Optimization of Multistage Cyclic and Branching Systems by Serial Procedures

RUTHERFORD ARIS
University of Minnesota, Minneapolis, Minnesota
GEORGE L. NEMHAUSER
The Johns Hopkins University, Baltimore, Maryland
DOUGLASS J. WILDE
Stanford University, Stanford, California

The calculus of variations, dynamic programing, and Pontryagin's maximum principle all are methods for optimizing serial decision processes, the kind associated with multistage operations having no recycle or bypass. Addition of recycle to a serial process makes it cyclic, and branching structures can be built up by connecting serial ones. The concept of cut state makes possible the decomposition of cyclic and branched systems into serial ones solvable by serial procedures. Under favorable circumstances cut states can be directed to their optimum values by efficient optimum seeking methods, which is not possible for ordinary state variables. These methods are worthwhile only for loops having at least three stages, and the treatment of converging branches is more complicated than for diverging ones. Visualization of the various system structures is aided by functional diagrams. Definitions and nomenclature are developed for continued research on optimization of macrosystems by serial techniques.

This article presents techniques for optimizing large macrosystems composed of relatively simple units interconnected by complicated flows of material, energy, and information. The simpler microstructures to be discussed are branching processes, both converging and diverging, and cyclic processes of the kind generated by the recycling, feedback, and parallel operations so widespread in the chemical industry. The objective at all times is to minimize the effects of a problem's dimensionality, which if not watched with the greatest care may easily grow to prohibitive proportions. To attain this goal it is necessary first to clarify and make precise the underlying concepts useful for optimizing composite systems, large or small.

For a process with a simple serial structure, the stages or units being linked in a straight line, the algorithms of Bellman's dynamic programing have often had striking success (*3*, *5*). The methods of dynamic programing are of course equivalent in a broad sense to the discrete version (*6*, *7*, *11*) of Pontryagin's maximum principle (*14*) as well as to suitable adaptations of the calculus of variations (*9*). Unfortunately these serial optimization techniques cannot be applied without considerable care if any feedback or cross linkage from one stage to another destroys the serial structure. Fan and Wang (*8*) have recently shown how to adapt the maximum principle to cyclic structures, and Jackson (*10*) has extended variational methods to both cyclic and branching systems. Although the methods of this article have the dynamic programing orientation of Mitten and Nemhauser (*12*) and Aris and Yesberg (*1*, *2*), they can also be used in combination with the other serial methods. In all these cases structural complexity has induced formidable increases in dimensionality. It will be shown that dimensionality need not bring a proportionate, and disastrous, increase in the number of optimization computations required.

The key notion is to formulate the problem so that optimum-seeking techniques (*16*) can be applied to the additional state variables entering the picture owing to structural complexity. Such techniques, which have always been used on descision variables, cannot ordinarily be employed on state variables, which usually must be examined exhaustively. To see the importance of this distinction, suppose one wishes the particular value from among a thousand possibilities at which a certain variable gives the maximum of some objective function depending on it. If the variable is a state variable, all thousand cases must be examined, but for a decision variable only fifteen evaluations are needed when the Fibonacci search technique is applicable (*12*, *16*). This is why knowing when to treat a state as a decision is of the utmost importance, not only to reduce computer time but also to conserve computer storage as well.

A major goal of this paper is to define states and decisions precisely. There are different kinds of states, depending on the roles they play in the overall optimization plan. This article does not discuss the important problem of assigning these roles properly, but such problems are not really difficult for the simple structures studied here. One problem studied in this article is the choice of the order in which to carry out the computations, for reversals of computation direction, as well as interchanges of states and decisions, require the definition of inversions.

The particular structures discussed are the branched and the cyclic systems. Visualization of these structures is aided by the introduction of functional diagrams which have evolved from similar graphic techniques used to analyze serial systems (*14*). It is shown how each cyclic and branching structure may be decomposed into interconnected serial problems and then solved by applying existing serial optimization techniques. Of particular interest to chemical engineers is the fact that these tactics should not be used on recycle systems having less than three decision stages in the loop. In such cases it is politic to ignore the structure entirely and use an optimum seeking technique. On the other hand, the decompositions described should prove useful for the multistage cycles and the multiloop systems abounding in the process industries. Also of interest is the fact that converging branches require different treatment than do diverging ones.

VARIABLES, TRANSFORMATIONS, AND STAGES

One striking characteristic of the structured optimization problems considered here is the large number of variables involved. To understand such problems and their efficient solution, it is helpful to classify the variables and the functions relating them into several groups. Consider a function P by which one can find the value of some single variable y when the values of some other k variables $x_1, \ldots, x_k$ (abbreviated as a single vector $\mathbf{x}$) are given:

$$y = P(\mathbf{x})$$

The function is said to transform the independent variables $\mathbf{x}$ into the dependent variable y; in control terminology y is the output of a transformation process P for which the components of $\mathbf{x}$ are the inputs. In fortunate circumstances the transformation process may be a simple formula, but in extreme cases it may be thought of as a complicated computer program.

For the problems under consideration certain of the outputs are added together to form a total return R. The ouputs contributing directly to R are called stage returns and written r_i, the index i being an integer identifying the stage. Thus

$$R = \Sigma r_i \tag{1}$$

The inputs upon which r_i depends are given the same index, as is the return function R_i accomplishing the transformation:

$$r_i = R_i\,(\mathbf{x}) \tag{2}$$

The total return may be composed from the stage returns by some operators other than addition without altering the results (4). For simplicity this article deals only with the addition operation.

