Integer Programming Algorithms: A Framework and State-of-the-Art Survey

A. M. Geoffrion; R. E. Marsten

Management Science, Vol. 18, No. 9, Theory Series (May, 1972), 465-491.

Stable URL:
http://links.jstor.org/sici?sici=0025-1909%28197205%2918%3A9%3C465%3AIPAAFA%3E2.0.CO%3B2-6

Management Science is currently published by INFORMS.

MANAGEMENT SCIENCE
Vol. 18, No. 9, May, 1972
Printed in U.S.A.

INTEGER PROGRAMMING ALGORITHMS: A FRAMEWORK AND STATE-OF-THE-ART SURVEY*†

A. M. GEOFFRION AND R. E. MARSTEN‡

University of California, Los Angeles

A unifying framework is developed to facilitate the understanding of most known computational approaches to integer programming. A number of currently operational algorithms are related to this framework, and prospects for future progress are assessed.

I. Introduction

This paper was written with two complementary purposes in mind. The first is to undertake a survey of existing general purpose integer linear programming algorithms that have been implemented with a notable degree of computational success. The second is to explain a general algorithmic framework for integer programming that we have found quite useful in organizing our understanding of the field.

We have tried to make this paper accessible to as wide an audience as possible by keeping the exposition elementary. Readers wishing to delve more deeply will find suitable references throughout the text. In addition, there are the excellent previous surveys [8], [10], and [9], and also such general frameworks for enumerative type algorithms as [1], [6], [16], [48], [59], and [62].

The proposed general framework is presented in §II, after a discussion of the three underlying concepts on which it is based: separation, relaxation, and fathoming. No claim is made for the originality of these concepts; they appear, either explicitly or implicitly, in many of the references just mentioned. What distinguishes the present treatment is perhaps mainly in the realm of emphasis—especially the central role assigned here to the use of relaxation—and the recognition that one framework serves equally well for algorithms of both enumerative and nonenumerative type.

§III surveys selected current algorithms. In each case it is indicated how the algorithm fits into the general framework, with digressions as necessary to explain any new features of general interest. Computational experience is then cited. The algorithms are grouped according to whether they are based primarily on enumeration (§3.1), Benders Decomposition (§3.2), cutting-planes (§3.3), or on group theory (§3.4). Regrettably, space does not permit discussion of a number of effective algorithms proposed recently for special applications such as vessel scheduling [3], facilities location [21], and crew scheduling [52]. A review of numerous specialized algorithms can be found in [9].

§IV attempts to summarize the current state-of-the-art for integer programming, and presents some opinions regarding promising directions for the future development of the field.

Throughout this paper, the canonical statement of the mixed integer linear programming problem is taken to be

$$\text{(MIP)} \qquad \text{Minimize}_{x \geqq 0;\, y \geqq 0} \quad cx + dy$$
$$\text{subject to} \quad Cx + Dy \leqq b, \qquad x \text{ integer},$$

* Received March 1971.

† This research was supported by the National Science Foundation under Grant GP-26294. An earlier version was presented at the Third Operations Research Symposium, jointly sponsored by CPPA and TAPPI, Montreal, Quebec, August 24–26, 1970.

‡ Now at Northwestern University.

where C and D are matrices and c, d, b, x and y are vectors of appropriate dimensions. By taking d and D to be null, this problem specializes to the "pure integer" linear program. It will be assumed for simplicity that the feasible region of (MIP) is bounded if it is not empty.

II. A General Framework

The general algorithmic framework is built upon three key notions: separation, relaxation, and fathoming. Each of these is discussed in turn before the general framework itself is presented.

2.1. *Separation*

For any optimization problem (P), let $F(P)$ denote its set of feasible solutions. Problem (P) is said to be *separated* into subproblems $(P_1), \cdots, (P_q)$ if the following conditions hold:

(S1) *Every feasible solution of* (P) *is a feasible solution of exactly one of the subproblems* $(P_1), \cdots, (P_q)$.

(S2) A feasible solution of any of the subproblems $(P_1), \cdots, (P_q)$ is a feasible solution of (P).

These conditions assert that $F(P_1), \cdots, F(P_q)$ is a partition of $F(P)$. The subproblems $(P_1), \cdots, (P_q)$ are called *descendants* of (P). Creating descendants of the descendants of (P) is equivalent to refining the partition of $F(P)$.

Our interest in separation is that it enables an obvious divide-and-conquer strategy for solving (P). Leaving aside for a moment the important question of *how* one separates a problem that is difficult to solve, we can sketch a rudimentary strategy of this type as follows. First make a reasonable effort to solve (P). If this effort is unsuccessful, separate (P) into two or more subproblems, thereby initiating what will be called a *candidate list* of subproblems. Extract one of the subproblems from this list—call it the current *candidate problem* (CP)—and attempt to solve it. If it can be solved with a reasonable amount of effort, go back to the candidate list and extract a new candidate problem to be attempted; otherwise, separate (CP) and add its descendants to the candidate list. Continue in this fashion until the candidate list is exhausted. If we refer to the best solution found so far to any candidate problem as the current *incumbent*, then the final incumbent must obviously be an optimal solution of (P) (if all candidate problems were infeasible, then so is (P)).

The finite termination of such an approach is obviously assured if $F(P)$ is finite [provided that there is a finite limit on the number of times a degenerate separation of a candidate problem can occur due to all but one descendant being infeasible]. The usefulness of this strategy depends, of course, upon a sufficient level of success at solving the candidate problems without further separation. Considerable subsequent discussion will be addressed to this point.

The most popular way of separating an integer programming problem is by means of contradictory constraints on a single integer variable (the separation or branching variable). For example, if x_4 is declared to be a binary variable, then (MIP) can be separated into two subproblems by means of the mutually exclusive and exhaustive constraints "$x_4 = 0$" and "$x_4 = 1$". For general integer variables there are other possibilities. For example, if x_5 is declared to be an integer between 0 and 4, then the constraints $x_5 = 0$, $x_5 = 1$, $x_5 = 2$, $x_5 = 3$, $x_5 = 4$ are mutually exclusive and exhaustive, as are $0 \leqq x_5 \leqq 2$, $3 \leqq x_5 \leqq 4$. Several possible rules for choosing the integer variable on which the separation should be based will be mentioned in §III.

Sometimes special problem structure can be exploited by devising more powerful separation techniques. Beale and Tomlin [12] report success along these lines for problems containing "multiple choice" constraints such as $x_1 + x_2 + x_3 + x_4 + x_5 = 1$, where each of the variables is binary. A separation can be effected here by mutually exclusive conditions of the sort $x_1 + x_2 + x_3 = 0$, $x_4 + x_5 = 0$. In this case the freedom of several variables is restricted simultaneously.

2.2. *Relaxation*

Any constrained optimization problem (P) can be *relaxed* by loosening its constraints, resulting in a new problem (P_R). For example, one may simply omit some of the constraints of (P). Other possibilities for relaxation include dropping the integrality conditions or the nonnegativity conditions on the variables of (P). The only requirement for (P_R) to be a valid relaxation of (P) is: $F(P) \subseteq F(P_R)$. This defining property of relaxation trivially implies the following relationships, where, by convention, (P) is taken to be a minimization problem:

(R1) If (P_R) has no feasible solutions, then the same is true of (P).

(R2) The minimal value of (P) is no less than the minimal value of (P_R).

(R3) If an optimal solution of (P_R) is feasible in (P), then it is an optimal solution of (P).

In selecting between alternative types of relaxation for a given problem, there are two main criteria to be considered. On the one hand, it is desirable for the relaxed problem to be significantly easier to solve than the original. On the other hand, for reasons that will become clear, one would like (P_R) to yield an optimal solution of (P) via (R3) or, failing that, the minimal value of (P_R) should be as close as possible to that of (P). Unfortunately these objectives tend to be antagonistic. In general, the easier (P_R) is to solve, the greater is the "gap" between the original and relaxed problems.

In §III the principal relaxation techniques used in integer programming will be illustrated by reference to various algorithms. By far the most popular type of relaxation for an integer linear program is to drop all integrality requirements on the variables. The resulting ordinary linear program is often an excellent compromise between the two criteria mentioned above. A number of other interesting types of relaxation can be gleaned, for instance, from [40], [42], [60], [75], [76]. A discussion of the role of relaxation in continuous mathematical programming can be found in [30, §3.3].

2.3. *The Fathoming Criteria*

In §2.1 a simple divide-and-conquer strategy based on the notion of separation was briefly sketched. A sequence of candidate problems must be examined, and whenever one of them cannot be solved with a reasonable amount of effort, it must be separated and its descendants must be examined subsequently. Use of the vague phrase "reasonable amount of effort" was deliberate, for the success of the strategy depends to a great extent on the judicious choice of how and how hard to try to solve each candidate problem. It is in this regard that the concept of relaxation plays an important role; rather than trying to deal with the candidate problem itself, one deals instead with some relaxation of it. The so-called fathoming criteria introduced below are an attempt to clarify and formalize this role.

Let (CP) be a typical candidate problem arising from the attempt to solve (P). The ultimate objective in dealing with (CP) is to determine whether its feasible region

$F(CP)$ may possibly contain an optimal solution of (P) and, if so, to find it. If it can be ascertained by some means that $F(CP)$ cannot contain a feasible solution better than the incumbent (the best feasible solution yet found), this is certainly good enough to dismiss (CP) from further consideration, and we say that (CP) has been *fathomed.* Or if an optimal solution of (CP) can actually be found, we also say that (CP) has been fathomed. In either case, the candidate problem has been entirely accounted for and can be discarded without further separation.

It is useful to distinguish three general types of fathoming, all of them based on relaxation. We shall suppose that a particular relaxation (CP_R) of the candidate (CP) has been decided upon and that (CP_R) has been solved. All problems are to minimize, $v(\cdot)$ denotes the optimal value of problem $(\cdot)$, and Z^* is the value of the incumbent (let $Z^* = +\infty$ if no feasible solution of (P) has yet been found).

If (CP_R) has no feasible solution, then by (R1) the same is true of (CP). Thus $F(CP)$ is empty and cannot possibly contain an optimal solution of (P). In this event (CP) is certainly fathomed.

Secondly, if

$$v(CP) \geqq Z^* \tag{1}$$

then $F(CP)$ cannot possibly contain a feasible solution of (P) better than the incumbent. Unfortunately, $v(CP)$ is unknown, but $v(CP_R)$ *is* known and by (R2) we have $v(CP) \geqq v(CP_R)$. It follows that (CP) has been fathomed if

$$v(CP_R) \geqq Z^*. \tag{2}$$

Note that it is possible to have $v(CP) \geqq Z^* > v(CP_R)$, so that condition (1) is satisfied but condition (2) is not satisfied. In this case the outcome of (CP_R) will not permit discarding (CP). It now becomes evident why, as mentioned earlier, it is desirable to have $v(CP_R)$ as close to $v(CP)$ as possible.

