BRANCH-AND-BOUND METHODS: A SURVEY*

E. L. Lawler and D. E. Wood

The University of Michigan, Ann Arbor, Michigan

(Received February 11, 1966)

The essential features of the branch-and-bound approach to constrained optimization are described, and several specific applications are reviewed. These include integer linear programming (LAND-DOIG and BALAS methods), nonlinear programming (minimization of nonconvex objective functions), the traveling-salesman problem (EASTMAN AND LITTLE, ET. AL. methods), and the quadratic assignment problem (GILMORE AND LAWLER methods). Computational considerations, including trade-offs between length of computation and storage requirements, are discussed and a comparison with dynamic programming is made. Various applications outside the domain of mathematical programming are also mentioned.

AMONG the most general approaches to the solution of constrained optimization problems is that of 'branching-and-bounding' (or, in the words of BERTIER AND ROY,[10] "séparation et évaluation progressives"). Like dynamic programming, branching-and-bounding is an intelligently structured search of the space of all feasible solutions. Most commonly, the space of all feasible solutions is repeatedly partitioned into smaller and smaller subsets, and a lower bound (in the case of minimization) is calculated for the cost of the solutions within each subset. After each partitioning, those subsets with a bound that exceeds the cost of a known feasible solution are excluded from all further partitionings. The partitioning continues until a feasible solution is found such that its cost is no greater than the bound for any subset. These basic ideas of branching-and-bounding are formalized in the next section of this paper.

There has been some recent interest in branch-and-bound methods for the solution of integer linear programming and traveling-salesman problems. These particular applications, and others, are reviewed in the third section. The remaining sections discuss general computational considerations, relations with dynamic programming, and various applications outside the domain of mathematical programming.

ESSENTIAL FEATURES OF BRANCHING-AND-BOUNDING

THERE ARE many constrained optimization problems for which 'direct' methods of solution either do not exist, or are inefficient. Problems may

* This research was supported in part by NSF Grant GP-2778, and in part by Air Force contract AF 30(602)-3546.

be of this nature because the objective function or constraints are non-convex, or some or all of the variables are restricted to discrete values. Branching-and-bounding enables us to solve these 'difficult' problems by applying existing methods of solution to 'easy' problems.

Suppose we are confronted with a 'difficult' constrained optimization problem of the form

$$\left.\begin{array}{ll} \text{minimize} & c^{(0)}(x) \\ \text{subject to} & g_1^{(0)}(x) \geqq 0, \\ & g_2^{(0)}(x) \geqq 0, \\ & \cdots \\ & g_m^{(0)}(x) \geqq 0 \\ \text{and} & x \in X^{(0)}, \end{array}\right\} \qquad (0)$$

where the set $X^{(0)}$ denotes the permissible domain of optimization, e.g., the positive orthant of Euclidean n-space, and x denotes a vector $(x_1, x_2, \cdots, x_n)$. In line with accepted terminology, a solution vector x that satisfies the constraints and lies within the domain of optimization is said to be a *feasible* solution, and a feasible solution for which $c^{(0)}(x)$ is minimal is said to be an *optimal* solution.

One way to solve a difficult problem is to solve a related 'easy' problem, and hope that the solution to the easy problem can be shown to be a solution to the difficult problem. In the case at hand, suppose we replace problem (0) with an easy problem (1), which *bounds* problem (0), in the sense that the following 'bounding' property is satisfied.

(B) There exists at least one optimal solution $x^{(0)}$ of problem (0), such that $x^{(0)}$ is feasible for problem (1) and $c^{(1)}(x^{(0)}) \leqq c^{(0)}(x^{(0)})$.

[$c^{(1)}$ denotes the objective function of problem (1).]

Suppose we obtain an optimal solution $x^{(1)}$ to problem (1). It is not hard to show that if $x^{(1)}$ satisfies the following *optimality* conditions:

(O1) $x^{(1)}$ is a feasible solution to problem (0),

and

$$(O2) \qquad c^{(1)}(x^{(1)}) = c^{(0)}(x^{(1)}),$$

then $x^{(1)}$ is an optimal solution to problem (0) as well.

Of course, if the solution $x^{(1)}$ does not satisfy both (O1) and (O2) we must try something else. If condition (O1) is not satisfied, we might try to strengthen the constraint or to restrict the domain of optimization of problem (1), in order to make the solution $x^{(1)}$ infeasible for (1). If (O2) is not satisfied, we might attempt to modify the objective function $c^{(1)}$ so as to increase the cost of $x^{(1)}$, in order to either make $x^{(1)}$ nonoptimal, or

to cause the equality (O2) to be satisfied. Regardless of what the objective is, it generally proves to be easiest to replace problem (1) by a *set* P = {(2), (3), ···} of problems that bound problem (0) in the sense that they jointly satisfy the following generalized bounding property:

(B′) There exists at least one optimal solution $x^{(0)}$ of problem (0), such that $x^{(0)}$ is feasible for at least one problem $(j)\epsilon P$, and $c^{(j)}(x^{(0)}) \leqq c^{(0)}(x^{(0)})$.

Now suppose we obtain an optimal solution $x^{(j)}$ to each problem $(j)\epsilon P$. Let $x^{(k)}$ be such that

$$c^{(k)}(x^{(k)}) = \min_{(j)\epsilon P} c^{(j)}(x^{(j)}).$$

It is not hard to show that $x^{(k)}$ is an optimal soluton to problem (0) if conditions (O1) and (02) are satisfied (with $x^{(k)}$ replacing $x^{(1)}$ in their statement).

Now, of course, if the solution $x^{(k)}$ does not satisfy both the conditions (O1) and (O2), we must again try something else. Our approach is to replace one of the problems in the bounding set P by a set of new problems. For example, suppose we replace problem (k) by a set of problems $P^{(k)}$. In addition to requiring that the new set of bounding problems $(P-\{k\})\cup P^{(k)}$ satisfy the bounding condition (B'), we may impose the following weak 'convergence' condition,

(C) For each problem $(j)\epsilon P^{(k)}$, either $x^{(k)}$ is infeasible for (j) or $c^{(j)}(x^{(k)}) > c^{(k)}(x^{(k)})$,

or the stronger condition,

(C′) For each problem $(j)\epsilon P^{(k)}$ and each feasible solution x to problem (k), either x is infeasible for (j) or $c^{(j)}(x) > c^{(k)}(x)$.