All outputs which are not returns are called state variables and written $\tilde{s}_i$, where i is the index of the inputs generating the state. The transformation having the state $\tilde{s}_i$ as its output is given the name transition function and written T_i so that

$$\tilde{s}_i = T_i\,(x_i) \tag{3}$$

Many states act also as inputs; a state serving as an input to T_i is written s_i. All inputs which are not states are called *decision* variables and written d_i, the index i identifying the return and transition having d_i as an input. Thus Equations (2) and (3) can be written

$$r_i = R_i\,(d_i, s_i) \tag{4}$$

$$\tilde{s}_i = T_i\,(d_i, s_i) \tag{5}$$

The set of all transformations carrying the same index i is called the i^{th} *stage*, and the set of all stages is called the *system*. Combinations of Equations (1) and (4) give the total return R as a function of the decision and state variables:

$$R = \sum_{i=1}^{N} R_i\,(d_i, s_i) \tag{6}$$

Multiple decision or state variables may be distinguished from each other by adding a second subscript (d_{i1}, d_{i2}, d_{i3}, etc.). A stage can have several transition functions, given double subscripts where necessary to avoid confusion. However, each stage has only one return function, and the returns are always scalar. Each decision and state is for simplicity treated as a single variable here, although under circumstances discussed later the relations derived hold for vector variables as well.

SERIAL OPTIMIZATION

A certain structure is imposed on the system when a state variable is usually the output of some stage (say i) and the input to at least one other stage (say j). Then the state variable can be represented by more than one symbol, either $\tilde{s}_i$ or s_j for example. The identification of the several symbols with each other ($\tilde{s}_i \equiv s_j$) defines precisely how the stages i and j interact, and such a relation will be called an *incidence identity*. A system is specified completely by its stages and its incidence identities.

Suppose for example that there are N stages which can be numbered so that the incidence identities are

$$\tilde{s}_{i+1} \equiv s_i;\ i = 1, 2, \ldots, N-1 \tag{7}$$

It is enlightening to represent the stages schematically by appropriately numbered rectangles, with arrows leaving a stage rectangle standing for output states from the stage. Arrows pointing to a stage denote a stage input. States and their corresponding incidence identities are shown as arrows connecting the stages, as in Figure 1. Such a representation, known as a *functional diagram,* shows graphically the system structure in a manner which, unlike the incidence identities, does not really depend on how the numbers are assigned to the stages. One could in fact draw the functional diagram without numbering the stages at all. Figure 1 shows why such a configuration, having the output of one stage as the input to the next, is called a *serial* system.

Notice that the functional diagram is not the same as the process flow diagram. The former represents information flow, the latter material or energy flow.

For a serial system the $N - 1$ transition Equations (5) through (5*a*) can be used together with the incidence identities to eliminate state variables $s_1, \ldots, s_{N-1}$ from Equation (6) and give R as a function only of the decisions $d_1, \ldots, d_N$ and the initial state s_N. Thus the total return function may be written $R(d_1, \ldots, d_N, s_N)$, or more compactly as $R(d_i, s_N)$.

Strictly speaking s_N, not being the output of any stage, should be called a *decision* rather than a *state*. This use of the word state is a carry over from the existing literature of dynamic programing, which gives connotations to state not entirely consistent with the strict definition in this article. If one can choose freely among the many possible values of s_N, then s_N will be called a *choice* variable and written c_N instead of s_N. Although there is absolutely no difference between a choice and a decision as far as actual computations are concerned, both kinds of variables lending themselves to the application of optimum seeking methods, the distinction in notation will be preserved for analytical reasons to be made apparent later. When as in the section to follow optimum seeking methods are for some reason prohibited, the notation s_N will be resurrected.

If no side conditions relate the decisions d_i to the choice c_N, one can define the maximum total return R^* as

$$R^* = \max_{d_i, c_N} \{R(d_i, c_N)\} \tag{8}$$

The serial optimization problem is to find a set of optimal values $d_1^*, \ldots, d_N^*$, c_N^* for which the total return attains this maximum:

$$R(d_i^*, c_N^*) = R^* \tag{9}$$

This set of optimal values is called an *optimal policy.* There being N 1 independent variables whose values are to be given by the optimal policy, the problem is said to have $N + 1$ degrees of freedom, one for each choice and decision variable.

INITIAL VALUE PROBLEM

Often one must determine the maximum serial return for every possible value of the input to stage N. Since in this case optimum seeking methods cannot be used on this input variable, which must be examined exhaustively, it will be written s_N again and called the *initial state*. The special status of s_N is suggested by reading the total return $R(d_i, s_N)$ as the total return, given the initial state s_N.

The corresponding initial value maximum return function is $R^*(s_N)$, a function of s_N, defined as the function which for all values of $d_1, \ldots, d_N$ and for any particular value of s_N is such that

$$R^*(s_N) = \max_{d_i} \{R(d_i, s_N)\} \qquad (10)$$

The initial value serial optimization problem is to find a set of functions of s_N, namely $d_1^*(s_N), \ldots, d_N^*(s_N)$, for which $R(d_i, s_N)$ is maximum:

$$R(d_i^*(s_N), s_N) = R^*(s_N) \qquad (11)$$

The functions $d_i^*(s_N)$ form an initial value optimal policy, and since there are N functions of the single state variable s_N to be specified by the policy, the problem is said to be an N-decision, one-state, optimization problem. The number of decisions plus the number of states equals the number of degrees of freedom.