Thirdly, suppose that an optimal solution of (CP_R) has been found which happens to be feasible in (CP). Then by (R3) this solution must be optimal for (CP) and hence (CP) is fathomed. By (S2), of course, it is also feasible in (P) and so becomes the new incumbent if its value is less than Z^*.

The above discussion can be summarized and slightly generalized in terms of three *fathoming criteria.* Candidate problem (CP) is fathomed if any one of these criteria is satisfied.

(FC1) An analysis of (CP_R) reveals that (CP) has no feasible solution; e.g., $F(CP_R) = \varnothing$.

(FC2) An analysis of (CP_R) reveals that (CP) has no feasible solution better than the incumbent; e.g., $v(CP_R) \geqq Z^*$.

(FC3) An analysis of (CP_R) reveals an optimal solution of (CP); e.g., an optimal solution of (CP_R) is found which happens to be feasible in (CP).

Among the various algorithms that have been proposed, there is considerable variation in the kinds of "analysis" actually employed to implement these criteria. The standard analysis, which conforms to the discussion above, is to solve (CP_R) optimally and then check whether $F(CP_R) = \varnothing$ or $v(CP_R) \geqq Z^*$ or the optimal solution obtained for (CP_R) is feasible in (CP). But there are deviations from this norm in both directions. Some algorithms do not go so far as to solve (CP_R) optimally, but merely employ sufficient conditions for the tests $F(CP_R) = \varnothing$ and $v(CP_R) \geqq Z^*$ (e.g., any lower bound $\hat{v}(CP_R)$ on $v(CP_R)$—say a good suboptimal solution of the dual problem if (CP_R) is a linear program—leads to the sufficient condition $\hat{v}(CP_R) \geqq Z^*$). Other algorithms not only go so far as to solve (CP_R) optimally, but continue

the analysis even further in an effort to account at least partially for whatever "gap" there may be between (CP) and (CP_R). As we shall see in §III, when (CP_R) is a linear program this further analysis is often a kind of postoptimality study aimed at finding a better lower bound $\hat{v}(CP)$ on the value of (CP) than $v(CP_R)$ itself; the test $\hat{v}(CP) \geqq Z^*$ is then stronger than the standard test $v(CP_R) \geqq Z^*$.

There are two alternative courses of action that may be taken if a particular relaxation (CP_R) of (CP) does not pass any of the fathoming criteria. The first is to separate (CP) immediately and add its descendants to the candidate list. The second alternative is to persist in trying to fathom (CP), as by choosing a new relaxation (CP_R'). For example, some previously ignored constraints can be reimposed. Each new relaxation provides another chance to pass one of the fathoming criteria. The relaxation of (CP) can be modified as many times as desired, but if it appears that too much time is being expended on this one candidate problem, a switch ought to be made to the first alternative—separation.

2.4. *General Procedure*

The ingredients of a general procedure for solving (MIP) are now at hand. A flowchart is given in Figure 1.

Step 1. Initialize the candidate list to consist of (MIP) alone, and Z^* to be an arbitrarily large number.

Step 2. Stop if the candidate list is empty: if there exists an incumbent then it must be optimal in (MIP), otherwise (MIP) has no feasible solution.

Step 3. Select one of the problems from the candidate list to become the current candidate problem (CP).

Step 4. Choose a relaxation (CP_R) of (CP).

Step 5. Apply an appropriate algorithm to (CP_R).

Step 6. Fathoming Criterion 1. If the outcome of Step 5 reveals (CP) to be infeasible (e.g., $F(CP_R) = \varnothing$), go to Step 2.

Step 7. Fathoming Criterion 2. If the outcome of Step 5 reveals that (CP) has no feasible solution better than the incumbent (e.g., $v(CP_R) \geqq Z^*$), go to Step 2.

Step 8. Fathoming Criterion 3. If the outcome of Step 5 reveals an optimal solution of (CP) [e.g., an optimal solution of (CP_R) is feasible in (CP)], go to Step 12.

Step 9. Decide whether or not to persist in attempting to fathom (CP). If so, go to Step 10; otherwise go to Step 11.

Step 10. Modify (perhaps by tightening) the relaxation of (CP) and return to Step 5.

Step 11. Separate (CP) and add its descendants to the candidate list. Go to Step 2.

Step 12. A feasible solution of (MIP) has been found; if $v(CP) < Z^*$, record this solution as the new incumbent and set $Z^* = v(CP)$. Go to Step 2.

Remarks. Step 1. Prior knowledge about (MIP) can be used to improve the (default) initializations given above for Step 1. One may know an upper bound Z_0 on the optimal value of (MIP), for example, in which case initializing Z^* to the value Z_0 will cause the procedure to look only for solutions with value less than Z_0. One may also have reason to prefer starting with an initial candidate list obtained by making certain prior separations of (MIP). This too is permissible, and simply leads to an initial candidate list with more than one member.

Step 2. If Z^* was initialized at Z_0 rather than an arbitrarily large number, the absence of an incumbent upon termination should be interpreted as proof that (MIP) has no feasible solution with value less than Z_0.

Step 3. Different rules for selecting the next candidate problem lead to different

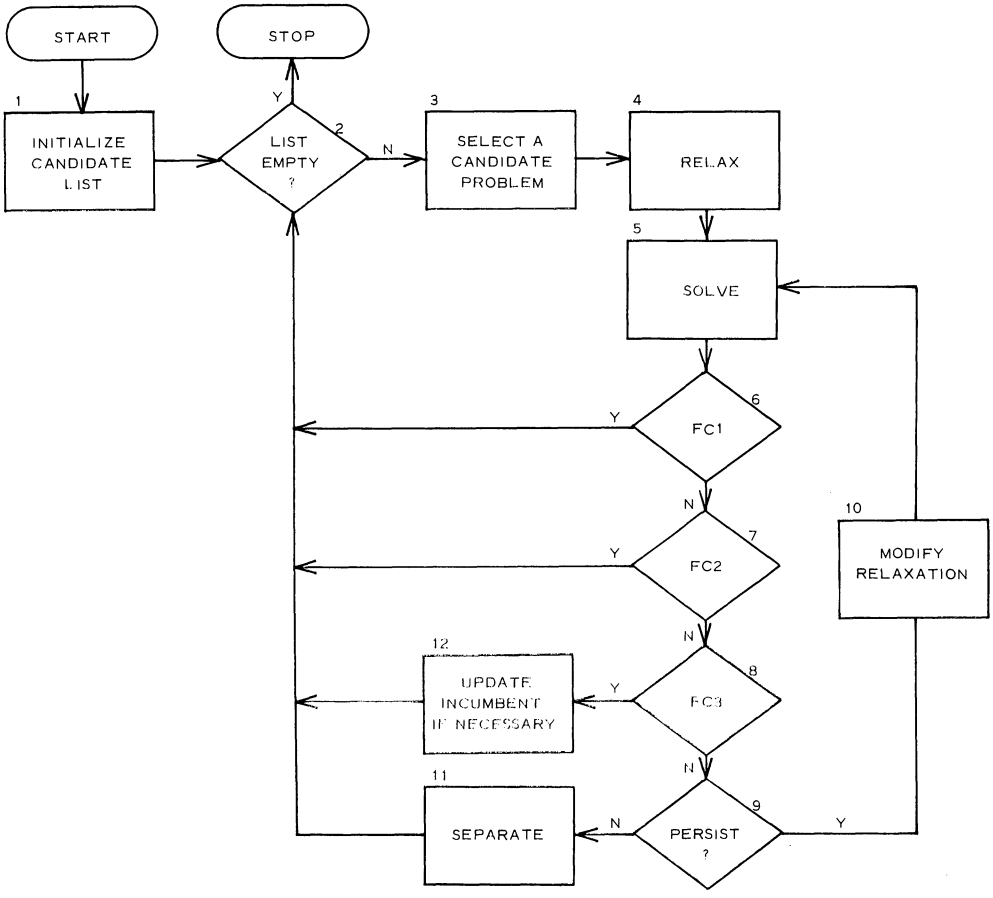

FIGURE 1. A general procedure.

patterns of enumeration. Two main types of rules have been employed, as will be illustrated in the next part of this paper: Last-In-First-Out (*LIFO*) and *Priority*. Under LIFO, the problem selected is always the last one that was added to the candidate list. The two main advantages are that the bookkeeping associated with maintaining the candidate list can be done very compactly, and reoptimization at Step 5 is greatly facilitated (see the remark on Step 5 below). The main disadvantage is that there is less flexibility to control the enumerative process; occasions do arise when some older problem in the candidate list may look more promising. Priority rules, on the other hand, achieve flexibility by selecting the problem with the best priority index, where the index for each problem may be given by (or be computed from) a tag affixed at the time of creation. The tag may be, for instance, a lower bound on the optimal value. Unfortunately, such flexibility must be purchased at the price of more cumbersome bookkeeping and lessened opportunity to reoptimize efficiently at Step 5. The most effective rules for Step 3 therefore strike a balance between the LIFO and Priority extremes.

Step 4. It is conceivable that (CP_R) is so relaxed by comparison with (CP) that it has unbounded optimal value. Since the feasible region of (MIP) was assumed bounded, however, we may guarantee without loss of generality that (CP_R) is like-

wise bounded by including adequately large bounds on the variables of (CP_R) if necessary.

Step 5. It is important not to solve each candidate problem from scratch, but rather to update earlier solutions efficiently whenever possible. Whether this is practical depends on the type of algorithm used for this step and on the choices made at Steps 3, 4, and 11. Note that it may not always be necessary to solve (CP_R) completely, since the sufficient conditions employed to implement Steps 6, 7 and 8 may be designed to operate with less information than an optimal solution of (CP_R).

Steps 6, 7, and 8. It bears emphasis that one need only apply convenient *sufficient* conditions at these steps, for to fathom with absolute certainty could be computationally onerous. The order of these steps is not at all rigid. In fact, it is best to conceptualize Steps 5 through 8 as a single unit devoted to the attempted fathoming of (CP).

Step 9. A very simple rule would be: give up after a certain number of unsuccessful attempts to fathom (CP). A more sophisticated stopping rule could be based on the apparent rate of progress toward an optimal solution of (CP).

Step 12. An additional function can be performed if an improved feasible solution of (MIP) has been found and the problems in the candidate list are tagged with lower bounds on their optimal values: purge the list of those problems whose lower bounds are not smaller than the new Z^*, for they obviously cannot contain a feasible solution better than the new incumbent.

This general procedure admits a great deal of flexibility, as will be evident from the diversity of algorithms discussed in the next section. It should also be clear that, although the scope of this paper is confined to integer linear programs, the framework given here extends readily to integer nonlinear programs and many other combinatorial optimization problems that do not have a natural or efficient mathematical programming formulation.