These conditions are not sufficient to guarantee that repeated revision of the set P will yield an optimal solution to problem (0) with a finite amount of computation. However, they do represent minimal conditions we would like to impose in order that some progress toward a final solution can be made. (Actually, in practice the finiteness of branch-and-bound methods is rarely an issue.)

A tree like that shown in Fig. 1 helps to visualize matters. Each node of the tree is identified with a problem (j), which is of the form

$$\left.\begin{array}{ll} \text{minimize} & c^{(j)}(x), \\ \text{subject to} & g_1^{(j)}(x) \geqq 0, \\ & g_2^{(j)}(x) \geqq 0, \\ & \cdots \\ & g_m^{(j)}(x) \geqq 0, \\ \text{and} & x\epsilon X^{(j)}. \end{array}\right\} \qquad (j)$$

(The number of constraints may vary from problem to problem; for notational simplicity, this number is simply designated as m.) The problems that replace problem (j) in the bounding set are pointed to by the branches directed outward from node (j) in the tree. (Hence the term 'branching,' in 'branching-and-bounding.') Thus, at any intermediate point in the calculations, the set P of current bounding problems is identified with the set of nodes that are 'leaves' of the tree, as it has been developed at that point in the computation.

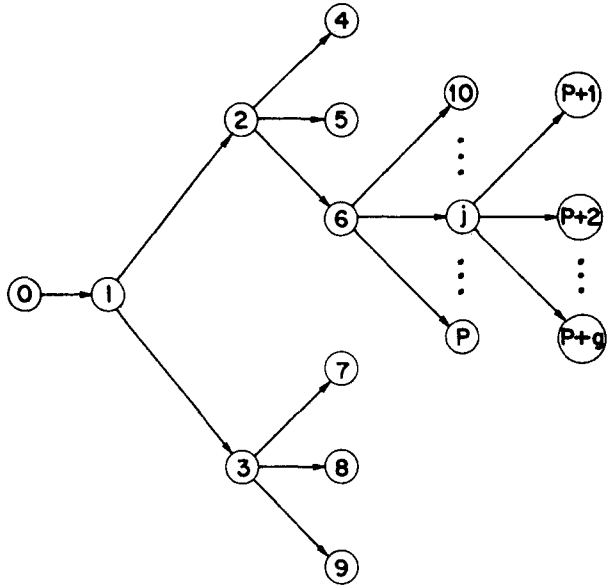


Fig. 1. Tree representation.

The total amount of computation is related to the number of distinct bounding problems created, and hence to the total number of nodes in the fully developed tree. The amount of temporary storage required is related to maximum cardinality of the bounding set P, and hence to the maximum number of leaves of the tree at any intermediate stage in its development. In a later section, we describe how various strategies may be employed to reduce the amount of computation at the expense of additional storage, and vice-versa. We now merely make a simple observation that indicates how the cardinality of P can be reduced somewhat.

During the course of the calculations, various feasible solutions to problem (0) are discovered. At any intermediate stage, let $\hat{x}$ denote the least costly (in terms of $c^{(0)}$) feasible solution that has been discovered. Sup-

pose (j) is a bounding problem with optimal solution $x^{(j)} \neq \hat{x}$. Then, if $c^{(0)}(\hat{x}) \leqq c^{(j)}(x^{(j)})$, it follows that problem (j) can be removed from the set P without affecting the bounding condition (B').

We associate with each node (j) of the tree a *bound* $c^{(j)}(x^{(j)})$. We say that any leaf node of the tree whose bound is strictly less than $c^{(0)}(\hat{x})$ is *active*; otherwise it is designated as *terminated*, and need not be considered in any further computation. Roughly speaking, what we seek to do in branching-and-bounding is to develop the tree until every leaf can be terminated.

The complete description of a branch-and-bound algorithm requires a specification of the rule that determines which of the currently active bounding problems is to be chosen for branching, as well as the method for deriving new bounding problems. In the fourth section, various rules for choosing active problems are discussed; this discussion will be more meaningful in the context of the applications presented in the following section.

APPLICATIONS

FOUR AREAS of application in mathematical programming will serve to illustrate branching-and-bounding. These are (1) integer programming, (2) nonlinear programming, (3) the traveling-salesman problem, and (4) the quadratic assignment problem. Branch-and-bound methods can, of course, be applied to a variety of other types of problems in scheduling, combinatorics, decision processes, etc.

(1) Integer Programming

It is well-known that a wide variety of nonlinear and combinatorial problems can be formulated as integer linear programming problems. These problems differ from ordinary linear programming problems in that the variables are restricted to nonnegative integer values:

minimize	cx,	
subject to	$Ax \geqq b$,	
	x_j nonnegative integer.	$(j=1, 2, \cdots, n)$

The Gomory cutting-plane and all-integer algorithms are often used to solve these problems. Another approach has been suggested by LAND AND DOIG.[28] This is roughly as follows: Apply the simplex method to the problem. If the optimal solution thereby obtained is integral (and there are many linear programs for which this will be the case), the problem is solved. If the optimal solution is not integral, then create two new problems as follows: Choose some variable that has a noninteger value (say $x_3 = 4.4$), and restrict that variable to the next lower integer value for one

problem ($x_3 \leqq 4$) and to the next higher integer value for the other ($x_3 \geqq 5$). The process is then repeated on each of the new problems.

Under the Land-Doig approach all bounding problems are conventional linear programming problems, and hence differ from problem (0) essentially only in their domain of optimization. Each feasible solution x of problem (0) is a feasible solution of at least one bounding problem (j) and it is always true that $c^{(0)}(x) = c^{(j)}(x)$, hence condition (B') is easily seen to be satisfied. Branching forces fractional solutions to become infeasible, and condition (C') is seen to hold.

The Land-Doig approach is particularly appropriate for 'mixed,' problems (only a proper subset of the variables restricted to integers). Various elaborations and refinements of the Land-Doig approach have been proposed and tested by BEALE AND SMALL,[6] DAKIN,[11] and DRIEBECK.[12] Reportedly, computational results have been favorable, i.e., competitive with cutting-plane and all-integer methods. In particular, Dakin[11] reports the solution of a problem with 17 constraints and 93 variables, 80 of which were constrained to be integers. The solution of this problem took 7 minutes and 28 seconds on a KDF9 computer, while the GOMORY[22] mixed-integer algorithm failed to find the solution in over 2000 iterations. Several smaller problems were reported solved in less than a minute on the KDF9.