DECOMPOSITION BY DYNAMIC PROGRAMING

The sequential structure of a serial system can be exploited to transform the N-decision, one-state, initial value optimization problem into a set of N one-decision, one-state problems. This is accomplished by the procedure called *dynamic programing*, which will now be translated into the new notation.

Let S_n be the sum of the returns from stages 1 through n:

$$S_n \equiv \sum_{i=1}^{n} R_i(d_i, \bar{s}_i); \; n = 1, \ldots, N \qquad (12)$$

$$S_n = S_n(d_1, \ldots, d_n, s_n); \; n = 1, \ldots, N \qquad (13)$$

Let $f_n(s_n)$ be the set of values of S_n such that for any given value of s_n and for all values of $d_1, \ldots, d_n$

$$f_n(s_n) = \max_{d_1, \ldots, d_n} \{S_n(d_1, \ldots, d_n, s_n)\}; \; n = 1, \ldots, N \qquad (14)$$

From this definition it follows that

$$f_{n+1}(s_{n+1}) = \max_{d_{n+1}} \{R_{n+1}(d_{n+1}, s_{n+1}) + f_n(T_{n+1}(d_{n+1}, s_{n+1}))\}; \; n = 1, \ldots, N-1 \qquad (15)$$

for any given value of s_{n+1} and for all values of d_{n+1}. This recursion relation, which states the principle of optimality [3] in mathematical terms, shows that $f_{n+1}(s_{n+1})$ can be obtained from $f_n(s_n)$ by a one-decision, one-state optimization. To get the recursion started Equation (14) generates $f_1(s_1)$ by a one-decision (d_1), one-state (s_1) optimization. Finally, after N such partial optimizations, the N decision functions $d_1^*(s_1), \ldots, d_N^*(s_N)$ are obtained, together with the N stage initial value maximum return function $f_N(s_N)$ which by definitions (10) and (14) must be $R^*(s_N)$:

$$R^*(s_N) = f_N(s_N)$$

When one starts with $d_N^*(s_N)$, the recursive substitution of $d_i^*(s_N)$ into transition Equation (5) to beget the optimal input function $s_{i-1}(s_N)$, which is in turn put into $d^*_{i-1}(s_{i-1})$ to give $d^*_{i-1}(s_N)$, eventually generates the entire initial value optimal policy.

The partial optimizations of dynamic programing emphasize how different is the role of a decision variable from that of a state variable when the computations are done numerically. For every possible value of an input state, s_i for instance, one finds the value of the decision variable d_i maximizing the function $R_i(d_i, s_i) + f_{i-1}(d_i, s_i)$. Since d_i is a system input, unaffected by any stage outputs, none of its values are important except the optimal ones, which may be found fairly efficiently by optimum seeking methods. On the other hand since s_i is an output from other parts of the system, one cannot know which of its values might be optimal until the entire problem has been solved. For this reason an optimal decision $d_i^*(s_i)$ must be found for every value of s_i, which rules out using optimum seeking techniques on the state variables. Furthermore $d_i^*(s_i)$ must be saved for all i and every value of s_i in order to generate the optimal policy. This storage requirement can exceed the rapid access memory of large digital computers.

The states and decisions may in general be vectors whose components are single variables. Except for obvious changes in notation the partial optimization procedure is no different than before, although the number of computations certainly increases. But state dimensionality is more of a computational and storage burden than decision dimensionality.

In practice one rarely needs to know $R^*(s_N)$ for every possible value of s_N. Usually either s_N is specified to be a particular constant value k_N or else it can be selected at will and is therefore really a choice variable c_N amenable to optimum seeking techniques. In the former case one simply determines $R^*(k_n)$, a constant, and the corresponding optimal policy. The other situation, formulated in the previous section, can be solved by dynamic programing with one modification. Since stage N has a choice c_N and a decision d_N as inputs, the optimum return R^* is the result of a two-decision, no-state optimization:

$$R^* = \max_{d_N, c_N} \{R_N(d_N, c_N) + f_{N-1}(d_N, c_N)\} \qquad (17)$$

FINAL VALUE PROBLEM—STATE INVERSION

Consider the problem of obtaining the maximum return from a serial process as a function of $\bar{s}_1$, called the *final state* because it is the only state not a stage input. Suppose there exist N inverse transition functions $T_i(d_i, \bar{s}_i)$ which express the input s_i as a function of the decision and output:

$$s_i = T_i(d_i, \bar{s}_i); \; i = 1, \ldots, N \qquad (18)$$

Finding such a function for a stage is called *state inversion* because the roles of the input and output states are interchanged. The inversion transitions can be used to express the stage returns in terms of decision and output state:

$$\mathcal{R}_i(d_i, \bar{s}_i) = R_i(d_i, T_i(d_i, \bar{s}_i)); \; i = 1, \ldots, N \qquad (19)$$

A total return function $R(d_i, \bar{s}_1)$ depending only on the decisions and the final state $\bar{s}_1$ can be obtained by adding the inverted stage returns and using the inverse transitions and incidence identities to eliminate states s_1 through s_N. The corresponding final value maximum return function, given $\bar{s}_1$, is

$$R^*(\bar{s}_1) = \max_{d_1, \ldots, d_N} \{R(d_i, \bar{s}_1)\} \qquad (20)$$

The final value optimization problem is to find a set of functions $d_1^*(\bar{s}_1), \ldots, d_N^*(\bar{s}_1)$ for which

$$R(d_i^*(\tilde{s}_1), \tilde{s}_1) = R^*(\tilde{s}_1) \qquad (21)$$

The N functions $d_i^*(\tilde{s}_1)$ form a final value optimal policy, and like the initial value case, finding them is an N-decision, one-state problem. As before there are $N + 1$ degrees of freedom.

