage one, we optimize $f_2 = r_1(y_0, x_1) + F_1(y_1)$, i.e., the return of stage 1 plus the parametric optimal return of stage 2, where $y_1 = t_1(y_0, x_1)$.

The principle of optimality in its general form states that any part of an optimal policy must be optimal; specifically, the decisions remaining after stage no. n (i.e., $x_{n+1}, \ldots, x_N$) must constitute an optimal policy for the series of stages $n + 1, \ldots, N$ with regard to the state y_n resulting from the first n decisions.

9. The multistage structures dealt with above (cf. Fig. 10) are *serial systems*, i. e., ordered sequences of stages where the output state of stage no. n is the input state to stage no. $n + 1$. Moreover, they are special in that there is *only one decision variable* x_n and *only one (output) state variable* y_n for each stage.

---

[1] See G. L. Nemhauser (1966), pp. 34—39.

[2] The principle of optimality was first stated by R. Bellman, the originator of dynamic programming. See R. E. Bellman (1957), p. 83.

As an obvious generalization, the x_n and the y_n may be vectors so that there are *several decision and state variables per stage*[1]. Cases of this kind are treated in Chapter VII below.

Nonserial multistage systems—important in the chemical industries—represent another generalization, characterized by branches or loops in the flow diagram[2].

A third extension of the N-stage serial multistage structure is an *infinite-stage system* where N tends to infinity. This case, relevant to some applications, is treated below in Chapter VIII.

10. The advantage of dynamic programming as a procedure for solving optimization problems is the simplification obtained by decomposition. It is often simpler and easier to solve a series of single-variable stage optimization problems, and in some cases this is the only possible procedure because "simultaneous" solution is mathematically or computationally difficult or downright impossible. Certain classes of optimization problems, however, such as linear programming problems are more efficiently solved by special algorithms without decomposition, e.g. the simplex method.

Dynamic programming has been particularly successful in its discrete version. Tabular computations are well suited for computer solution and can be used to handle problems involving irregular functions, and may be the only practicable way of solving a problem in which the variables are required to be integers. As a rough illustration of the computational advantages of DP, consider an optimization problem in N decision variables, each of which can assume m alternative discrete values (e.g. $0, 1, 2, \ldots, m-1$). If, for lack of other methods of solution, we had to solve the problem by total enumeration, we would have to examine each of m^N alternative solutions for feasibility and optimality, whereas in a dynamic programming procedure—assuming that decomposition is possible—the number of alternatives to be enumerated would be reduced to mN, namely m for each of the N stages. Thus, roughly speaking, the computational labour increases exponentially with the number of decision variables in the case of total enumeration, but only proportionately if the problem is decomposed. For large values of m and N the computational advantages become enormous; for example, in a problem with 20 variables each of which can assume integral values from 0 to 9 we have $m^N = 10^{20}$—an astronomic number—as against $mN = 200$ possible solutions if the problem is reformulated as a 20-stage DP problem.

---

[1] In such cases the direction may not be arbitrary because inversion of the transformation equations may be difficult.

[2] See G. L. Nemhauser (1966), Ch. VI.