Whereas the Land-Doig approach uses the simplex method to obtain a lower bound on the integer program, a totally different approach, not employing the simplex method, is proposed by Balas.[2,3,4] Balas characterizes his algorithm as 'additive,' in that no multiplications or divisions are required.

For convenience, Balas deals only with variables that are restricted to 0, 1 values. (An integer variable x_j that is bounded above by $2^K - 1$ can be expressed as the sum of $K(0, 1)$-variables: $x_{j1}, x_{j2}, \cdots, x_{jK}$ by the relation $x_j = x_{j1} + 2x_{j2} + \cdots + 2^{K-1}x_{jK}$. For each problem that is created by branching, Balas partitions the variables into three classes:

(1) those which are set to 0,

(2) those which are set to 1,

(3) those which are free to assume either the value 0 or 1.

Assume all cost coefficients c_j are positive (it is a simple matter to put the problem into this form). If the solution obtained by setting all variables in class (3) to 0 is feasible, it is clearly optimal. If this solution is not feasible the sum of the c_j corresponding to the class (2) variables plus the smallest c_j corresponding to a class (3) variable is an easily-obtained lower bound on the cost of an optimal solution. Branching is effected by forming one problem in which a specific class (3) variable is transferred to class (1) and a second problem in which the same variable is transferred to class (2).

An essential feature of the Balas algorithm is the testing of the bounding problems for infeasibility. Each linear constraint is examined to see if it can be satisfied by setting each class (3) variable equal to its 'more favorable' value. Inability to satisfy a constraint implies the nonexistence of feasible solutions; however, since the constraints are tested independently, the converse is not true. An infeasible bounding problem is, of course, terminated. (In effect, its bound is $+\infty$.)

Although some persons view the Balas algorithm as involving 'branching,' but not 'bounding,' the present authors do not share this view. The concept of bounding is certainly an essential aspect of the algorithm. The only departure from the formulation of the preceding section is that the bounds determined are not the actual costs of optimal solutions to the bounding problems, but are lower bounds on the actual costs.

R. J. FREEMAN[16] reports favorable results with the Balas algorithm as modified by GEOFFRION[18] at Rand. Freeman reports the solution of the ten fixed-charge problems given by HALDI[23] in times ranging from 2 seconds to 7 minutes on an IBM 7090. The sizes of these problems ranged from 4 inequalities in 14 binary variables to 6 inequalities in 20 binary variables. Also closely related to the method of Balas are the proposals of BENDERS, ET AL.,[9] GILMORE AND GOMORY,[20] and LAWLER AND BELL.[32]

(2) *Nonlinear Programming*

Branch-and-bound methods are possibly the best way to obtain globally optimal solutions to nonlinear programming problems in which a nonconvex function is to be minimized.

A limited amount of computation experience has been reported by BEALE[5] for a particular problem of this type. However, this experience is not sufficient to base any general conclusions concerning the expected amount of computation or general applicability of the approach. The following example is intended to be suggestive of the possibilities in this area.

Suppose we seek to minimize a separable cost function

$$f(x) = \sum_{j=1}^{j=n} f_j(x_j),$$

where the f_j (and therefore f) are not necessarily convex functions. Suppose, for example, that f_1 is as shown in Fig. 2.

Suppose we can bound x_j from above by x_j'. We define the *convex envelope* of f_j to be the largest function f_j^c such that

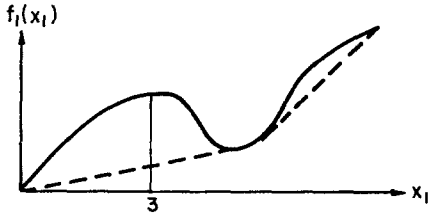
(1) $\qquad f_j^c(x_j) \leqq f_j(x_j), \qquad (0 \leqq x_j \leqq x_j')$

(2) $\qquad f^c$ is convex.

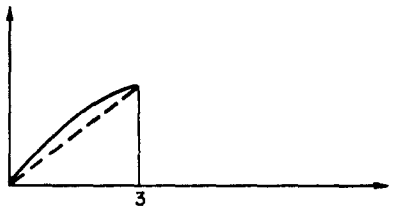
The convex envelope of f_j is indicated by dashed lines in Fig. 2.

Having replaced each f_j by its convex envelope, we proceed to apply an appropriate convex programming algorithm to the minimization of the convex function

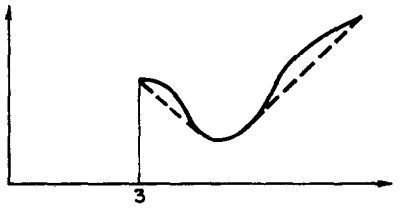
$$f^c(x) = \sum_{j=1}^{j=n} f_j^c(x_j),$$



(a) Nonconvex Function and Convex Envelope.



(b) Convex Envelope for Restriction of Function to $x_1 \leq 3$.



(c) Convex Envelope for Restriction of Function $x_1 \geq 3$

Fig. 2. Example of nonconvex function.

subject to the constraints of the original problem. If the optimal solution x^0 thereby obtained has the property that

$$f^c(x^0) = f(x^0),$$

then a global optimum has been obtained and the problem is solved. If this is not the case, then two new problems are created as follows. Choose some variable x_j such that

$$f_j^c(x_j^0) < f_j(x_j^0).$$

In one of the new problems, f_j^c is replaced by a convex envelope that is valid for the interval $0 \leqq x_j \leqq x_j^0$ and in the other for the interval $x_j^0 \leqq x_j \leqq x_j'$. This is illustrated in Fig. 2 for the case $x_1^0 = 3$.

It is believed that, properly managed, the process will yield a globally optimal solution in a finite number of steps, provided each f_j is piecewise convex. In any case, computations can be carried out until a solution is obtained that differs from the optimum by no more than a predetermined amount (*see* the fifth section).