When every stage has been inverted, the final value problem can be put into the same notational form as the initial value problem by renumbering the stages in reverse. Then the final value problem can be decomposed into N one-decision, one-state problems by dynamic programing in the manner already described.

Certain complications arise in state inversion when the states are vectors, depending upon the difference between the input and output state dimensionalities. Let the dimensionality of s_i and $\tilde{s}_i$ be m and n, respectively, and assume that the transformation T_i consists of n independent equations, each one expressing one of the components of $\tilde{s}_i$ as a function of s_i and d_i. If all n equations are not independent, one deals with the largest set of independent equations and makes a corresponding adjustment in the dimensionality of the output state vector. There are three cases to be considered, $m = n$, $m > n$, and $m < n$:

1. If $m = n$, the m input state variables can be expressed in terms of the decision variables and n output state variables.
2. If $m > n$, n of the input state variables can be expressed in terms of the decision variables, n output state variables, and the remaining $m - n$ input state variables, which become choice variables.
3. If $m < n$, m of the equations of T_i can be used to express the m input variables in terms of the decisions and m of the output state variables; the remaining $m - n$ equations become constraints relating d_i and $\tilde{s}_i$.

Of course, the computational feasibility of state inversion is another question. Fortunately, when state inversion is not practical, there is another way of formulating and decomposing the final value problem.

FINAL VALUE AND TWO POINT BOUNDARY VALUE PROBLEMS—DECISION INVERSION

Even when state inversion is impractical, the N-decision, one-state final value problem can still be decomposed into N smaller problems. These subproblems each involve only one decision, but instead of a single state as in the initial value case, each has two state variables. Since the effort needed to solve these two-state subproblems is considerably greater than for one-state functions, this approach should only be used for final value serial problems when state inversion is either impossible or computationally infeasible. The main reason for studying this method is its applicability to two point boundary value problems arising in the optimization of certain systems with cycles and branches. Moreover, the apparent doubling of the state dimensionality is not usually as disastrous as might appear at first glance, for one of the state variables can under favorable circumstances be treated as a choice variable to which optimum seeking methods may be applied.

Suppose that

$$\tilde{s}_1 = T_1\,(s_1, \mathrm{d}_1) \qquad (5a)$$

can be solved for d_1 in terms of s_1 and $\tilde{s}_1$ to give

$$d_1 = \hat{T}_1\,(s_1, \tilde{s}_1) \qquad (22)$$

This mathematical interchange of the roles of d_1 and $\tilde{s}_1$ is called *decision inversion*, and Equation (22) tells what decision is needed to transform s_1 to $\tilde{s}_1$. As with state inversion, decision inversion calls for care when there are several output and decision variables. If for example there are more decision variables than output state variables, then there is still some freedom of choice left among the decisions which would permit limited optimization at stage 1. Even when there are exactly as many decisions as outputs and decision inversion is possible, any constraints on the decision variables will be translated into complicated state constraints. But although decision inversion is in principle just as difficult as state inversion, only one decision inversion is needed to solve a final value problem which would otherwise require N state inversions.

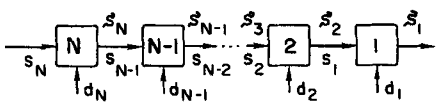

Fig. 1. A serial system.

For the serial system of Figure 1 the sum of returns from stages 1 through n can be expressed in terms of the $n - 1$ independent decisions $d_2, \ldots, d_n$ and the two states s_n and s_1 by applying decision inversion Equation (22) to Equation (13) to obtain

$$S_n = S_n\,(\tilde{s}_1, d_2, \ldots, d_n, s_n);\; n = 1, \ldots, N \qquad (23)$$

Let the two-state n-stage maximum return function $f_n(\tilde{s}_1, s_n)$ be defined by

$$f_n(\tilde{s}_1, s_n) \equiv \max_{d_2, \ldots, d_n} \{S_n(\tilde{s}_1, d_2, \ldots, d_n, s_n)\};\, n = 1, \ldots, N$$

When $n = 1$ the definition degenerates into an equation involving no maximization, since the right side of Equation (24) would have no decision variables:

$$f_1(\tilde{s}_1, s_1) = R_1(\hat{T}_1(s_1, \tilde{s}_1), \tilde{s}_1) \qquad (25)$$

Thus Equation (25) gives the optimal (and only possible) one stage return as a function of the input and output states.