III. Current Algorithms

We now consider a number of recent general purpose algorithms whose computational effectiveness has been empirically demonstrated for medium-to-large problems. Each algorithm is described in terms of the general framework, with digressions as necessary to explain novel features of general interest. Computational experience is cited, most of which is as yet unpublished.

First a number of enumerative (implicit enumeration, branch-and-bound) algorithms are covered, including those of [23], [61] (OPHELIE MIXED), [14] (MPSX-MIP), [69], [70] (UMPIRE), [29] (RIP30C), [22], [58], and [43] (MARISABETH). Benders Decomposition is explained in the next section, and computational results from [2] (MIDAS-2), [19] (FMPS-MIP), and [51] (IPE) are reported. In §3.3 the cutting-plane approach is briefly discussed, including computational results from [55], [56]. We treat the group theory approach in a manner strongly influenced by [39] (IPA).

Any omissions of computationally successful general purpose codes should be interpreted as due to lack of information on our part, rather than as the exercise of personal opinion.

The reader is cautioned to interpret the numerical results quoted here with care, especially when it comes to making effectiveness comparisons between different algorithms. The figures quoted here are believed accurate, but the description of the problems run and the manner in which each code was actually operated are neces-

sarily incomplete. Taken in the aggregate, however, the results reported do give a fairly good feeling for the current state-of-the-art.

3.1. *Enumerative Algorithms*

By "enumerative" algorithms we mean to include all those which come under the popular headings of "implicit enumeration" and "branch-and-bound." Such algorithms methodically search the set of possible integer solutions in such a way that not all possibilities need be considered individually.

It seems natural to classify enumerative algorithms into two categories: those which base their fathoming tests primarily on the logical implications of the problem constraints, and those which base them primarily on associated linear programs (derived by ignoring integrality requirements). The first category of algorithms, applicable only to all-integer problems, was stimulated in this country to a great extent by Balas [4]. Work in this vein includes [28], [33], [41], [49], [74], and many more. Although there have been some notable successes in solving problems of very special structure (e.g., [26], [44], [60]), it appears that in general computing times tend to increase approximately exponentially with the number of variables. This is undoubtedly the reason why, so far as we know, no computer code for an algorithm of this type has yet proven to be of *general* practical utility.

The other category, however, has spawned a number of computer codes of general utility for mixed as well as pure integer programs. Work employing linear programming as the primary fathoming device was stimulated by the seminal contribution of Land and Doig [46]. Before discussing several recent developments in this vein, it will be useful to review the generally superior variant of the Land and Doig algorithm proposed by Dakin [20].

The following summary indicates how Dakin's algorithm for (MIP) fits into the general framework.

Algorithm of Dakin.

Step 1. Standard.

Step 2. Standard.

Step 3. LIFO rule.

Step 4. Relax all integrality requirements.

Step 5. Solve by a linear programming algorithm.

Step 6. Standard.[1]

Step 7. Standard.[1]

Step 8. Standard.[1]

Step 9. Always go to Step 11.

Step 10. Omit.

Step 11. Dichotomize the current candidate problem via the alternative constraints $x_j \leqq [\bar{x}_j]$ and $x_j \geqq [\bar{x}_j] + 1$, where $\bar{x}_j$ is a fractional-valued x-variable in the solution found at Step 5.[2] See the text for the particular rule used to select the separation variable and the order in which the new candidate problems are added to the list.

Step 12. Standard.

Step 11 is one of the main respects in which the various enumerative algorithms differ. Dakin's interval dichotomization constraints have been widely adopted, but many other ways have been suggested for selecting the variable on which to base the

[1] The "standard" option for this step is always the parenthetical one given in §2.4.

[2] $[x]$ stands for the integer part of x.

separation, and the order in which to add the new candidate problems to the list (when this matters). Dakin selected the separation variable by first estimating, for each alternative constraint of each fractional-valued variable, the amount by which the optimal value of the current linear program would increase if the constraint were introduced. The estimate used was the increase that would actually occur during the very first dual simplex method iteration, assuming that this method is used for reoptimization. The separation variable selected is the one corresponding to the largest such estimate, with the corresponding new candidate problem being placed on the list last (after its alternative).

For computational experience with this basic algorithm, see the original paper by Dakin, [15], and [73].

3.1.1. *Davis, Kendrick and Weitzman* [23] *and Penalties.* The "estimates" used at Step 11 by Dakin contain the essence of an important development commonly known as *penalties*. Widespread appreciation of the significance of penalties followed the publication of [24], which gave an independent development in the context of a quite rudimentary enumerative scheme. We prefer to describe here the work of Davis, Kendrick and Weitzman (DKW), however, since they gave one of the first systematic developments of penalties in the context of a powerful enumerative scheme.

The DKW paper addresses mixed integer programs in which all integer variables are binary. In terms of the general framework, its outline would be the same as that given above for [20] except for Steps 3, 7 and 11. Leaving aside for a moment the details of computing and using penalties, these three steps can be described in general terms as follows.

Step 3. Select the candidate problem with the smallest associated bound (see Step 11).[3]

Step 7. If $v(CP_R) + PEN(CP_R) \geqq Z^*$, go to Step 2.

Step 11. Dichotomize (CP) in the manner of Dakin. The variable selected is the one with the greatest "up penality" or "down penalty"; the associated bounds of the two descendants are $v(CP_R)$ plus the appropriate penalties.

Encouraging computational results have been obtained for a number of test problems. The largest of these, having to do with hospital scheduling, had 197 constraints, 217 continuous variables, and 100 0-1 variables; it was solved in $5\frac{1}{2}$ minutes on an IBM 7094. More recently, it is reported [27] that an investment planning model for the electric power industry with 290 constraints, 410 continuous variables, and 92 binary variables was solved in 21 minutes on an IBM 360/85.

It remains now to discuss the penalties referred to above. The key fact is that an optimal linear programming tableau contains information that enables an estimate to be readily constructed of the amount by which the optimal value would change if any one of the variables were forced to assume a particular integer value. The situation can be visualized in terms of the following figure, which depicts the optimal value of a minimizing linear program—say the one solved at Step 5—as a function, $V_j(x_j)$, of the value of x_j considered as a parameter. It is well known and not difficult to show that $V_j(x_j)$ is a convex piecewise-linear function attaining its minimum at $\bar{x}_j$, the optimal value of x_j in the linear program. (Although this possibility is not illustrated, it is possible for the linear program to become infeasible for x_j too large

[3] DKW also suggested the following modification: if no feasible integer solution has yet been found and separation has just taken place (rather than fathoming), limit the selection to the two problems just created (choose the one with the smallest associated bound).

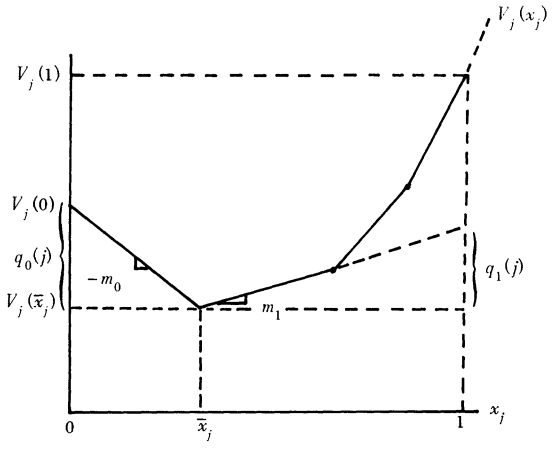

FIGURE 2

or too small; in this case $V_j(x_j)$ could be considered as having infinite value.) Designate by m_1 $(-m_0)$ the right-hand (left-hand) derivative or slope of V_j at the point $\bar{x}_j$. These derivatives can usually be computed directly from the optimal tableau, and enable estimation of $V_j(0)$ and $V_j(1)$ by linear extrapolation. The resulting estimate of $V_j(0)$ is $V_j(\bar{x}_j) + m_0\bar{x}_j$, which happens to equal the true value in this case. The estimate of $V_j(1)$ is $V_j(\bar{x}_j) + m_1(1 - \bar{x}_j)$, which undershoots the true value. In general, by convexity these estimates can never exceed the true values. The quantity $m_0\bar{x}_j$ will be denoted by $q_0(j)$ and called the *simple down penalty* associated with x_j; $m_1(1 - \bar{x}_j)$ will be denoted by $q_1(j)$, the *simple up penalty*.

Clearly $q_u(j) \geqq 0$ and $V_j(u) \geqq V_j(\bar{x}_j) + q_u(j)$ for all j and $u = 0$ or 1. Furthermore, since V_j must increase by at least Min $\{q_0(j), q_1(j)\}$ if x_j is to be an integer, the optimal value of the linear program must increase by at least

$$PEN \triangleq \text{Max}_j \{\text{Min } \{q_0(j), q_1(j)\}\} \geqq 0$$

for any solution that is all integer.

This analysis directly yields the improved version of Step 7 given above for DKW, since

$$v(CP) \geqq v(CP_R) + PEN(CP_R) \geqq v(CP_R)$$

(the new notation makes explicit the dependence of PEN on (CP_R)).

Before discussing the use of penalties at Step 11, we mention one rather obvious further use of penalties at Step 7. Even if $v(CP_R) + PEN(CP_R) < Z^*$, it is still possible to have $v(CP_R) + q_{\hat{u}}(\hat{j}) \geqq Z^*$ for some $\hat{j}$ and $\hat{u} = 0$ or 1. In this case the variable $x_{\hat{j}}$ can be fixed at the value $1 - \hat{u}$ in the current candidate problem and in any of its descendants.

At Step 11 one computes $\text{Max}_j \{\text{Max } \{q_0(j), q_1(j)\}\}$. If $q_{\hat{u}}(r)$ achieves this maximum, then x_r is chosen to be the separation variable. A lower bound on the optimal value of the descendant associated with $x_r = \hat{u}$ is $v(CP_R) + q_{\hat{u}}(r)$, while one associated with the other descendant is $v(CP_R) + PEN(CP_R)$ [note that while

$$v(CP_R) + q_{1-\hat{u}}(r)$$

TABLE 1
Computational Results with OPHELIE MIXED

Binary Var.	12	99	114	150	20	48	54	93	117	50	55	81	69	240
Contin. Var.	4,146	839	0	611	2,595	947	252	20	1,970	0	284	1,074	4,858	102
Rows	2,042	1,065	60	566	423	457	279	63	2,164	6	205	264	1,574	397
Nonzero Entries	31,137	5,220	664	2,867	9,088	8,148	774		13,860				30,438	2,616
Application					WH.LOC.	RFNRY.	DISTR.		INV.PLN.			PROD.	DISTR.	CHEM.INV.
Init. LP (Min.)														
CP	660.	1.4	0.01	0.4									141.7	t
IO	1,800.	3.3	0.07	1.2										
SS									t					
Total Time (Min.)														
CP	735.	2.6	0.3	4.0				4.0		0.3			901.7	$4.6+t$
IO	2,010.	9.6	2.0	19.3				25.0						
SS					7.0	41.8	3.3	15.5	$23.2+t$	0.8	3-4	3.3		
Solution Tol. %	0	0	0	0	0	11%	0	0	0.7%	0	0	<1%	0	0
Reference	← Edney [25] →				← Blondeau [17] →					← Sommer [67] →				

is also a valid lower bound, the given bound is superior since

$$q_{1-\hat{u}}(r) = \text{Min}\,\{q_0(r), q_1(r)\} \leqq PEN(CP_R)].$$

3.1.2. *OPHELIE MIXED (Roy, Benayoun and Tergny* [61]). The algorithm described in [61] is quite close to the DKW algorithm just described. The only discernible differences are in the elaborate heuristics used at Steps 3 and 11.