Note that this proposal uses branching to overcome the obstacle of a nonconvex cost function, but could also be used to overcome nonconvex constraints. Problems involving nonconvex (but monotone) cost functions and nonconvex constraints can also be solved by the method of Lawler and Bell.[32]

(3) *The Traveling-Salesman Problem*

The well-known traveling-salesman problem requires for its solution the shortest possible circuit through n cities, where the distances between the cities are given. Any feasible solution to this problem can be expressed in terms of n^2 variables $x_{ij} (i = 1, 2, \cdots, n; j = 1, 2, \cdots, n)$, where

$x_{ij} = 1$ if the solution requires the salesman to travel from city i directly to city j,
$= 0$ otherwise.

The salesman is to enter each city exactly once, so we have,

$$\sum_{i=1}^{i=n} x_{ij} = 1, \qquad (j = 1, 2, \cdots, n)$$

and he is to leave it exactly once, so

$$\sum_{j=1}^{j=n} x_{ij} = 1. \qquad (i = 1, 2, \cdots, n)$$

We are to minimize

$$\sum_{i=1}^{i=n} \sum_{j=1}^{j=n} d_{ij} x_{ij},$$

where d_{ij} represents the distance from city i to city j.

It is evident that the traveling-salesman problem is very much like the conventional assignment problem. Indeed, *the n-city traveling-salesman problem is an $n \times n$ assignment problem with the added constraint that the solution must be cyclic*, i.e., representable by a cyclic permutation of the integers 1 to n.

The approach of EASTMAN[13,14] is to solve the assignment problem corresponding to the traveling-salesman problem. If the optimal solution

thereby obtained is cyclic, then the problem is solved. On the other hand, if the solution is not cyclic, a constraint can be added as follows.

Suppose the optimal solution to the assignment problem contains the subcycle (1, 3, 5), i.e., $x_{13}=1, x_{35}=1, x_{51}=1$. At least one of the links in this subcycle must be missing from any feasible solution to the traveling-salesman problem. Hence, a possible constraint is

$$\begin{aligned} &\text{either} && x_{13}=0, \\ &\text{or} && x_{35}=0, \\ &\text{or} && x_{51}=0. \end{aligned}$$

This suggests three new problems. A feasible solution to the traveling-salesman problem is a feasible solution to *at least* one of the new problems.

Consider the 10-city (unsymmetric) problem as defined by the following matrix:

	1	2	3	4	5	6	7	8	9	10
1	∞	24	18	22	31	19	33	25	30	26
2	15	∞	19	27	26	32	25	31	28	18
3	22	23	∞	23	16	29	27	18	16	27
4	24	31	18	∞	19	13	28	9	19	27
5	23	18	34	20	∞	31	24	15	25	8
6	24	12	17	15	10	∞	11	16	21	31
7	28	15	27	35	19	18	∞	21	21	19
8	13	24	18	13	13	22	25	∞	29	24
9	17	21	18	24	27	24	34	31	∞	18
10	18	19	29	16	23	17	18	31	23	∞

The initial assignment problem, which has this matrix as its cost matrix, we designate problem (1). An optimal solution to this assignment problem is the permutation (1, 6, 7, 2) (3, 9) (4, 8, 5, 10), with a cost of 140.

Two new assignment problems are created: problem (2) with $x_{39}=0$ and problem (3) with $x_{93}=0$. These two problems have optimal solutions with costs of 147 and 143 respectively. Since there is a lesser lower bound on the solutions to problem (3), we proceed to branch from it next. An optimal solution to problem (3) is the permutation (1, 6, 7, 2, 3, 9) (4, 8, 5, 10), and we create problems (4), (5), (6), and (7), for which $x_{48}=0$, $x_{85}=0$, $x_{5,10}=0$, $x_{10,4}=0$, respectively.

The complete solution of the example is indicated in Fig. 3a. The nodes of the tree represent assignment problems and the branches are given labels which denote excluded links. The number, l, within the node is an index specifying the point in the computation where problem (l) is derived, while

the bound for problem (l) is the number adjacent to the node. It can be seen that the problem requires the solution of 11 assignment problems. (There are various ways in which this number could be reduced.) At the conclusion, when a cyclic optimal solution is found to problem (11), with a cost of 146, every other assignment problem has a (noncyclic) optimal solution whose cost is at least as great.

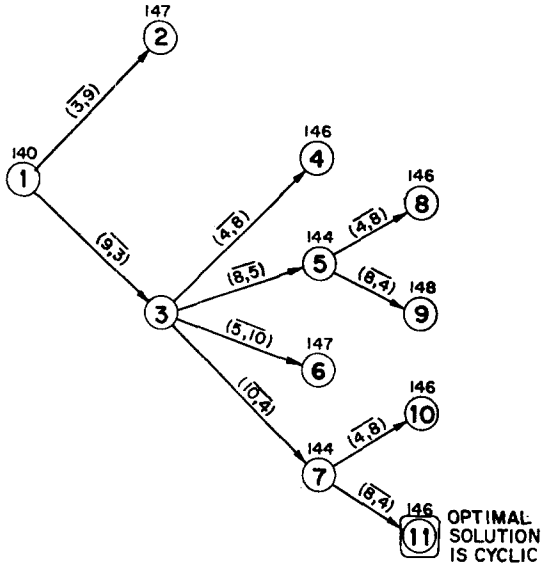


Fig. 3a. Solution of example of traveling-salesman problem using the Eastman[13] approach.

LITTLE'S ET AL.[34] approach to the traveling-salesman problem differs mainly in that

(1) two problems are created at each branching step, corresponding to $x_{ij}=0$ and $x_{ij}=1$.
(2) an optimal solution to the assignment problem is not obtained. Instead, the smallest element is subtracted from each row of the distance matrix, and then the smallest element from each column of the result. The sum of these elements is a valid (and fairly good) lower bound on the optimal solution.

Little employs what appears to be a very worthwhile heuristic. At each branching step, the pair (i, j) is chosen in such a way that the problem corresponding to $x_{ij}=0$ will yield as large a bound as possible. The bound is deemed to be less important for the other problem (corresponding

to $x_{ij}=1$), since the dimension of this problem is effectively diminished by one.