As in the initial value problem, knowing $f_n(\tilde{s}_1, s_n)$ allows one to determine the maximum $n + 1$ stage return $f_{n+1}(\tilde{s}_1, s_{n+1})$ from the following relation:

$$f_{n+1}(\tilde{s}_1, s_{n+1}) = \max_{d_{n+1}} \{R_{n+1}(d_{n+1}, s_{n+1}) + f_n(\tilde{s}_1, T_{n+1}(d_{n+1}, s_{n+1}))\};\; n = 1, \ldots, N-1 \quad (26)$$

It is a one-decision, two-state optimization problem to find the two-state decision function $d^*_{n+1}(\tilde{s}_1, s_{n+1})$ such that

$$R_{n+1}(d^*_{n+1}(\tilde{s}_1, s_{n+1}), s_{n+1}) + f_n(s_1, T_{n+1}(d^*_{n+1}(\tilde{s}_1, s_{n+1}), s_{n+1})) = f_{n+1}(\tilde{s}_1, s_{n+1});\; n = 1, \ldots, N-1 \quad (27)$$

From the way the final value problem is formulated, the initial state (the state input to stage N) is free to take on its optimal value and is therefore a choice state, written c_N. Hence the desired final value maximum return function $R^*(s_1)$ is obtained ultimately by a two decision one-state optimization:

$$R^*(\tilde{s}_1) = \max_{c_N, d_N} \{R_N(c_N, d_N) + f_{N-1}(\tilde{s}_1, T_N(d_N, c_N))\} \qquad (28)$$

In all there is a decision inversion, only N—2 one-decision, two-state optimizations, and a two-decision no-state optimization. If s_N were not a choice state the problem would be a two-point boundary value problem with the maximization in Equation (28) being over d_N only. Boundary value problems always require decision inversion, while final value problems can be solved by either state or decision inversion.

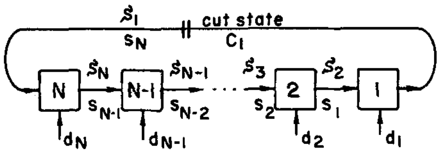
Fig. 2. A cyclic system.

Notice that $\tilde{s}_1$ appears as one of the states in every partial optimization. When a state such as $\tilde{s}_1$ is repeated at successive stages, and calculations are being done tabularly, its value should be fixed so that an ordinary one-state optimization by dynamic programing can be carried out. This completed, the maximum return and optimal policy for that value of $\tilde{s}_1$ can be stored and the intermediate calculations discarded. The computations are then repeated for another value of $\tilde{s}_1$ and so on. The advantage of multiple one-state optimizations over a single two-state optimization is the former method's saving in storage. Moreover, if $\tilde{s}_1$ itself is free to assume any value advantageous for maximizing the total return, then it can be treated as a choice state and directed by optimum seeking methods.

CYCLIC OPTIMIZATION

Consider now an N stage system similar to a serial one except that the initial state s_N and the final state $\tilde{s}_1$ are identical:

$$\tilde{s}_1 \equiv s_N \tag{29}$$

Such a system is said to be cyclic because its functional diagram, shown in Figure 2, is a closed loop. Cycles arise in chemical engineering recycle or feedback technologies. Since removal of the loop identity converts a cyclic system into a serial one, the variables $\tilde{s}_1$ will be called a *cut state* and written c_1. The use of this symbol already given to choice states is intentional, for it will be shown that any cut state is also a choice state (although not vice versa). The functional diagram shows that there are only N degrees of freedom in a cyclic system, one less than for a serial one, the loop identity having eliminated the input variable s_N.

The cyclic return function Φ depending on the cut state c_1 and the N—1 decisions $d_2 \ldots, d_N$ can be obtained from Equation (23) by using Equation (29) to eliminate s_N and writing c_1 for $\tilde{s}_1$:

$$\Phi(c_1, d_2, \ldots, d_N) = S_N(c_1, d_2, \ldots, d_N, c_1) \tag{30}$$

The maximum cyclic return Φ^* is defined by

$$\Phi^* \equiv \max_{c_1, d_2, \ldots, d_N} \{\Phi(c_1, d_2, \ldots, d_N)\} \tag{31}$$

The cyclic optimization problem is to find an optimal cyclic policy, that is a value c_1^* of the cut state and a set of optimal decisions $d_2^*, \ldots, d_N^*$ such that

$$\Phi(c_1^*, d_2^*, \ldots, d_N^*) = \Phi^* \tag{32}$$

This is an N-decision, no-state optimization problem.

To solve it, first select a particular value of c_1 and find the N $-$ 1 stage maximum return function $f_{N-1}(c_1, s_{N-1})$ by a decision inversion and $N - 2$ one-decision, one-state optimizations as described in the preceding section. Then find $\Phi^*(c_1)$ by the one-decision, no-state optimization:

$$\Phi^*(c_1) = \max_{d_N} \{R_N(c_1, d_N) + f_{N-1}(c_1, T_N(d_N, c_1))\} \tag{33}$$

This return, together with the corresponding policy $d_2^*(c_1), \ldots, d_N^*(c_1)$, is stored and the procedure re-

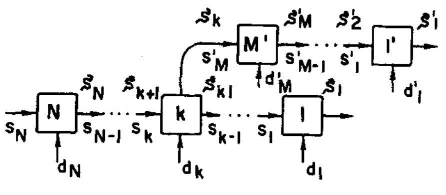
Fig. 3. A diverging branch.

peated for a different choice of c_1, perhaps selected by an optimum seeking technique. In this way a sequence of $N - 1$ decision serial problems is solved until one finds the optimal cut state c_1^*:

$$\Phi^* = \max_{c_1} \{\Phi^*(c_1)\} = \Phi(c_1^*) \tag{34}$$

Since there are two fewer one-decision optimizations than there are stages, the decomposition just described is not worth performing when the loop has less than three stages. In a two-stage loop, for example, there are only two degrees of freedom anyway, so one might as well perform a two-decision, no-state optimization directly without bothering to make a preliminary decision inversion. Similar considerations hold for a single-stage cycle, sometimes called a *self-loop*. The important thing in optimizing a self-loop correctly is to express the return in terms of a single variable, using both the loop identity (29) and the sole transition function (5) to eliminate two out of the three original variables s_1, $\tilde{s}_1$, and d_1. Related problems involving different kinds of loop structures are discussed later.