Step 3 mainly uses the priority rule of selecting the candidate problem with the most promising bound, but secondary criteria are also employed to take into consideration the probable ease with which a given candidate problem could be reoptimized and the plausibility of the associated bounds (e.g., the more variables fixed the better the likely quality of the bound).

Step 11 is done in the manner of DKW, except that the choice of separation variable may be limited to a set determined by a "hierarchy graph" reflecting the analyst's opinions concerning the relative importance of different variables.

The algorithm has been commercially implemented for the CDC 6600 as OPHELIE MIXED by SEMA using the OPHELIE II linear programming system for Step 5. Table 1 quotes some unpublished results obtained from private sources. In interpreting the computing times, all on the CDC 6600, it should be kept in mind that customer billing is done on the basis of "system seconds" (SS), which is central processing (CP) seconds plus {IO seconds $\times$ fraction of central memory used}. System seconds is therefore the best measure of efficiency. See also the extensive published computational results in [61].

3.1.3. *MPSX-MIP (Bénichou et al.* [14]). The algorithm employed in [14] coincides with that of Dakin in its essentials except for Steps 3 and 11. Step 3 employs a compromise LIFO/Priority strategy, while Step 11 makes use of a preconceived priority order for selecting the separation variable. More specifically, the version of these steps used in most of the computations is as follows.

Step 3. Select a problem from among those designated "Class 2" if possible; otherwise, from among those designated "Class 1"; if both of these classes are empty, select from the remaining unclassified problems. When selecting from Class 2, the choice is arbitrary; when from Class 1, select the problem with the best "estimation" (see below), redesignate both this problem and its alternative (the one created at the same execution of Step 11—it has the same estimation) as Class 2, and cancel any remaining Class 1 designations; in selecting from the problems with no class designation, choose the problem with the best estimation and designate both this problem and its alternative as Class 2.

Step 11. Dichotomize (CP) in the manner of Dakin. Choose the separation variable from among those with a fractional part of between 0.1 and 0.9 unless this subset is empty. In either case, select according to a priority order which ranks variables in decreasing order of their absolute cost coefficients. Designate the two descendants as Class 1, and associate with each the estimation associated with the candidate problem being dichotomized.

The "estimation" referred to is a heuristic formula that estimates $v(CP)$ by

$$v(CP_R) + \sum_j \text{Min}\,\{PCL_j(\bar{x}_j - [\bar{x}_j]), PCU_j([\bar{x}_j] + 1 - \bar{x}_j)\},$$

where the sum is over the fractional-valued variables and PCL_j (PCU_j) is a lower (upper) "pseudocost" for variable j. Pseudocosts are quite similar in spirit to penalties, but are empirically determined rather than being derived from the optimal tableau like $q_u(j)$. It appears that they are taken to be the most recently observed

actual rate of increase in the optimal value of a relaxed candidate problem due to forcing a fractional variable to 0 or 1.

Bénichou et al. have suggested a number of variants and embellishments on the above version of their algorithm, such as using criteria other than "estimations" at Step 3 and alternative priority orders at Step 11. Space does not permit a detailed description here.

A commercial implementation has been carried out by IBM France based on IBM's Extended Mathematical Programming System. A number of quite large problems have been solved. One production scheduling problem with 721 constraints, 39 binary variables and 1117 continuous variables was solved in 18.3 minutes on a 360/65 once the first LP solution was obtained. Another with 157 constraints and 78 binary variables was solved in 6.07 minutes (not including first LP) on an IBM 370/155. The largest problem reported in terms of integer variables had 590 of them with 69 constraints, and was solved in 25.7 minutes (not including first LP) on a 360/65.

3.1.4. *UMPIRE and Improved Penalty Bounds (Tomlin,* [69], [70]). Tomlin [69] (see also [70]) proposed two ways of improving the simple penalty bounds discussed previously in §3.1.1, and has tested these improvements in an implementation of the algorithm of Beale and Small [11] for general mixed integer programs. The latter algorithm will be reviewed below.

Tomlin's first method for obtaining improved bounds is based on the following simple observation: forcing a fractional variable in a linear programming solution to an integer value requires increasing at least one nonbasic variable above zero, but any such increase must be by *at least unity* if this nonbasic variable is specified as integer in the original problem statement. The latter condition is ignored in the earlier derivation of simple penalties. Tomlin showed how this requirement could be used to yield stronger up and down penalty bounds.

The smaller of the new up and down penalties associated with a given fractional variable is a lower bound on the increase in optimal value caused by forcing that variable to be integer (cf., Min $\{q_0(j), q_1(j)\}$ in §3.1.1). Tomlin has shown, however, that a Gomory cut associated with the same fractional variable yields a still better bound than the smaller of the improved up and down penalties. Such a cut is generated in the usual way, and a linear estimate of the increase in optimal value caused by adding it to the linear program is constructed from information available in the final tableau in the same manner by which one obtains $q_u(j)$. The resulting bound might be called the Gomory-penalty $P_G(j)$ associated with the given fractional variable (*cf.*, p. 907 of [49]).

These improved penalties can be used in the obvious way to improve any algorithm that employs simple penalties. Tomlin chose to incorporate them in the algorithm described in [11], which is the same as that of Dakin described previously except for Steps 7 and 11. These two steps happen also to be the ones where penalties are used. They become as follows when Tomlin's more powerful penalties are used in place of the simple penalties employed by Beale and Small.

Step 7. If $v(CP_R) + \text{Max}_j \{P_G(j)\} \geqq Z^*$, go to Step 2.

Step 11. Dichotomize (CP) in the manner of Dakin. Choose for the separation variable the one with the largest improved up or down penalty, put the corresponding descendant next on the candidate list, and finally place its alternative at the very end of the list.

It is reported in [69] that using the more powerful penalties in place of simple penalties can reduce the work to solve all-integer or predominantly integer problems by

50% or more. For a capital budgeting problem with 50 binary variables and 5 constraints, for example, the solution time on a Univac 1108 was reduced from 94 sec. to 44 sec.

Additional computational experience with these improved penalties is cited in [70]. The algorithm used was essentially the same as the one above except that a different type of separation was employed at Step 11 for variables appearing in multiple-choice type constraints (i.e., constraints of the form $\sum_{j \in J} x_j = 1$). The type of separation used for these variables, first described in [12], is of the form: $x_j = 0$, all $j \in J_1$ or $x_j = 0$, all $j \in J - J_1$, where J_1 is some subset of J. Penalties appropriate to this type of dichotomization are derived. The implementation, within the UMPIRE system, was done by Scientific Control Systems Ltd. for the Univac 1108. Computational experience with a number of quite large problems is reported. One had about 250 rows, 200 continuous variables and 164 binary variables; another had about 200 rows, 700 continuous variables and 40 binary variables; and two others had about 1000 constraints and 200 binary variables. Although a strict optimal solution was not obtained for any of these problems, satisfactory integer solutions are claimed. At the very least, it can be concluded that the modifications made to the original Beale and Small algorithm—improved penalties and a special facility for handling multiple-choice variables—greatly improve the performance.

3.1.5. *RIP30C and Surrogate Constraints* (*Geoffrion* [29]). All of the above algorithms use linear programming as the main fathoming device, and make essentially no use of the logical fathoming devices employed by the additive algorithm of [4] and others of this school mentioned in the introduction to §3.1. The main limitation of these fathoming devices is that they apply at reasonable computational cost to only one constraint at a time. The joint logical implications of several constraints taken simultaneously are therefore lost, with the apparent result that computation times tend to increase approximately exponentially with the number of integer variables for methods relying only on logical fathoming devices. In [29] it is shown how a "surrogate" constraint (see Glover [33]) can be constructed which captures more of the joint logical implications of the entire set of original constraints. It was demonstrated that the dual variables in the solution of a candidate problem as a linear program yield a surrogate constraint that is as "strong" as possible in an appropriate sense. This constraint is constructed by taking a nonnegative linear combination of the original constraints using the values of the dual variables as weights, and then adding to it the constraint that the objective function must have a better value than that of the incumbent. A closely related construct was developed independently in [5].

A computer code (RIP30C) was built to determine how much improvement, if any, such strongest surrogate constraints would make in Balas' additive algorithm as organized in [28]. The search strategy of that algorithm is LIFO at Step 3 and uses a "least total infeasibility after branching" rule at Step 11 to select the separation variable. The standard logical fathoming strategy for Steps 4–10, which will not be reviewed here, was supplemented with a linear programming subroutine that could be used optionally to generate strongest surrogate constraints. Tests on a wide variety of all 0-1 problems showed decisively that the modification leads to a great improvement in performance. Of even more significance than the large observed reductions in computing times for specific problems, perhaps, was the observed behavior of computing times as a function of problem size. Sequences were run of set-covering, optimal routing, and capital budgeting problems of varying sizes up to 90 0-1 variables. The modification appeared to mitigate solution-time dependence on the number of vari-

ables from an exponential to a low-order polynomial increase. The dependence was approximately linear for the first two problem classes, with 90-variable problems typically being solved in about 15 seconds on an IBM 7044; and approximately cubic for the third class, with 80-variable problems typically solved in under 2 minutes.

The point is sometimes raised that since linear programming is used to generate strongest surrogate constraints, it is possible to attribute the improved efficiency to the fathoming power of the linear program itself rather than to the power of surrogate constraints. Some light can be thrown on this issue by running the RIP30C code with the linear programming routine fully operative but with no memory of surrogate constraints from one candidate problem to the next. For a representative battery of 8 test problems in the range of 44 to 90 variables, this increased the running time per problem by an average of 100% (the median was 42%). This is if anything a conservative rather than true indication of the value of surrogate constraints, however, because the standard LP fathoming tests at Steps 6 and 7—which remained operational—are really equivalent to the construction of a binary infeasible surrogate constraint, a task that could often be done by means other than by linear programming. On the other hand, it must be admitted that any naturally integer linear programming solutions at Step 8 have nothing to do with surrogate constraints.