Little's method is similar to Balas' method for integer programming in that instead of seeking optimal solutions to bounding problems, only lower bounds on the costs of optimal solutions are calculated. This enables

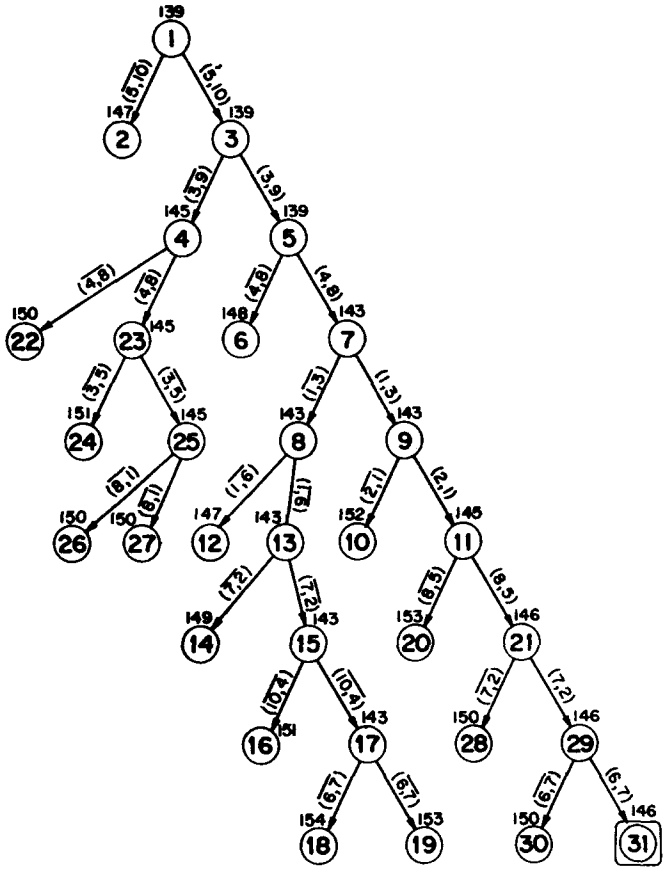


Fig. 3b. Solution of example of traveling-salesman problem using the Little, et. al.[14] approach.

Little to take advantage of one of the trade-offs possible in branch-and bound methods. Namely, a poorer, but more easily calculated, bound results in more branching operations, but less computation at each node. The solution of the example problem by the Little, et al., algorithm is indicated in Fig. 3b with the same notational conventions for the nodes as in Fig. 3a. A branch label of the form (j, k) indicates that link (j, k)

is included in the solution. On the other hand, $(\overline{j, k})$ indicates that link (j, k) is excluded. The trade-off between the amount of computation for each bounding problem and the number of problems solved is dramatically seen by a comparison of Figs. 3a and 3b.

Little's, et al., computational results have been favorable. The mean computation time on an IBM 7090 for 100 thirty-city problems was 58.5 seconds and for 5 forty-city problems was 8.37 minutes.

Eastman's method is not known to have been programmed.

Another type of bound which could be used for the traveling-salesman problem, but has not been tested, is based on the construction of minimal spanning trees. One city can be duplicated to produce an $(n+1)$-city network. Then a minimal tree sapnning for all $n+1$ cities can be calculated by one of the simple algorithms available for that purpose. The total length of the arcs in the tree is a valid lower bound (sometimes better and sometimes worse) than the bound obtained from an optimal solution to the assignment problem.

(4) *The Quadratic Assignment Problem*

The quadratic assignment problem differs from the ordinary assignment problem only in that an arbitrary quadratic cost function is to be minimized:

$$\text{minimize} \quad \sum_{i,j} \sum_{p,q} c_{ijpq}\, x_{ij}\, x_{pq},$$

$$\text{subject to} \quad \sum_i x_{ij} = 1, \qquad (j = 1, 2, \cdots, n)$$

$$\sum_j x_{ij} = 1, \qquad (i = 1, 2, \cdots, n)$$

$$x_{ij} = 0 \text{ or } 1. \qquad (i, j = 1, 2, \cdots, n)$$

GILMORE[19] (for a somewhat more specialized version of the problem) and LAWLER[30] have proposed essentially equivalent branch-and-bound methods. The bound is obtained by solving n^2 (usually very simple) $(n-1)\times(n-1)$ linear assignment problems and then one $n\times n$ assignment problem for which the initial computations determine the cost coefficients.

Gilmore reports some fairly favorable computational experience with problems of moderate dimension.

(5) *Miscellaneous Examples*

Efroymson and Ray[15] describe a branch-and-bound algorithm for the plant location problem. Simply stated, the plant location problem is a transportation problem in which there are no constraints on the sources (plants), but there is a setup cost incurred in opening each plant. The algorithm is extended to handle the cases where

(1) there is a 'variable plant cost' as well as the fixed cost;
(2) no fixed cost, but concave plant cost—two linear segments;
(3) a fixed cost and many linear segments.

A number of 50-plant, 200-customer problems have been solved on an IBM 7094 in about 15 minutes each. The problems were of the types indicated in (1) and (2) above.

The techniques of branching and bounding have also been applied to job shop and machine scheduling problems. LOMNICKI[35] describes a branch-and-bound algorithm for the solution of machine-scheduling problems with three machines. While there is no computer processing times given, the amount of computation is comparable with that reported by INGALL AND SCHRAGE.[25] Ingall and Schrage report the solution of three-machine scheduling problems with up to 10 jobs to be scheduled in less than 3 minutes on a CDC 1604. The mean time for 8 problems with 6 jobs was 1.8 seconds of CDC 1604 time.

The application of branch-and-bound techniques to sequencing problems and assembly-line balancing is given by JAESCHKE.[26] Jaeschke further considers the traveling-salesman problem in a manner similar to Little, et al.

STRATEGIES OF BRANCHING

ANY BOUNDING problem with a bound less than the cost of the least costly feasible solution discovered to date is a possible candidate for branching. A policy is needed to choose among the candidates, which may be quite large in number. Two such policies are

(1) branch from lowest bound,
and (2) branch from newest active bounding problem.

(1) *Branch from Lowest Bound*

As the name suggests, this policy is to branch from that bounding problem, which currently has the lowest bound. Suppose that for any given problem the set of new bounding problems is uniquely determined. Then this policy has the advantage that the total amount of computation is minimized, in the sense that any branching operation performed under this policy must also be performed under any other policy. However, the volume of intermediate data that must be stored in the computer may be quite large.

Land-Doig and Eastman proposed using the branch-from-lowest bound policy.