DIVERGING BRANCHES

The methods described previously for solving initial value serial problems can be adapted to the optimization of systems with a diverging branch as in Figure 3. In such systems one of the stages (say k) of a subsystem has, in addition to its ordinary output state s_{k1}, another output s_{k2} which is the initial state for a different sequence of M serial stages labelled 1′ through M' and forming a diverging branch. The branch transition functions are

$$\tilde{s}_i' = T_i'(d_i', s_i');\ i = 1, \ldots, M \tag{35}$$

and the incidence identities are

$$\tilde{s}_{i+1} \equiv s_i';\ i = 1, \ldots, M - 1 \tag{36}$$

Connection of the branch to the main system at stage k is represented by the additional transition function

$$\tilde{s}_{k2} = T_{k2}(d_k, s_k) \tag{37}$$

and incidence identity

$$s_{M'} \equiv \tilde{s}_{k2} \tag{38}$$

As usual there are M return functions

$$r_i' = R_i'(d_i', s_i');\ i = 1, \ldots, M \tag{39}$$

and it is desired to optimize the sum of the returns from all $M + N$ stages.

Let $f_{M'}(s_{M'})$ be the diverging branch maximum return function defined by

$$f_{M'}(s_{M'}) \equiv \max_{d_{1'}, \ldots, d_{M'}} \left\{ \sum_{i=1}^{M} r_i' \right\} \tag{40}$$

Finding this function is an initial value problem solvable by the methods described previously. By the connection Equation (37) and identify (38) this branch return can be combined with the stage k return to give a new return function depending only on d_k and state s_k:

$$R_k'(d_k, s_k) \equiv f_{M'}(T_{k2}(d_k, s_k)) + R_k(d_k, s_k) \tag{41}$$

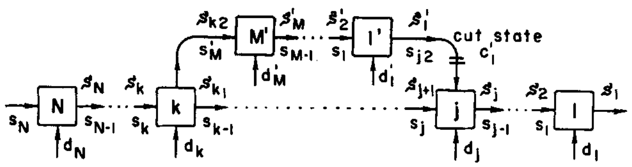
Fig. 4. A feed-forward loop.

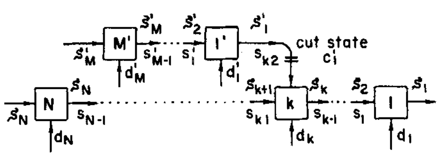
Fig. 5. A converging branch.

This new function can be used in place of $R_k(d_k, s_k)$ in the regular optimization plan for the main systems, even when the main system is not serial, since nonserial systems can be optimized by serial methods, as shown in the preceding section. The replacement of $R_k(d_k, s_k)$ by $R_k'(d_k, s_k)$ is called *absorption of a diverging branch.*

In the section to follow it will be important to know how to handle a final value version of the diverging branch problem. This would involve first finding the two-state M stage maximum return function $f_M'(\tilde{s}_1', s_M')$ of Equation (24) by using Equation (25) and iterating recursion relation (26) $M - 1$ times. This would take a decision inversion and $M - 1$ one-decision, two-state optimizations. Absorption of the branch requires that the three variable function $R_k'(d_k, \tilde{s}_1', s_k)$ be substituted for $R_k(d_k, s_k)$ in the main stem optimization scheme:

$$R_k'(d_k, \tilde{s}_1', s_k) \equiv R_k(d_k, s_k) + f_M'(\tilde{s}_1, T_{k2}(d_k, s_k)) \tag{42}$$

SINGLE LOOP OPTIMIZATION

Suppose that the output state $\tilde{s}_1'$ from the diverging branch of the preceding section is also an input to some stage j of the main stem. When, as in Figure 4, $j < k$, a configuration arising for example in the bypass and multiple feed technologies of chemical engineering, the system is said to have a feedforward loop comprised of stages k through j together with stages 1′ through M'. On the other hand if $j \geqq k$, the loop is described as feedback, a situation brought about in the chemical industries by recycling unreacted materials back to the feed for reprocessing. The cyclic system of Figure 2 is a special feedback case where $j = 1$ and $k = N$.

Finding the maximum total return from all stages of either feedforward or feedback systems is called the *single loop optimization problem.* Its mathematical description differs only slightly from that of the diverging branch problem. Thus one need only add the incidence identity

$$s_{j2} \equiv \tilde{s}_1' \tag{43}$$

and replace Equations (4) and (5) for the junction stage j by the three-variable functions

$$r_j = R_j(d_j, s_j, s_{j2}) \tag{44}$$

$$\tilde{s}_j = T_j(d_j, s_j, s_{j2}) \tag{45}$$

To optimize such a system, feedback or feedforward, one treats $\tilde{s}_1'$ as a cut state, rewritten c_1' or c_{j2}. When this cut state is fixed at a specific value, the system can be treated as a serial one with a diverging branch having a fixed output c_1'.