The RIP30C code referred to above has subsequently undergone extensive modification. Among the new features used routinely are the following: a dual linear programming subroutine which resolves degeneracies more efficiently and permits column-generation and reduced storage requirements for large problems; penalties of the type discussed in §3.1.4 for fathoming and for selecting a separation variable; and means for "purging" the candidate list as suggested in the remark on Step 12 in §2.4. The performance of the code is perhaps best indicated by the ratio of total time required to solve an integer program to the elapsed time upon solution of the initial linear program. For an eclectic set of 8 realistic test problems in the size range of 80–131 binary variables, this ratio currently ranges from 1.05 to 3.1, with a mean of 1.67 and a median of 1.5. The computing times themselves range from 0.95 to 7.5 secs. on an IBM 360/91. All runs were made using the standard default options of the code, with no exploitation of special structure.

3.1.6. *Other Enumerative Algorithms.* Davis [22] has implemented an algorithm which, like [29], employs both logical tests and linear programming for fathoming under a primarily LIFO search strategy. In contrast to the latter algorithm, however, he relies mostly on simple penalties rather than on strongest surrogate constraints to enhance the power of the standard fathoming tests. Davis also introduced an innovation aimed at obtaining an earlier proof of termination or, failing that, an indication that certain variables may be essentially dropped for the remainder of the search because they must assume particular values. This is done by periodically solving a linear program composed of (MIP) with the integrality requirements dropped, augmented by linear constraints which specify that only those solutions need be considered which: (a) improve on the value of the current incumbent, and (b) do not duplicate any solutions already accounted for by fathoming. An implementation for the Univac 1108 has been extensively tested. To mention just the larger problems: two project selection problems with 105 binary variables and 2 constraints were solved in 5 and $6\frac{1}{2}$ seconds; a project selection problem with 50 binary variables and 5 constraints was solved in $10\frac{1}{2}$ seconds; two machine scheduling problems with 131 binary variables and 40 constraints were solved in 15 and 20 seconds; and an investment planning problem with 11 binary variables, 288 continuous variables and 99 constraints was solved in 100 seconds.

Mitra [58] has done comparative studies of the relative effectiveness of a number of variants of Dakin's basic plan. The variants consist of five alternative options for Step 3 and six alternative options for the choice of separation variable at Step 11. Presumably Step 7 uses some kind of penalties, although this is not spelled out. Computational experience is cited with 9 practical problems, many of them very large (thousands of constraints and hundreds of 0-1 variables). However, no problem with more than 40 0-1 variables was solved to optimality. This makes it difficult to draw any firm conclusions from the 17 reported runs. The reader should consult the original paper, as an adequate summary would not be possible in this limited space.

MARISABETH is a mixed integer code built by Shell Berre-Paris for the Univac 1108. The algorithm is of the branch-and-bound type, but details have not been made public. The implementation is based on the Marie-Claire linear programming code of Univac France. Performance data [43] are given in Table 2. In this table, NR stands for the number of rows, NC (NB) for the number of continuous (0-1) variables, $T1$ for the time to solve the initial linear program, $T2$ for the total computing time, and NCP for the number of candidate problems examined at Step 5. Problem 1 is concerned with oil refinery planning, problems 2–5 with multi-period investments, problems 6–8 with scheduling oil product deliveries by rail, and problems 9 and 10 with coastal shipping of oil products.

3.2. *Benders Decomposition*

Benders' approach [13] to mixed integer programming rests on an equivalent version of (MIP) in terms of its integer variables alone. The equivalent version is obtained in three stages. The first stage is to "project" (MIP) onto the space of its x-variables as follows:

(MIP-1) $\text{Minimize}_{x \geqq 0,\ \text{integer}} [cx + \text{Infimum}_{y \geqq 0} \{dy \mid Dy \leqq b - Cx\}]$.

Notice that evaluating the objective function requires solving the linear programming subproblem

(SUB) $\text{Minimize}_{y \geqq 0}\ dy \quad \text{subject to} \quad Dy \leqq b - Cx.$

The second stage is to dualize (SUB), so that (MIP-1) becomes

(MIP-2) $\text{Minimize}_{x \geqq 0,\ \text{integer}} [cx + \text{Supremum}_{u \geqq 0} \{u(Cx - b) \mid d + uD \geqq 0\}]$.

TABLE 2

Computational Results with MARISABETH

Problem	NR	NC	NB	$T1$	$T2$	NCP
1	156	98	25	22 sec.	3 min. 43 sec.	195
2	671	803	48	7 min. 9 sec.	14 min. 55 sec.	121
3	146	317	18	2 min. 8 sec.	4 min. 38 sec.	61
4	280	252	54	32 sec.	17 min. 52 sec.	791
5	355	769	14	4 min. 13 sec.	18 min. 12 sec.	67
6*	885	260	584	6 min. 17 sec.	68 min.	1161
7*	1254	254	839	17 min.	43 min.	889
8*	762	174	638	11 min.	75 min.	344
9*	1581	485	282	1 min. 32 sec.	17 min.	107
10*	2011	702	294	1 min. 23 sec.	2 min. 20 sec.	39

* The solutions of these problems are suboptimal, to an unspecified degree of approximation.

In terms of the extreme points $\{u^1, \cdots, u^p\}$ and extreme rays $\{u^{p+1}, \cdots, u^{p+q}\}$ of the feasible region of the dual of (SUB), this problem can be written

$$\text{(MIP-3)} \qquad \begin{aligned} &\text{Minimize}_{x \geqq 0,\ \text{integer}} \left[cx + \text{Max}\ \{u^1(Cx - b), \cdots, u^p(Cx - b)\}\right] \\ &\text{subject to} \quad u^j(Cx - b) \leqq 0, \qquad j = p + 1, \cdots, p + q. \end{aligned}$$

The third and final stage is to linearize (MIP-3) by expressing the maximum in the objective function as a least upper bound:

$$\text{(MIP-B)} \qquad \begin{aligned} &\text{Minimize}_{x \geqq 0,\ \text{integer};\ x_0} \ cx + x_0 \\ &\text{subject to} \quad x_0 \geqq u^j(Cx - b), \qquad j = 1, \cdots, p, \\ &\qquad\qquad\ \ u^j(Cx - b) \leqq 0, \qquad j = p + 1, \cdots, p + q. \end{aligned}$$

See the original paper by Benders or [8] or [30], [31] for further details. The equivalence between (MIP) and (MIP-B) which justifies solving one problem via the other is this: if (x_0^*, x^*) is optimal in (MIP-B), then x^* is also optimal in (MIP) in the sense that solving (SUB) with $x = x^*$ will yield y^* such that (x^*, y^*) is optimal in (MIP).

The advantage of (MIP-B) over (MIP) is that it reduces the number of continuous variables to one (x_0); but the disadvantage is that it has many constraints that are only implicitly known. No one would seriously propose computing $u^1, \cdots, u^{p+q}$ explicitly before attempting to solve (MIP-B). Consequently, this problem is an excellent candidate for relaxation by omitting most of the constraints and generating them only as needed. This can be done within our general framework by specializing it as follows (read (MIP-B) for (MIP) at Steps 1, 2 and 12).

Step 1. Standard.

Step 2. Standard.

Step 3. A formality (this step will be executed only once).

Step 4. Omit all of the $p + q$ constraints involving u (bound x and x_0 if necessary during the early iterations—see the remark on Step 4 in §2.4).

Step 5. Solve (CP_R) optimally.

Step 6. Omit.[4]

Step 7. Standard.

Step 8. Test the feasibility of the optimal solution $(\hat{x}, \hat{x}_0)$ just obtained for (CP_R) in (MIP-B) by solving (SUB) with $x = \hat{x}$. Denote the optimal value by $v(\hat{x})$. There are three possible outcomes:

(i) (SUB) is infeasible. Fathoming Criterion 3 is not satisfied, but the corresponding dual solution generates a violated constraint of (MIP-B). Go to Step 9.

(ii) (SUB) is feasible and $v(\hat{x}) > \hat{x}_0$. Fathoming Criterion 3 is not satisfied, but the corresponding dual solution generates the most violated constraint of (MIP-B). Furthermore, a feasible solution of (MIP-B) is $(\hat{x}, x_0 = v(\hat{x}))$; if the value of this solution is less than Z^*, make it the new incumbent and reset Z^*. Go to Step 9.

(iii) (SUB) is feasible and $v(\hat{x}) \leqq \hat{x}_0$. Fathoming Criterion 3 is satisfied; make $(\hat{x}, \hat{x}_0)$ the new incumbent and go to Step 12.

Step 9. Always go to Step 10.

[4] Assuming that (MIP) has a finite optimal solution, $F(CP_R) \neq \varnothing$.

Step 10. Tighten the relaxation by adding the violated constraint just determined.
Step 11. Omit.
Step 12. Standard.

This approach, which is obviously finitely convergent, has the advantage of providing both lower and upper bounds on the optimal objective function value. A lower bound is provided each time Step 5 is executed, and an upper bound each time outcome (ii) obtains at Step 8. The two bounds approach one another as the calculations proceed, and coincide at termination.

Recent computational experience is cited by [2], [19], [45] and [51]. Aldrich's code, MIDAS-2, actually solves a somewhat weaker approximate version of the relaxed problem at Step 5. This diminishes the computational effectiveness of the code, which is implemented within IBM's Mathematical Programming System for the 360/65. MPS is used to solve the linear program at Step 8, and the integer program at Step 5 is solved by an "enhanced" version of Balas' additive algorithm [4]. The largest problem solved was of the investment planning type and had 378 constraints, 1326 continuous variables and 24 integer variables; it was solved in 4 iterations (executions of Step 5—no time given). Experience is also cited for a warehouse location problem with 39 constraints, 53 continuous variables and 14 binary variables. It was solved in 4.46 minutes and 12 iterations.

Childress cites experience with the mixed integer programming option available with Bonner and Moore's Functional Mathematical Programming System. Again a Balas-class algorithm was used for Step 5. The following two refinery design models were the largest for which computational experience was reported. The first model had 288 constraints, 390 continuous variables, and 56 binary variables, and was solved to within $\frac{1}{3}$ of 1% of optimum in 18.13 minutes on a Univac 1108. The second model, with 515 constraints, 851 continuous variables and 48 binary variables, took 19.17 minutes to within 1.3% of optimum.