(2) *Branch from Newest Active Bounding Problem*

The idea here is always to branch from the latest problem created by branching. This strategy can be implemented with a 'pushdown' stack.

Each time branching is carried out, the new problems are placed on the top of the stack, and those below are 'pushed down.' Each time branching is to be performed, the problem currently on the top of the stack is examined. If it is active, branching is carried out upon it. If it is not active, it is discarded and the next problem on the top of the stack is examined. This policy has the distinct advantage that, for most problems, a minimum of computer storage is required. (The maximum size of the stack is proportional to the length of the longest possible path directed away from the root of the branching tree.) However, many more branching operations may have to be performed than under the branch-from-lowest-bound policy.

Little, et al., argued in favor of the branch-from-newest-bounding-problem policy. However, the branch-from-lowest-bound policy was used in reported computations.

It should be noted that in most situations the set of derived bounding problems at any point in the computation is not uniquely determined. For example, consider the Balas algorithm for integer programs, with the variables restricted to be either 0 or 1. In this case we may choose any one of the available 'free' variables [i.e., class (3) variables] x_j to create two new problems corresponding to $x_j=0$ and $x_j=1$. The choice of which x_j to choose in general must be made by some heuristic rule, such as that used by Little, et al.

SUBOPTIMIZATION

A VERY USEFUL feature of branching-and-bounding is the opportunity to compute solutions that differ from the optimum by no more than a prescribed amount. Suppose, at the outset, that it is decided that a feasible solution whose cost exceeds the cost of an optimal solution by no more than 10 per cent would be acceptable. Then, if a feasible solution is found with a cost of 150, we can terminate all problems with bounds of 137 or more ($1.10\times137=150.7>150$). Both Gilmore and Lawler have pointed out that this technique would substantially reduce the amount of computation required for the quadratic assignment problem.

In order to find the best solution possible in a bounded length of time, various 'adaptive' computational schemes are possible. A scheme to find the best solution in a length of time T might proceed as follows:

(i) Search for an optimal solution for a length of time equal to $T/2$. If an optimal solution is not found proceed to (ii).

(ii) Search for a suboptimal solution that differs from the optimal by no more than 5 per cent. If such a solution is not found in length $T/4$ proceed to (iii).

(iii) Search for a suboptimal solution differing from the optimum by no more than 10 per cent for $T/8$ units of time, etc.

RELATION TO DYNAMIC PROGRAMMING

CERTAIN branch-and-bound algorithms are closely related to dynamic programming. A further discussion of the traveling-salesman problem may help to illustrate this relation.

There are many ways to branch in the traveling-salesman problem. We may initially create $n-1$ problems, corresponding to the alternatives

$$(1) \qquad x_{12}=1,\ x_{1j}=0, \qquad (j\neq 2)$$

$$(2) \qquad x_{13}=1,\ x_{1j}=0, \qquad (j\neq 3)$$

$$\cdots$$

$$(n-1) \qquad x_{1n}=1,\ x_{1j}=0. \qquad (j\neq n)$$

Each of these $n-1$ problems has dimension $n-1$. The appropriate $(n-1)$-dimensional assignment problem for case (k) is obtained by removing row 1 and column k from the distance matrix.

Now suppose we carry out further branching upon each of these $(n-1)$-dimensional problems. For problem (k) we have these cases:

$$x_{k2}=1,\ x_{kj}=0, \qquad (j\neq 2)$$

$$x_{k3}=1,\ x_{jk}=0, \qquad (j\neq 3)$$

$$\cdots$$

$$x_{kn}=1,\ x_{kj}=0. \qquad (j\neq n)$$

The appropriate $(n-2)\times(n-2)$ assignment problem for the case $x_{kl}=1$ is obtained by removing row k and column l from the matrix of distances.

When the process is carried one step further, many of the assignment problems are seen to be identical. For example, the $(n-3)\times(n-3)$ assignment problem for the case $x_{12}=x_{23}=x_{34}=1$ is the same as the problem for the case $x_{13}=x_{32}=x_{24}=1$. In both cases, the first, second, and third rows and second, third, and fourth columns of the original distance matrix have been removed. All that differs between the two problems are the 'constant' costs $d_{12}+d_{23}+d_{34}$ and $d_{13}+d_{32}+d_{24}$.

If we represent the branching process by a tree, as before, and we identify those nodes that correspond to equivalent assignment problems, we notice that the tree 'collapses,' as shown in Fig. 4.

This result should be compared with BELLMAN'S dynamic programming treatment[7] of the same problem, which is as follows. With no loss in generality, fix the origin of the circuit at some city, say 1. Suppose that at some stage in an optimal circuit starting at city 1 one has reached city i and there remain k cities $j_1, j_2, \cdots, j_k$ to be visited before returning to 1. Then it is clear that if the tour is to be optimal, the path from i through $j_1, j_2, \cdots, j_k$ in some order and then to 1 must be of minimum length.