Applying decision inversion to state 1 one finds the diverging branch maximum return $f_M'(s_M', c_1')$. However, when c_1' is treated as a constant, the $M - 1$ optimizations are all one-state one-decision problems. Stages 1 through $j - 1$ are treated using the initial value model to obtain $f_{j-1}(s_{j-1})$. For the junction stage j the return and transition Equations (44) and (45) are used to obtain $f_j(s_j, c_1')$, again for the particular value of c_1'. Then $f_{k-1}(s_{k-1}, c_1')$, is determined by proceeding in this way through stage $k - 1$. At stage k, absorption Equation (42) is applied to absorb $f_M'(s_M, c_1')$ into r_k. From a partial optimization at stage k, $f_k(s_k, c_1')$ is obtained together with the decision functions $d_i'^*(s_i, c_1')$, $i = 1, \ldots, M$, $d_i^*(s_i)$, $i = 1, \ldots, j - 1$ and $d_i^*(s_i, c_1')$, $i = j, \ldots, k$. These functions are all stored, and the process is then repeated for a new value of c_1', possibly selected by an optimum seeking method. Ultimately this gives the optimal value $c_1'^*$ and hence $f_k(s_k)$. Stages $k + 1$ through N are then treated with the initial value model by applying Equation (15). At $n = N$ one obtains the maximum total return $f_N(s_N) = \Phi^*(s_N)$:

$$\Phi^*(s_N) \equiv \max_{c_1'} \{\Phi^*(c_1', s_N)\} = \Phi^*(c_1'^*, s_N) \tag{46}$$

This return function is of course a single value if the input state s_N is a constant k_N, and if it is a choice variable c_N the optimum seeking method can guide c_N simultaneously with c_1' to find the single maximum return Φ^*:

$$\Phi^* \equiv \max_{c_1', c_N} \{\Phi^*(c_1', c_N)\} \tag{47}$$

CONVERGING BRANCHES

Figure 5 illustrates a system with an M stage converging serial branch, that is one whose output state $\tilde{s}_1'$ is an input to stage k ($\neq 1$) of another N stage system. Although the return, transition, and incidence Equations (35), (36), and (39) for a diverging branch hold also for a converging one, the mathematical description of connecting stage k is different. The diverging junction Equations (4), (5), (6), (37), and (38) must be replaced by the triple input functions

$$r_k = R_k(d_k, s_{k1}, s_{k2}) \tag{48}$$

and

$$\tilde{s}_k = T_k(d_k, s_{k1}, s_{k2}) \tag{49}$$

with the new incidence relations being

$$\tilde{s}_{k-1} \equiv s_{k1} \tag{50}$$

and

$$\tilde{s}_1' \equiv s_{k2} \tag{51}$$

The problem is to maximize the sum of returns from all $M + N$ stages. To solve it, first find $f_{k-1}(s_{k-1})$, the optimal return function for stages 1 through $k - 1$, using dynamic programing Equations (14) and (15). If s_M' must be considered a state variable, next choose a particular value c_1' of the branch output $\tilde{s}_1'$, treated as a cut state, and determine the optimal branch return $f_M'(c_1', s_M)$, a boundary value serial problem. Simultaneous selection of a value of decision d_k would, for a given value of state s_k, determine a total return which is the maximand inside the braces in Equation (52) below. Once this quantity has been stored, together with the corresponding policy, new values of c_1' and d_k can be chosen by an optimum seeking method. Repetition of this guided search eventually gives $c_1'^*(s_k)$ and $d_k^*(s_k)$, the optimal values of c_1' and d_k for every value of state s_k, as well as the optimal $M + k$ stage return function $f_{M+k}(s_k, s_M)$, defined as

$$f_{M+k}(s_k, s_M') \equiv \max_{c_1', d_k} \{R_k(c_1', d_k, s_k) + f_{k-1}(T_k(c_1', d_k, s_k)) + f_M'(c_1', s_M')\} \tag{52}$$

This is not difficult to do if the branch input s_M' is a constant k_M'.

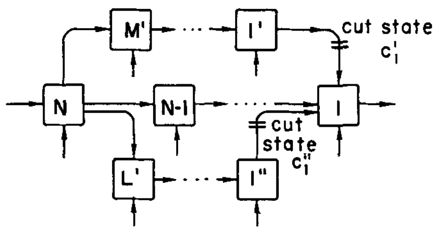

Fig. 6. A double loop.

When s_M can be treated as a choice state c_M, the optimal branch return $f_{M'}(c_{1'})$ can be found more easily by state inversion of the branch, followed by solution of M one-decision one-state problems. Moreover the optimization of Equation (52) would then involve only one state variable (s_k). On the other hand, if state inversion were not possible and decision inversion had to be used, one would determine $f_{M'}(c_{1'}, c_M)$, treating $c_{1'}$ as a cut state as given above, and then determine f_{M+k} by a three-decision one-state optimization using Equation (53):

$$f_{M+k}(s_k) \equiv \max_{c_{1'}, c_{M'}, d_k} \{R_k(c_{1'}, d_k, s_k) + f_{k-1}(T_k(c_{1'}, d_k, s_k)) + f_{M'}(c_{1'}, c_{M'})\} \quad (53)$$

This illustrates the advantage of using state inversion whenever possible, since with state inversion the combining branch problem is only about as difficult as the diverging branch problem. Otherwise, even though there is no loop, its difficulty would be comparable to that of a single loop problem. If s_N is a choice state but s_M is not, stages $k + 1$ through N should be treated as the converging branch and stages $(1, \ldots, k, 1', \ldots, M')$ as the serial chain.