Inman and Manne have undertaken an implementation for problems in which all of the integer variables are binary and subject to multiple choice type constraints. They are specifically interested in project evaluation problems for economic or industrial development, where the objective is to decide which of a number of possible combinations of projects to undertake. The multiple choice structure makes it possible, for problems of moderate size, to carry out the optimization at Step 5 by complete enumeration. Step 8 was accomplished using IBM's Mathematical Programming System. Computational experience is cited for one problem that dealt with plant size, time-phasing, and process choice for the steel, oil and electricity sectors of Mexico; there were 350 continuous variables, 27 binary variables, and 275 constraints. Two versions of the problem were solved on an IBM 360/67. After an initial simplex solution, one took 10 iterations and 5.30 minutes to reach optimality, and the other 25 iterations and 10.50 minutes. Another problem dealt with the location of electricity generating plants and transmission lines in Mexico; there were 622 continuous variables, 22 binary variables, and 531 constraints. After an initial simplex solution, this took 16 iterations and 28.64 minutes. Still another facility location problem had 98 continuous variables, 22 binary variables and 40 constraints. After an initial simplex solution, it took 18 iterations and 5.22 minutes.

Some earlier computational results by B. R. Buzby for a class of nonlinear distribution problems are given in [47]. Very recent results for a large multicommodity distribution problem are given in [32], but are not quoted here because the implementation is so highly specialized.

A possibly useful variation of the classical Benders Decomposition approach arises if one also relaxes the integrality requirements at Step 4. The advantage of doing this is that Step 5 can then be done by ordinary linear programming, at the cost, however, of adding additional machinery to cope with the need for integrality. Cutting-planes and enumeration are the two main choices for this extra machinery. If enumeration is selected (as in [49] and [2]), then the following alterations are necessary in the previous outline of Benders Decomposition: Step 3 is no longer a mere formality, as it will be executed more than once; Step 4 should drop all integrality requirements on x, but should retain those of the $p + q$ constraints which have so far been explicitly generated at Step 8; Step 6 cannot be omitted; Step 8 should check $\hat{x}$ for integrality first (there is no need to solve (SUB) if $\hat{x}$ is not integer); Step 9 should persist to Step 10 if and only if a violated constraint of (MIP-B) was just generated at Step 8; and Step 11 cannot be omitted. These changes also suffice to handle the case where some but not all of the integrality requirements on x are dropped at Step 4—except, of course, that Step 5 then requires a mixed integer rather than ordinary linear programming routine.

To sum up this variant of Benders Decomposition in a few words, it could correctly be called an enumerative algorithm for (MIP-B) with the $p + q$ constraints relaxed at Step 4 and generated as needed at Step 8.

3.3. *Cutting-Plane Algorithms*

Historically, this was the first general approach taken to solving integer programs. The foundations were laid by Gomory in a series of well-known papers, and there is now an extensive literature on the subject [9]. The classical pattern for cutting-plane algorithms can be described in terms of the general framework as follows.

Step 1. Standard.
Step 2. Standard.
Step 3. A formality (this step will be executed only once).
Step 4. Relax all integrality requirements.
Step 5. Solve by a linear programming algorithm.
Step 6. Standard.
Step 7. Omit.
Step 8. Standard.
Step 9. Always go to Step 10.
Step 10. Tighten the current relaxation by adding a cutting-plane.
Step 11. Never occurs.
Step 12. Standard.

Notice that the approach is based entirely on successively improved relaxations, with no use whatever made of separation. Thus termination occurs the first time that either Fathoming Criterion 1 or 3 is satisfied. It follows that the cutting-plane approach has the disadvantage of not ordinarily providing a feasible solution to the original problem until termination.

The "cutting-planes" referred to at Step 10 are, of course, linear constraints. Thus $(CP_R{}^k)$, the linear program to be solved at the kth execution of Step 5, consists of the original integer linear program with the integrality requirements dropped and $k - 1$ new linear constraints added. One infers directly from the general outline given above that the new "cuts" must be such that

$$F(CP_R{}^1) \supset F(CP_R{}^2) \supset \cdots \supset F(CP_R{}^k) \supseteq F(CP). \tag{3}$$

That is, each new cut must properly tighten the previous relaxation, and yet still yield

a valid relaxation of the (one and only) candidate problem itself. In other words, each new cut must lop off some of the feasible region of the current linear program without also lopping off any feasible integer solutions of the original integer program. It follows from (3) that

$$v(CP_R^1) \leqq v(CP_R^2) \leqq \cdots \leqq v(CP_R^k) \leqq v(CP).$$

There are many ways of deriving cuts satisfying (3). A few recent references are [7], [18], [35], [37], [77]. To be useful, however, the cuts should have certain additional desirable properties. For instance, it is customary for each new cut to cut away the solution just found at Step 5, for otherwise that solution would still be valid at the next execution of Step 5. Another desirable property is that at most a finite number of cuts will be necessary in order to find an optimal solution of the given problem or discover that none exists. Still more desirable is the property that the cuts actually are faces of the smallest convex polytope containing $F(CP)$, namely the convex hull of the feasible integer solutions of the original problem. This elusive object is the ultimate objective of considerable current research, e.g., [38].

Almost all of the computational experience with cutting-plane algorithms reported in the literature has been confined to very small problems. See [9], [72] and [57] for reviews and evaluations of these efforts. A conspicuous exception is the work of Glenn Martin at Control Data Corporation [53], [54], [55], who has had outstanding success with certain types of problems having a strong tendency toward natural integrality and basic determinants of moderate size (in the associated continuous LP problem). Problems relating to networks, the traveling salesman problem, and generalized set covering problems tend to fall in this category. By far the most numerous and successful applications in Martin's experience have been crew scheduling problems, many hundreds of which he has successfully solved during the past 8 years [56]. He reports that a typical "moderate" sized crew scheduling problem of about 100 inequality constraints and 4,000 binary variables takes on the order of 10 minutes to solve on a CDC 3600. A typical larger problem of 150 rows and 7,000 binary variables can often be solved with a few cuts in on the order of 40 minutes of CDC 3600 time.

3.4. *Group Theoretic Algorithms*

The group theoretic approach was initiated by Gomory [36] and has been carried forward primarily by Shapiro [63], [64], [65]. Other recent contributions have been [34], [68], [75], and [76]. The present discussion follows Gorry and Shapiro [39].

The group theoretic approach has been applied almost exclusively to the pure integer programming problem; that is, to (MIP) without any continuous variables (y). Writing $A = [C \mid I]$ to introduce slack variables, we have

$$\text{(4)} \qquad \text{Minimize } cx \quad \text{subject to } Ax = b, \quad x \geqq 0 \text{ and integer.}$$

The first step in the group theoretic approach is to transform this into an equivalent form. Let B be any dual feasible basis for (4) regarded as a linear program, and reorder the variables if necessary so that $A = [B \mid R]$. Then (4) is equivalent to the problem

$$\text{(4A)} \qquad \text{Minimize } \bar{c}x_R \quad \text{subject to } x_B = \bar{b} - \bar{A}x_R \geqq 0 \text{ and integer}, \quad x_R \geqq 0 \text{ and integer},$$

where x_B and x_R denote the basic and nonbasic variables, respectively, and

$$\begin{aligned} \bar{b} &= B^{-1}b, \\ \bar{A} &= B^{-1}R, \\ \bar{c} &= c_R - c_B B^{-1}R \geqq 0. \end{aligned}$$

Problem (4A) is referred to as a *correction problem*, since the object is to find the cheapest nonnegative integer correction x_R which renders x_B nonnegative and integer. If B is an optimal basis, then x_B is already nonnegative when $x_R = 0$.

The major relaxation employed is to drop the $x_B \geqq 0$ conditions. The basic variables, though still required to be integer, are thus allowed to become negative. This relaxation of (4A) can then be shown to be equivalent to the following group problem:

$$\text{(GP)} \qquad \begin{aligned} &\text{Minimize } \bar{c}x_R \\ &\text{subject to } A^*x_R = b^* \pmod{D}, \qquad x_R \geqq 0 \quad \text{and integer,} \end{aligned}$$

where

$$\begin{aligned} D &= |\det B|, \\ A^* &= D\{\bar{A} - [\bar{A}]\}, \\ b^* &= D\{\bar{b} - [\bar{b}]\}, \end{aligned}$$

and the square brackets denote integer parts. Thus $[\bar{A}]$ is a matrix each of whose entries is the largest integer not greater than the corresponding entry in $\bar{A}$, and similarly for $[\bar{b}]$. Problem (GP) can be interpreted as an optimization over the finite Abelian group G generated by the columns of A^*, and can be solved by a special shortest path algorithm.

The group problem (GP) can be used to fathom candidate problems in exactly the same way linear programs are used in the algorithms of §3.1. The method of creating candidate problems used in [39] is considerably different from any discussed so far, however, and so will be described briefly. For simplicity, assume that the integer variables have upper bounds greater than one. Let $\tilde{x}_R = (\tilde{x}_1, \cdots, \tilde{x}_n)$ be an arbitrary nonnegative correction and define

$$j(\tilde{x}_R) = \min\{j \mid \tilde{x}_j > 0\}.^5$$

The candidate problem associated with $\tilde{x}_R$ is

$$(CP - \tilde{x}_R) \qquad \begin{aligned} &\text{Minimize } \bar{c}(\tilde{x}_R + u) \\ &\text{subject to } x_B = \bar{b} - \bar{A}(\tilde{x}_R + u) \geqq 0 \quad \text{and} \quad \text{integer,} \end{aligned}$$

$$u_j \geqq 0 \quad \text{and integer for} \quad j = 1, \cdots, j(\tilde{x}_R), \quad u_j = 0 \quad \text{for} \quad j = j(\tilde{x}_R) + 1, \cdots, n.$$

This candidate is relaxed to a group problem by dropping the $x_B \geqq 0$ condition. The list of candidate problems is separated according to "level number." Starting at $K = 0$ the corrections at level K are those corrections x_R with $\sum_{j=1}^{n} x_j = K$. If an arbitrary correction $\tilde{x}_R$ at level K cannot be fathomed, then it is replaced by the cor-

[5] If $\tilde{x}_R = 0$, then define $j(\tilde{x}_R) = n$.

rections

$$\tilde{x}_R + e_j \quad \text{for} \quad j = 1, \cdots, j(\tilde{x}_R),$$

each of which is at level $K + 1$. (Here e_j denotes the jth unit vector.) This effects a separation of $(CP - \tilde{x}_R)$ into $j(\tilde{x}_R)$ new candidate problems.

The algorithm of [39] can now be related to the general framework.

Step 1. The given problem (4) is transformed into a correction problem (4A). Any dual feasible basis can be used, but the usual procedure is to use an optimal basis. Let $\tilde{x}_R = 0$.

Step 2. Standard.

Step 3. The selection rule employed is called "directed search." Each candidate problem is assigned a "plausibility value". This is a lower bound on its minimal value, computed at the time of creation from the group problem solution. The candidate selected is the one with the lowest lower bound from among a subset (those in main storage) of the pending candidates.