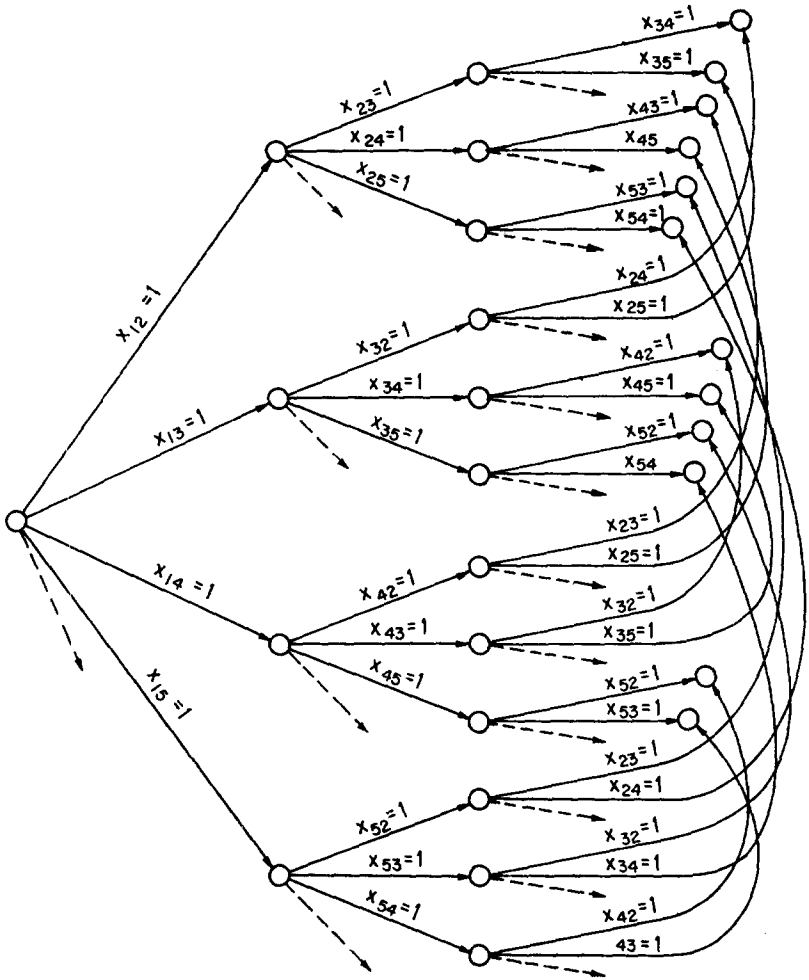


Fig. 4. Collapsing of tree for traveling-salesman problem.

Define

$C(i; j_1, j_2, \cdots, j_k)$ = the length of a path of minimum length from i to 1, which passes once and only once through each of the remaining k unvisited cities $j_1, j_2, \cdots, j_k$.

Thus, $C(1; 2, 3, \cdots, n)$, and the path that has this as its length, constitutes a solution to the problem.

It is easy to show that

$$C(i;j_1,j_2,\cdots,j_k)=\min_{1\leq m\leq k}\{d_{ij_m}+C(j_m;j_1,j_2,\cdots,j_{m-1},j_{m+1},\cdots,j_k)\}$$

and $$C(i;j)=d_{ij}+d_{j1},$$

which enables us to define a sequence of computations.

The computational scheme for dynamic programming is essentially to develop the *complete* collapsed tree shown in Fig. 4, to associate with the directed arc $x_{ij}=1$ the distance d_{ij}, and then to find the shortest path between the two ends of the tree. The traveling-salesman problem is thus converted to a shortest-route problem.

The branch-and-bound approach uses an assignment problem to calculate a lower bound on the length of the shortest path from any given node in the collapsed tree to the right-hand end. Thus, various routes are eliminated from consideration, even though the length of the shortest path is not known with precision.

NONMATHEMATICAL PROGRAMMING APPLICATIONS

COMPUTATIONAL techniques that are essentially the same as, or very similar to, branch-and-bound methods, have been applied in areas outside of mathematical programming. These include

(1) switching circuit minimization,
(2) 'artificial intelligence,' e.g., game-playing and theorem proving,
(3) 'pure' combinatorics.

(1) *Switching Circuit Minimization*

The synthesis problem for switching circuits is roughly as follows. Given a supply of 'elementary' switching circuits, use as few of these components as possible to construct a switching circuit having a prescribed input-output behavior. Algorithms proposed and programmed by J. Paul Roth and his associates at IBM definitely fall under the heading of branch-and-bound methods.[27,31,43,44,45]

It should also be mentioned that one of the fundamental problems in switching theory is the 'covering' problem, i.e., the special type of integer program with an A matrix composed entirely of 0's and 1's and a b-vector composed entirely of 1's. Switching theorists, such as Roth, and McCLUSKEY,[36] have proposed branch-and-bound methods for solving these problems. See also LAWLER.[33]

(2) *Artificial Intelligence*

It has long been observed that the analysis of chess and similar games can be carried out by means of a tree, where the branches of the tree correspond to possible moves. Game-playing programs, such as the checker-playing program of SAMUEL[46] use 'scoring functions' or other estimates to

assign values to the branches in an effort to reduce the extent of the tree that must be explored in order to make an intelligent move. The effect is a branch-and-bound method with (necessarily) inexact bounds.

An essentially similar approach is used for some types of theorem proving and heuristic programming.

(3) *Pure Combinatorics*

Branching methods are commonly used to enumerate or to prove or disprove the existence of combinatorial designs of various types. These methods should possibly be classified as 'branch-and-exclude' rather than 'branch-and-bound' in character. PROFESSOR D. H. LEHMER of the University of California at Berkeley has named this method *backtrack*. WALKER[49] and GOLOMB AND BAUMERT [21] have applied backtracking to various combinatorial problems.

REFERENCES

1. ABADIE, J., "État actuel de la programmation linéaire," presented at *Compte-Rendu du Congres de Paris*, August 1961.
2. BALAS, E., "Un algorithme additif pour la résolution des programmes linéaires en variables bivalentes," *C. R. Acad. Sc. Paris* **258,** 3817–3820 (1964).
3. ———, "Extension de l'algorithme additif à la programmation en nombres entiers et à la programmation non linéaire," *C. R. Acad. Sci. Paris* **258,** 5136–5139 (1964).
4. ———, "An Additive Algorithm for Solving Linear Programs with 0-1 Variables," *Opns. Res.* **13,** 517–549 (1965).
5. BEALE, E. M. L., "Two Transportation Problems," *Actes de la 3eme Conference Internationale de Recherche Operationelle*, Oslo (1963), 780–788, Dunod (1964).
6. ———, AND SMALL, R. E., "Mixed Integer Programming by a Branch-and-Bound Technique," presented at *IFIP Congress* **65,** New York, May 1965.
7. BELLMAN, R. E., "Dynamic Programming Treatment of the Traveling-Salesman Problem," *J. Assoc. for Comp. Mach.* **9,** 61–63 (1962).
8. ———, AND DREYFUS, S. E., *Applied Dynamic Programming*, Princeton University Press (1962).
9. BENDERS, J. F., CATCHPOLE, A. R., AND KUIKEN, L. C., "Discrete-Variable Optimization Problems," *Rand Symposium on Mathematical Programming*, Santa Monica, California, March 1959.
10. BERTIER, P., AND ROY, B., "Procédure de Résolution pour une Classe de Problèmes pouvant avoid un Caractère Combinatoire," *Cahiers du Centre D'Études de Recherche Opérationnelle* **6,** 202–208 (1964).
11. DAKIN, R. J., "A Tree-search Algorithm for Mixed Integer Programming Problems," *The Computer Journal* **8,** 250–255 (1965).
12. DRIEBECK, N. J., "An Algorithm for the Solution of Mixed Integer Programming Problems," Management Sci. **12,** 576–587 (1966).
13. EASTMAN, W. L., *Linear Programming with Pattern Constraints*, Ph.D. Thesis,

Report No. BL. 20, The Computation Laboratory, Harvard University (1958).