The rest of the main trunk is optimized by ordinary dynamic programing. It is important to carry out the double (or triple) decision optimization right at the junction stage k, for otherwise the states $\mathfrak{s}_{1'}$ and $s_{M'}$ would have to be carried as state variables to be examined exhaustively during every partial optimization following.

MULTIPLE LOOPS

The methods described for solving single loop optimization problems can be extended to systems with several loops. For example, cutting the double loop system of Figure 6 in the two places shown permits solution of the system as if it were a serial one with two branches. As one would expect, the presence of two loops requires two cut states, but in favorable circumstances these can be guided simultaneously to their optimum values by an optimum seeking method. The power of the concept of cut state becomes more apparent as the number of loops increases, for solution of multiple loop problems would be computationally impractical if the cut states had to be examined exhaustively.

There are many different ways to cut multiloop and multibranch systems. Since choosing a good optimization plan is primarily a strategic problem, such questions will be deferred to a later article.

VARIATIONAL METHODS

Systems of any structure may be optimized by the methods demonstrated as long as one can find the functions $R^*(s_N)$, $R^*(\mathfrak{s}_1)$, and $f_{N-1}(\mathfrak{s}_1, s_N)$. The first is the solution to an initial value problem, the second to a final value problem, and the third to a two-point boundary value problem. To solve such problems one need not necessarily carry out the decomposition procedure of dynamic programing. Often it is more practical to use the calculus of variations or related techniques such as the maximum principle of Pontryagin to find those functions. The purpose of this article has not been to indicate preference for any of these methods, but rather to show how to decompose certain microstructures into initial, final, or two-point systems solvable by serial procedures.

NOTATION

c	=	cut state variable, Equation (8)
d	=	decision variable, Equation (4)
f	=	maximum return function, Equation (14)
k	=	constant state
M	=	number of stages in a branch Equation (35)
N	=	number of serial stages, Equation (6)
P	=	transformation process
r	=	return, Equation (1)
R	=	return function, Equation (1)
R^*	=	optimal return, Equation (8)
$\mathfrak{s}$	=	state variable, Equation (3)
S	=	sum of stage returns, Equation (12)
s	=	input state, Equation (4)
T	=	transition function, Equation (3)
$\mathcal{T}$	=	inverse transition function, Equation (18)
$\hat{T}$	=	decision inversion function, Equation (22)
x	=	input variable, Equation (2)
y	=	output variable
Φ	=	cyclic return function, Equation (30)
$'$	=	branch stage identification, Equation (35)

Subscripts

i	=	stage index, Equation (2)
j	=	converging junction stage, Equation (44)
k	=	diverging junction stage, Equation (37)
n	=	number of stages partially optimized, Equation (12)

LITERATURE CITED

1. Aris, Rutherford, "Discrete Dynamic Programming," Blaisdell, New York (1964).
2. ———, and Yesberg, *J. Math. Anal. Appl.*, **7**, No. 3, p. 421 (Dec., 1963).
3. Bellman, R., "Dynamic Programming," Princeton Univ. Press, Princeton, New Jersey (1957).
4. ———, "Adaptive Control Processes: a Guided Tour," p. 54, Univ. Press, Princeton, New Jersey, (1960).
5. ———, and S. Dreyfus, "Applied Dynamic Programming," Univ. Press, Princeton, New Jersey (1962).
6. Chang, S. S. L., "Synthesis of Optimum Control Systems," pp. 217-22, McGraw-Hill, New York (1961).
7. Fan, L. T., and C. S. Wang, *Ind. Eng. Chem. Fund Quart.*, 3, No. 1, p. 38 (Feb., 1964).
8. ———, *Chem. Eng. Sci.*, **19**, 86-7 (Jan., 1964).
9. Horn, F., *Zeitschrift für Electrochemie*, **65**, No. 3, pp. 209-22 (1961).
10. Jackson, R., *Chem. Eng. Sci.*, **19**, 19-31 (Jan., 1964).
11. Katz, S., *J. Electron Contr.*, **13**, 179 (1962).
12. Kiefer, J., *Proc. Am. Math. Soc.*, **4**, 502 (1953).
13. Mitten, L., and G. Nemhauser, *Chem. Eng. Progr.*, **59**, 52 (1963).
14. Pontryagin, L. S., V. G. Boltyanskii, R. V. Gamkrelidze, and E. F. Mishchenko, "The Mathematical Theory of Optimal Processes," translated by K. N. Trirogoff, Interscience, New York (1962).
15. Wilde, D., "Advances in Chemical Engineering," Vol. 3, pp. 293-302, Academic Press, New York (1962).
16. Wilde, D., "Optimum-Seeking Methods," Prentice-Hall, Englewood Cliffs, New Jersey (1964).

Manuscript received February 10, 1964; revision received April 23, 1964; paper accepted April 24, 1964.