Step 4. Relax the candidate problem $(CP - \tilde{x}_R)$ to a group problem by dropping the nonnegativity conditions on the basic variables.

Step 5. For the relaxation performed at Step 4, a known algorithm is available to set up a canonical representation of the group and solve the problem. For the other kinds of relaxation performed at Step 10, the appropriate algorithms are assumed to be available.

Step 6. Standard.

Step 7. Standard.

Step 8. Standard.

Step 9. Heuristic rules may be used so that "a reasonable computational effort" is made to fathom the current candidate problem.

Step 10. Several alternative types of relaxation are proposed for this step. These include Lagrangian group problems [66], cutting planes (§3.3), and surrogate constraints (§3.1.5).

Step 11. The separation technique is as described earlier. The group (or other) problem solution is used to compute lower bounds on the minimal values of the new candidate problems. These are referred to as "plausibility values."

Step 12. Standard.

Step 10 could also incorporate a variety of heuristics and adaptive devices to choose a "suitable" relaxation for each candidate problem, depending on its algebraic structure.

Some computational results are given. For example, several media selection problems with about 75 constraints and 150 integer variables were solved in under a minute on a Univac 1108.

Some efforts have been made to extend the theory to mixed integer programming [75]. The computational difficulties encountered, however, seem formidable. More promising are attempts to synthesize the group approach with Benders Decomposition which separates the pure integer and continuous parts of a mixed integer problem.

IV. Summary and Prognosis

In this final section, we would like to summarize very briefly the current state-of-the-art of integer linear programming, and offer some opinions as to promising directions for the future.

Enumerative algorithms have received the lion's share of attention in recent years,

especially as measured by new implementations. This is due partly to disillusionment with the erratic computational performance of the early cutting-plane algorithms, and partly to the publication of such influential papers as [46], [50], [4], and [24]. Recent computational experience seems to vindicate this emphasis. A number of existing codes are quite reliable in obtaining optimal solutions within a short time for general all-integer linear programs with a moderate number of variables—say on the order of 75—and well into the hundreds of variables for problems of more special structure. Mixed integer linear programs of practical origin with up to half a hundred integer variables and a thousand or two continuous variables and constraints are tractible for several modern implementations based on a production quality linear programming system. Problems of this size with several hundred integer variables have been undertaken, but usually the result has been a suboptimal feasible solution (often an acceptably good one). All of the successful general purpose codes use linear programming at Step 5, almost all of them use the penalty idea in several ways, and many use some sort of compromise between strict LIFO and Priority at Step 3 and permit several options for selecting a separation variable at Step 11.

The available computational experience with Benders-type algorithms, although not nearly so voluminous as that for enumerative algorithms, does suggest that this approach may be competitive for mixed integer linear programs. The implementations to date, however—at least the ones of which we are aware—do not achieve the full potential inherent in the approach because they use only a relatively rudimentary enumerative all-integer algorithm at Step 5. Thus the computing time spent in Step 5 is longer than necessary, and may account for the fact that only a few dozen integer variables have so far been handled successfully. The incorporation of a modern integer algorithm into Step 5 would be a programming task of considerable magnitude, but the result might well be a code capable of routinely handling one or two hundred integer variables and a few thousand continuous variables and constraints. Such success will almost certainly come to pass if the number of iterations (executions of Step 5) required to find a near optimal solution tends to remain moderate as problem size increases. Currently there is inadequate computational experience to resolve this crucial issue, but the available evidence suggests some optimism. Often only a surprisingly small number of iterations has been necessary, and the number of iterations has usually not increased in proportion to the number of logical alternatives to be accounted for, or even in proportion to the number of integer variables (see especially Table 5 in [51] and §7.3.5 of [47]).

One should also keep in mind where Benders Decomposition is concerned the portentous fact that the integer and continuous parts of the problem are treated in a way that preserves the structure of the continuous part. If the integer variables represent different configurations of a system, for example, then the integer optimization at Step 5 selects a new configuration to be tried and Step 8 optimizes the system with this configuration. The latter optimization can sometimes be done very efficiently by a special purpose algorithm. An instance of this occurs in some problems of optimal multicommodity distribution system design [32], where Step 8 breaks apart into a number of independent ordinary transportation problems (one for each commodity).

As for the cutting-plane and group theory approaches, it appears that unless significant new theoretical advances are made it may be best to marry them with an enumerative approach. Since [39] explained so clearly how this can be done with the group theory approach, we confine our remarks here to the synthesis of the cutting-

plane approach with enumeration, a topic that has thus far received much less attention than it deserves (cf., p. 897 of [49] and p. 243 of [9]).

The idea is very simple. Just take a Dakin-type enumerative algorithm, say, and equip Step 10 with the means of generating cutting-planes for the current candidate problem. A plausible rule for Step 9 would then be: for each candidate problem, persist to Step 10 up to a certain number of times in succession, or until the rate of improvement due to adding new cuts falls below some threshold value. This rule is appealing because cutting-plane algorithms typically make the greatest rate of progress during the early iterations (e.g., [71]). This means that there is a chance of actually solving within a few cuts a candidate problem unfathomable by standard means, and in any case a better lower bound on its optimal value is generated each time a cut is added—which improves the chances of activating the second fathoming criterion. This synthesis, incidentally, can be viewed as a natural extension of the use of penalties at Step 7, for penalties are analogous to estimating the change induced in the optimal value of the relaxed candidate problem by adding a single very simple cut (cf., the conclusion of §3.1.4). Obviously, adding more than one cut will yield improved estimates of the optimal value of the candidate problem.

In addition to accommodating syntheses between traditionally distinct approaches, the general framework of Figure 1 also provides a checklist of opportunities for improving any algorithm falling within its scope. It is particularly valuable to refer to this checklist when it is desired to specialize an algorithm to a particular class of problems such as fixed charge, project selection, scheduling, etc. Can a prior analysis of the problem reveal some natural preliminary separations that ought to be made, and perhaps arranged in a certain desired order for processing? If so, this prior analysis should be built into Step 1. Is there a better way of telling how "promising" a candidate problem is other than by the usual bound that may be associated with it? If so, then Step 3 can be modified accordingly. Is there a different kind of relaxation that could be employed at Step 4 or 10 which would make the resulting relaxed candidate problems easier to solve at Step 5 without unduly compromising the power of Steps 6, 7 and 8, or that would enhance the power of these steps without being too much more difficult to solve at Step 5? Does the problem structure permit improved sufficient conditions for Steps 6 and 7? Can indicators be designed to predict with some degree of reliability when persistence at Step 9 is warranted? What class of separations should be allowed at Step 11, and by what rule should members of the class be chosen, so as to most enhance the fathoming tractability of the descendants of a candidate problem?

Finally, we hazard a guess that the field of integer programming may be profoundly affected by the advent of highly parallel fourth generation computers such as the ILLIAC IV. The reason is that the ability to process several candidate problems simultaneously gives rise to a new range of possible enumerative strategies quite different from those used to date. We hope that algorithm designers will take the initiative in exploiting these new possibilities.

References

1. AGIN, N., "Optimum Seeking with Branch and Bound," *Management Science*, Vol. 13, No. 4 (December 1966), pp. B-176-185.
2. ALDRICH, D. W., "A Decomposition Approach to the Mixed Integer Problem," Doctoral Dissertation, School of Industrial Engineering, Purdue University (1969).
3. APPELGREN, L. H., "Integer Programming Methods for a Vessel Scheduling Problem," Report No. R35 (January 1970), Institute for Optimization and Systems Theory, The Royal Institute of Technology, Stockholm.

4. BALAS, E., "An Additive Algorithm for Solving Linear Programs with Zero-One Variables," *Operations Research*, Vol. 13, No. 4 (July–August 1965), pp. 517–546.
5. ——, "Discrete Programming by the Filter Method," *Operations Research*, Vol. 15, No. 5 (September–October 1967), pp. 915–957.
6. ——, "A Note on the Branch and Bound Principle," *Operations Research*, Vol. 16, No. 2 (March–April 1968), pp. 442–445.
7. ——, "Intersection Cuts from Maximal Convex Extensions of the Ball and Octahedron," MSRP No. 214 (August 1970), Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh.
8. BALINSKI, M. L., "Integer Programming: Methods, Uses, Computation," *Management Science*, Vol. 12, No. 3 (November 1965), pp. 253–313.
9. —— AND SPIELBERG, K., "Methods for Integer Programming: Algebraic, Combinatorial and Enumerative," in J. S. Aronofsky (ed.), *Progress in Operations Research*, Vol. III, Wiley, New York, 1969.
10. BEALE, E. M. L., "Survey of Integer Programming," *Operational Research Quarterly*, Vol. 16, No. 2 (June 1965), pp. 219–228.
11. —— AND SMALL, R. E., "Mixed Integer Programming by a Branch and Bound Technique," in W. A. Kalenich (ed.), *Proceedings of the IFIP Congress 1965*, Vol. 2, Spartan Press, Washington, D. C., 1965.
12. —— AND TOMLIN, J. A., "Special Facilities in a General Mathematical Programming System for Non-Convex Problems Using Ordered Sets of Variables," in J. Lawrence (ed.), *Proceedings of the Fifth International Conference of Operational Research*, Venice, Tavistock Publications, London, 1969.
13. BENDERS, J. F., "Partitioning Procedures for Solving Mixed-Variables Programming Problems," *Numerische Mathematik*, Vol. 4 (1962), pp. 238–252.
14. BÉNICHOU, M., GAUTHIER, J. M., GIRODET, P., HENTGÈS, G., RIBIÈRE, G. AND VINCENT, O., "Experiments in Mixed Integer Linear Programming," presented at the Seventh International Mathematical Programming Symposium, 1970, The Hague, Holland. To appear in *Mathematical Programming*, Vol. 1, No. 1.
15. BENNETT, J. M., COOLEY, P. C. AND EDWARDS, J., "The Performance of an Integer Programming Algorithm with Test Examples," *Australian Computer Journal*, Vol. 1, No. 3 (November 1968), pp. 182–185.
16. BERTIER, P. AND ROY, B., "Procédure de Résolution pour une Classe de Problèmes pouvant avoir un Charactère Combinatoire," *Cahiers du Centre d'Etudes de Recherche Operationnelle*, Vol. 6 (1964), pp. 202–208.
17. BLONDEAU, J. P., Private communication, May 3, 1971, Control Data Corporation, Cybernet Service Division, Houston.
18. BOWMAN, J. AND NEMHAUSER, G., "Deep Cuts in Integer Programming," Technical Report No. 8 (February 1968), Dept. of Statistics, Oregon State University, Corvallis, Oregon.
19. CHILDRESS, J. P., "Five Petrochemical Industry Applications of Mixed Integer Programming," Bonner & Moore Associates, Inc., Houston, March 1969.
20. DAKIN, R. J., "A Tree Search Algorithm for Mixed Integer Programming Problems," *Computer Journal*, Vol. 8, No. 3 (1965), pp. 250–255.
21. DAVIS, P. S. AND RAY, T. L., "A Branch-Bound Algorithm for the Capacitated Facilities Location Problem," *Naval Research Logistics Quarterly*, Vol. 16, No. 3 (September 1969), pp. 331–344.
22. DAVIS, R. E., "A Simplex-Search Algorithm for Solving Zero-One Mixed Integer Programs," Technical Report No. 5 (October 1969), Dept. of Operations Research, Stanford University.
23. ——, KENDRICK, D. A. AND WEITZMAN, M., "A Branch and Bound Algorithm for Zero-One Mixed Integer Programming Problems," Development Economic Report No. 69 (1967), Center for International Affairs, Harvard University.
24. DRIEBEEK, N. J., "An Algorithm for the Solution of Mixed Integer Programming Problems," *Management Science*, Vol. 12, No. 7 (March 1966), pp. 576–587.
25. EDNEY, M. R., Private communication, December 10, 1970, University Computing Company, London.
26. FLEISCHMANN, B., "Computational Experience with the Algorithm of Balas," *Operations Research*, Vol. 15, No. 1 (January–February 1967), pp. 153–155.
27. GATELY, D., Private communication, February 19, 1971. The model is described in "Investment Planning for the Electric Power Industry: An Integer Programming Approach,"

Research Report 7035, November 1970, Dept. of Economics, The University of Western Ontario, London, Canada.

