MEMORANDUM
RM-5644-PR
JUNE 1968

AN IMPROVED IMPLICIT ENUMERATION APPROACH FOR INTEGER PROGRAMMING

A. M. Geoffrion

PREPARED FOR:
UNITED STATES AIR FORCE PROJECT RAND

The RAND Corporation
SANTA MONICA • CALIFORNIA

MEMORANDUM
RM-5644-PR
JUNE 1968

AN IMPROVED IMPLICIT ENUMERATION APPROACH FOR INTEGER PROGRAMMING

A. M. Geoffrion

This research is supported by the United States Air Force under Project RAND — Contract No. F44620-67-C-0045 — monitored by the Directorate of Operational Requirements and Development Plans, Deputy Chief of Staff, Research and Development, Hq USAF. RAND Memoranda are subject to critical review procedures at the research department and corporate levels. Views and conclusions expressed herein are nevertheless the primary responsibility of the author, and should not be interpreted as representing the official opinion or policy of the United States Air Force or of The RAND Corporation.

The RAND Corporation
1700 MAIN ST. • SANTA MONICA • CALIFORNIA • 90406

Published by The RAND Corporation

PREFACE

This Memorandum is part of a continuing RAND research effort in the general area of mathematical programming. It presents a contribution to the effective solution of a general class of discrete optimization problems such as the Air Force encounters in scheduling, sequencing, loading, and other combinatorial and resource allocation contexts.

The present study incorporates and extends findings from several of the author's earlier publications. It is a sequel to Integer Programming by Implicit Enumeration and Balas' Method, The RAND Corporation, RM-4783-PR, February 1966. The basic theoretical results reported here were established some time ago in an unpublished paper [13] and were given, along with preliminary computational experience, in Implicit Enumeration Using an Imbedded Linear Program, The RAND Corporation, RM-5406-PR, September 1967. Additional computational experience was summarized in Recent Computational Experience with Three Classes of Integer Linear Programs, The RAND Corporation, P-3699, October 1967. The present Memorandum supersedes all of these reports, and is intended for operations analysts with a background in mathematical programming and familiarity with RM-4783-PR.

The author is a consultant to The RAND Corporation.

SUMMARY

This Memorandum synthesizes the Balasian implicit enumeration approach to integer linear programming with the approach typified by Land and Doig [19] and by Roy, Bertier and Nghiem [23]. The synthesis results from the use of an imbedded linear program to compute surrogate constraints that are as "strong" as possible in a sense slightly different from that originally used by Glover [15]. A very simple implicit enumeration algorithm fitted with optional imbedded linear programming machinery was implemented and tested extensively on an IBM 7044. Use of the imbedded linear program dramatically reduced solution times in virtually every case, and sufficed to render the tested algorithm superior to the five other implicit enumeration algorithms for which comparable published experience was available. The crucial issue of the sensitivity of solution time to the number of integer variables was given special attention. Sequences were run of set covering, optimal routing, and knapsack problems with multiple constraints of varying sizes up to 90 variables. The results suggest an extraordinary working hypothesis: that use of the imbedded linear program in the prescribed way reduces solution time dependence on the number of variables from exponential to low-order monomial increase. The dependence appeared to be approximately linear for the first two problem classes, with 90 variable problems typically being solved in about 15 seconds; and approximately cubic for the third class, with 80 variable problems typically solved in less than 2 minutes. In the 35-variable range for all three classes, use of the imbedded linear program reduced solution times by a factor of about 100. To what extent the working hypothesis holds for various classes of problems is obviously a matter warranting further study, but on the basis of the existing evidence it would appear that the present approach permits the routine solution of practical integer programs involving hundreds of variables.

ACKNOWLEDGMENT

It is a pleasure to acknowledge the able programming assistance of A. B. Nelson of The RAND Corporation.

CONTENTS

I. INTRODUCTION

Any bounded-variable pure integer linear programming problem can be written in the form

(P) Minimize cx subject to $b + Ax \geq 0$, x_j binary,

where c and x are n-vectors, b is an m-vector, and A is an m x n matrix. The implicit enumeration approach to this problem has been the subject of considerable recent investigation.* This approach uses a flexible "backtracking" procedure for methodically searching through the possible solutions. Its efficiency depends on the exclusion of a sufficient proportion of the possible solutions from further consideration by means of various tests applied to partial solutions. A partial solution is a subset of the n variables that each have a specific binary value; variables excluded from the subset are termed free. The tests usually amount to examining the constraints in an effort to determine whether any completion of the current partial solution could possibly yield a feasible solution of (P) that has a lower value of the objective function than the best known feasible solution. Accordingly, the algorithm either continues by augmenting the current partial solution or backtracks to a different one. Backtracking implies that all possible completions of the current partial solution have been accounted for (implicitly enumerated).