14. ———, "A Solution to the Traveling-Salesman Problem," presented at the *American Summer Meeting of the Econometric Society*, Cambridge, Massachusetts, August 1958.
15. Efroymson, M. A., and Ray, T. L., "A Branch-Bound Algorithm for Plant Location," Esso Research and Engineering Company (mimeographed) (1965).
16. Freeman, R. J., "Computational Experience with the Balas Integer Programming Algorithm," The Rand Corporation, P-3241, October 1965.
17. Gavett, J. W., "Three Heuristic Rules for Sequencing Jobs to a Single Production Facility," *Management Sci.* **11**, B-166–B-176 (1965).
18. Geoffrion, A., "Integer Programming by Implicit Enumeration and Balas' Method," The Rand Corporation, RM-4783-PR, February, 1966.
19. Gilmore, P. C., "Optimal and Suboptimal Algorithms for the Quadratic Assignment Problem," *J. Soc. Indust. Appl. Math.* **10**, 305–313 (1962).
20. ———, and Gomory, R. E., "A Linear Programming Approach to the Cutting-Stock Problem, Part II," *Opns. Res.* **11**, 863–888 (1963).
21. Golomb, S. W., and Baumert, L. D., "Backtrack Programming," *J. Assoc. for Comp. Mach.* **12**, 516–524 (1965).
22. Gomory, R. E., "An Algorithm for the Mixed Integer Problem," The Rand Corporation, RM-2597 (1960).
23. Haldi, J., "Twenty-Five Integer Programming Test Problems," Working Paper no. 43, Graduate School of Business, Stanford University, December 1964.
24. Hall, M., Jr., and Knuth, D. E., "Combinatorial Analysis and Computers," *Am. Math. Monthly* **72**, 21–28 (1965).
25. Ignall, E., and Schrage, L., "Application of the Branch-and-Bound Technique to Some Flow-Shop Scheduling Problems," *Opns. Res.* **13**, 400–412 (1965).
26. Jaeschke, G., "Eine allgemeine Methode zur Lösung kombinatorischer Probleme," *Ablanf- und Planungforschung* **5**, 133–155 (1964).
27. Karp, R. M., McFarlin, F. E., Roth, J. P., and Wilts, J. R., "A Computer Program for the Synthesis of Combinatorial Switching Circuits," *Proc. Second Annual Symp. on Switching Circuit Theory and Logical Design*, AIEE Publication S-134 (1961).
28. Land, A. H., and Doig, A., "An Automatic Method of Solving Discrete Programming Problems," *Econometrica* **28**, 497–520 (1960).
29. ———, "A Problem of Assignment with Interrelated Costs," *Opnal. Res. Quart.* **14**, 185–199 (1963).
30. Lawler, E. L., "The Quadratic Assignment Problem," *Management Sci.* **9**, 586–599 (1963).
31. ———, "An Analysis of Roth's Methods of Synthesis," presented at the *Seventh Midwest Symposium on Circuit Theory*, Ann Arbor, Michigan, May 1964.
32. ———, and Bell, M. D., "A Method for Solving Discrete Optimization Problems," te appear in *Opns. Res.*

33. LAWLER, E. L., "Covering Problems: Duality Relations and a New Method of Solution," to appear in *Journal of SIAM*.
34. LITTLE, J. D. C., MURTY, K. G., SWEENEY, D. W., AND KAREL, C., "An Algorithm for the Traveling-Salesman Problem," *Opns. Res.* **11**, 972–989 (1963).
35. LOMNICKI, Z. A., "A 'Branch-and-Bound' Algorithm for the Exact Solution of the Three-machine Scheduling Problem," *Opnal. Res. Quart.* **16**, 89–100 (1965).
36. MCCLUSKEY, E. J., "Minimization of Boolean Functions," *Bell System Tech. J.* **35**, 1417–1444 (1956).
37. PANDIT, S. N. N., "The Loading Problem," *Opns. Res.* **10**, 639–646 (1962).
38. ———, "Some Observations on the Longest Path Problem," *Opns. Res.* **12**, 361–364 (1964).
39. ———, "An Enumerational Approach to Integer Programming Problems," (to appear).
40. RADO, F., "Linear Programming with Logic Conditions," *Communicarile Academiei Republicil Populare Romine* **13**, 1039–1041 (1963).
41. ROSSMAN, M. J., AND TWERY, R. J., "Combinatorial Programming," presented at the *Sixth Annual ORSA Meeting*, May 1958 (mimeographed).
42. ———, AND TWERY, R. J., "A Solution to the Traveling-Salesman Problem by Combinatorial Programming," Peat, Marwick, Mitchell and Co., Chicago (mimeographed).
43. ROTH, A. P., AND WAGNER, E. G., "Algebraic Topological Methods for the Synthesis of Switching Systems. Part III: Minimization of Nonsingular Boolean Trees," *IBM J. of Res. Dev.* **3**, 326–344 (1959).
44. ———, "Minimization over Boolean Trees," *IBM J. of Res. Dev.* **4**, 543–558 (1960).
45. ———, AND KARP, R. M., "Minimization over Boolean Graphs," *IBM J. of Res. Dev.* **6**, 227–238 (1962).
46. SAMUEL, A. L., "Some Studies in Machine Learning Using the Game of Checkers," *IBM J. of Res. Dev.* **3**, 210–229 (1959).
47. STANLEY, E. D., HONIG, D. P., AND GAINEN, L., "Linear Programming in Bid Evaluation," *Naval Res. Log. Quart.* **1**, 48–54 (1954).
48. SWEENEY, D. W., "The Exploration of a New Algorithm for Solving the Traveling-Salesman Problem," M.S. Thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts (1963).
49. WALKER, R. S., "An Enumerative Technique for a Class of Combinatorial Problems," *Am. Math. Soc. Symposium on Applied Math. Proc.* **10**, 91–94 (1960).