28. GEOFFRION, A. M., "Integer Programming by Implicit Enumeration and Balas' Method," *SIAM Review*, Vol. 9, No. 2 (April 1967), pp. 178–190.
29. ——, "An Improved Implicit Enumeration Approach for Integer Programming," *Operations Research*, Vol. 17, No. 3 (May–June 1969), pp. 437–454.
30. ——, "Elements of Large-Scale Mathematical Programming," *Management Science*, Vol. 16, No. 11 (July 1970), pp. 652–691.
31. ——, "Generalized Benders Decomposition," Working Paper No. 159 (April 1970), Western Management Science Institute, UCLA. To appear in *Journal of Optimization Theory and Application*.
32. —— AND GRAVES, G. W., "Multicommodity Distribution System Design by Benders Decomposition," Western Management Science Institute, UCLA (forthcoming 1971).
33. GLOVER, F., "A Multiphase-Dual Algorithm for the Zero-One Integer Programming Problem," *Operations Research*, Vol. 13, No. 6 (November–December 1965), pp. 879–919.
34. ——, "Integer Programming over a Finite Additive Group," *SIAM J. Control*, Vol. 7, No. 2 (May 1969), pp. 213–231.
35. ——, "Convexity Cuts," Working Paper (December 1969), School of Business, University of Texas.
36. GOMORY, R. E., "On the Relation between Integer and Non-Integer Solutions to Linear Programs," *Proc. Nat. Acad. Sci.*, Vol. 53 (1965), pp. 260–265.
37. ——, "Some Polyhedra Related to Combinatorial Problems," *Journal of Linear Algebra and Applications*, Vol. 2, No. 4 (October 1969), pp. 451–558.
38. —— AND JOHNSON, E. L., "Some Continuous Functions Related to Corner Polyhedra," RC-3311 (February 1971), IBM, Yorktown Heights, N. Y.
39. GORRY, G. A. AND SHAPIRO, J. F., "An Adaptive Group Theoretic Algorithm for Integer Programming Problems," *Management Science*, Vol. 17, No. 5 (January 1971), pp. 285–306.
40. ——,—— AND WOLSEY, L. A., "Relaxation Methods for Pure and Mixed Integer Programming Problems," WP 456–70 (April 1970), Sloan School of Management, M.I.T. Also in *Management Science*, Vol. 18, No. 5 (January 1972), pp. 229–239.
41. HAMMER, P. L. AND RUDEANU, S., *Boolean Methods in Operations Research and Related Areas*, Springer-Verlag, Berlin, 1968.
42. HELD, M. AND KARP, R. M., "The Traveling Salesman Problem and Minimum Spanning Trees," *Operations Research*, Vol. 18, No. 6 (November–December 1970), pp. 1138–1162.
43. HERVÉ, P., Private communication, October 2, 1970, Compagnie de Raffinage Shell Berre, Paris.
44. IBARAKI, T., LIU, T. K., BAUGH, C. R. AND MUROGA, S., "An Implicit Enumeration Program for Zero-One Integer Programming," Report No. 305 (January 1969), Dept. of Computer Science, University of Illinois, Urbana.
45. INMAN, R., "User's Guide to IPE," Memorandum 71-2 (February 1971), Development Research Center, International Bank for Reconstruction and Development, Washington, D. C.
46. LAND, A. H. AND DOIG, A. G., "An Automatic Method of Solving Discrete Programming Problems," *Econometrica*, Vol. 28 (1960), pp. 497–520.
47. LASDON, L. S., *Optimization Theory for Large Systems*, Macmillan, New York, 1970.
48. LAWLER, E. L. AND WOOD, D. E., "Branch and Bound Methods: A Survey," *Operations Research*, Vol. 14, No. 4 (July–August 1966), pp. 699–719.
49. LEMKE, C. E. AND SPIELBERG, K., "Direct Search Algorithms for Zero-One and Mixed-Integer Programming," *Operations Research*, Vol. 15, No. 5 (September–October 1967), pp. 892–914.
50. LITTLE, J. D. C., MURTY, K. G., SWEENEY, D. W. AND KAREL, C., "An Algorithm for the Travelling Salesman Problem," *Operations Research*, Vol. 11, No. 6 (November–December 1963), pp. 972–989.
51. MANNE, A. S., "A Mixed Integer Algorithm for Project Evaluation," Memorandum 71-3 (February 1971), Development Research Center, International Bank for Reconstruction and Development, Washington, D. C.
52. MARSTEN, R. E., "An Algorithm for the Set Partitioning Problem with Side Constraints," forthcoming Working Paper #181 (October 1971), Western Management Science Institute, UCLA.
53. MARTIN, G. T., "An Accelerated Euclidean Algorithm for Integer Linear Programming," in

R. L. Graves and P. Wolfe (eds.), *Recent Advances in Mathematical Programming*, McGraw-Hill, New York, 1963.

54. ——, "Solving the Travelling Salesman Problem by Integer Linear Programming," Control Data Corporation, New York, May 1966.
55. ——, "Integer Programming: Gomory Plus Ten," 38th National ORSA Meeting, Miami, November 1969.
56. ——, Private communication, May 4, 1971, Control Data Corporation, Midtown Data Center, New York.
57. MEARS, W. J. AND DAWKINS, G. S., "Comparison of Integer Programming Algorithms," The Pace Company, P.O. Box 26637, Houston. Presented at the 1968 Joint National Meeting of the Operations Research Society of America and The Institute of Management Sciences, San Francisco, May 1968.
58. MITRA, G., "Designing Branch and Bound Algorithms for Mathematical Programming," SIA Limited, 23 Lower Belgrave St., London. Presented at the Seventh International Symposium on Mathematical Programming, The Hague, Holland, September 1970.
59. MITTEN, L. G., "Branch-and-Bound Methods: General Formulation and Properties," *Operations Research*, Vol. 18, No. 1 (January–February 1970), pp. 24–34.
60. PIERCE, J. F., "Application of Combinatorial Programming to a Class of All-Zero-One Integer Programming Problems," *Management Science*, Vol. 15, No. 3 (November 1968), pp. 191–209.
61. ROY, B., BENAYOUN, R. AND TERGNY, J., "From S.E.P. Procedure to the Mixed Ophelie Program," in J. Abadie (ed.), *Integer and Nonlinear Programming*, North-Holland, Amsterdam, 1970.
62. SCHRAGE, L. AND WOILER, S., "A General Structure for Implicit Enumeration," Dept. of Industrial Engineering, Stanford University (1967).
63. SHAPIRO, J. F., "Dynamic Programming Algorithms for the Integer Programming Problem—I: The Integer Programming Problem Viewed as a Knapsack Type Problem," *Operations Research*, Vol. 16, No. 1 (January–February 1968), pp. 103–121.
64. ——, "Group Theoretic Algorithms for the Integer Programming Problem—II: Extension to a General Algorithm," *Operations Research*, Vol. 16, No. 5 (September–October 1968), pp. 928–947.
65. ——, "Turnpike Theorems for Integer Programming Problems," *Operations Research*, Vol. 18, No. 3 (May–June 1970), pp. 432–440.
66. ——, "Generalized Lagrange Multipliers in Integer Programming," *Operations Research*, Vol. 19, No. 1 (January–February 1971), pp. 68–76.
67. SOMMER, D., Private communication, March 30, 1971, Control Data Corporation, Cybernet Service Division, Minneapolis.
68. THIRIEZ, H., "Airline Crew Scheduling: A Group Theoretic Approach," Report R-69 (October 1969), Flight Transportation Laboratory, M.I.T.
69. TOMLIN, J. A., "An Improved Branch and Bound Method for Integer Programming," *Operations Research*, Vol. 19, No. 4 (July–August 1971), pp. 1070–1075.
70. ——, "Branch and Bound Methods for Integer and Non-Convex Programming," in J. Abadie (ed.), *Integer and Nonlinear Programming*, North-Holland, Amsterdam, 1970.
71. TRAUTH, C. A. AND WOOLSEY, R. E., "MESA, A Heuristic Integer Linear Programming Technique," Research Report SC-RR-68-299 (July 1968), Sandia Laboratories, Albuquerque, New Mexico.
72. —— AND ——, "Integer Linear Programming: A Study in Computational Efficiency," *Management Science*, Vol. 15, No. 9 (May 1969), pp. 481–493.
73. WAGNER, W. H., "An Application of Integer Programming to Legislative Redistricting," Crond, Inc., Wilmington, Delaware. Presented at the 34th National Meeting of the Operations Research Society of America, Philadelphia, November 1968.
74. WOILER, S., "Implicit Enumeration Algorithms for Discrete Optimization Problems," Technical Report No. 4 (May 1967), Dept. of Industrial Engineering, Stanford University.
75. WOLSEY, L. A., "Mixed Integer Programming: Discretization and the Group Theoretic Approach," Technical Report No. 42 (June 1969), Operations Research Center, M.I.T.
76. ——, "Extensions of the Group Theoretic Approach in Integer Programming," Working Paper (October 1970), Manchester Business School, University of Manchester, England.
77. YOUNG, R. D., "New Cuts for a Special Class of 0-1 Integer Programs," Research Report in Applied Mathematics and Systems Theory (November 1968), Rice University, Houston.