Most of these tests can be applied at a reasonable computational cost essentially to only one constraint at a time. Glover's suggestion [15] for mitigating this limitation is to periodically introduce additional "surrogate" constraints that are redundant in the usual sense and yet effective when the tests are applied to them individually. These constraints are composed primarily from the given constraints by nonnegative linear combination. This is the starting point of the present study.

In the next section we review the rudiments of implicit enumeration with surrogate constraints. In Sec. III we introduce a measure of the "strength" of a given surrogate constraint slightly different from the

*See, for example, Refs. 1, 2, 3, 4, 9, 10, 14, 15, 21, 22, and 26.

one implicitly used by Glover. It is shown how surrogate constraints that are as strong as possible in this sense can be computed by linear programming. This not only obviates the need for approximate methods of finding good surrogate constraints, but also leads to an additional important advantage because the dual of the required linear program coincides exactly with the continuous version of (P) in the free variables. Consequently the resulting class of algorithms for solving (P) can be considered a synthesis of the implicit enumeration approach as typified by Balas [1] and the approach typified by Land and Doig [19] and Roy, Bertier and Nghiem [23]. We also mention the connection of the present work to recent independent work by Balas [2] and Spielberg [24].

In Sec. IV we present extensive computational experience with problems of up to 80 variables taken from the literature. The algorithm, implemented and tested on the IBM 7044, was among the simplest possible with the present approach. No ad hoc "tests" beyond the basic binary infeasibility test (see Sec. II) were used, so that no advantage was taken of special problem structure. The improvements wrought by using an imbedded linear program in the prescribed way were dramatic, and sufficed to make even this bare-bones algorithm very efficient relative to the other implicit enumeration algorithms for which comparable published experience was available (those of Balas [1], Fleischmann [9,10], Lemke and Spielberg [21], Petersen [22], and Woiler [26]).

Section V presents empirical results concerning the influence of problem size (number of variables) on solution time. This is a crucial issue that must be faced by any integer programming approach offered as being of practical interest. Sequences were run of set covering, optimal routing, and knapsack problems with multiple constraints of varying sizes up to 90 variables. The results suggest the following working hypothesis: that use of the imbedded linear program reduces solution time dependence on the number of variables from exponential to low-order monomial increase. The dependence seemed to be approximately linear for the first two problem classes and cubic for the third. If this working hypothesis proves to be even approximately correct for these and other problem classes, then the present approach

permits practical integer programs involving hundreds of variables to be solved routinely. Ninety variable problems of the first two classes were typically solved in about 15 seconds, and 80 variable problems of the third class were typically solved in less than 2 minutes. In the range of 35 variables, for all three classes, use of the imbedded linear program reduced solution times by a factor of about 100.

Section VI presents the easy and natural extension of the present approach to the mixed integer case.

II. IMPLICIT ENUMERATION WITH SURROGATE CONSTRAINTS

Denote a partial solution by an ordered set S, where each element is a nonzero integer between -n and n that may or may not be underlined. An element j (- j) of S indicates that x_j takes on the value 1 (0) in the partial solution. Using an obvious notation, we write $x_j^S = 1$ (= 0) if j (- j) is in S. The significance of an underline at the k^{th} position (counting from the left) is that all completions of the partial solution up to and including the k^{th} element complemented have been accounted for. Associated with any partial solution S is an integer program (P_S) involving the free variables (the variables not fixed by S):

$$(P_S) \qquad \text{Minimize} \quad \sum_{j\in S} c_j x_j^S + \sum_{j\notin S} c_j x_j \quad \text{subject to}$$

$$b_i + \sum_{j\in S} a_{ij} x_j^S + \sum_{j\notin S} a_{ij} x_j \geq 0, \quad i = 1, \ldots, m$$

$$0 \leq x_j \leq 1 \text{ and integer}, j \notin S,$$

where the notation $j \in S$ ($\notin S$) refers to the fixed (free) variables.

In addition to the original m constraints, (P_S) may also be expanded to include one or more surrogate constraints, each of which is a nonnegative linear combination of the original constraints plus the constraint $(\bar{z} - cx) > 0$, where $\bar{z}$ is the value of the currently best-known feasible solution of (P).* More precisely, each surrogate constraint has the form

---

*If no feasible solution is known a priori (indeed, (P) may be infeasible), $\bar{z}$ can be initially taken as

$$\sum_{j=1}^{n} |c_j|.$$

$$\mu(b + Ax) + (\bar{z} - cx) > 0$$

for some nonnegative m-vector μ. Such a constraint is clearly satisfied by any feasible solution of (P) that has a better value of cx than $\bar{z}$.

From the results of Ref. 14 or 15 it follows that the following schema terminates in a finite number of steps either with an optimal solution of (P), or with an indication that no feasible solution of (P) exists with value less than the initial value of $\bar{z}$. The sequence of partial solutions generated is nonredundant in the appropriate sense.

SCHEMA FOR SOLVING (P) BY IMPLICIT ENUMERATION

Step 0: Initialize $\bar{z}$ at a known upper bound on the optimal value of (P), and S at an arbitrary partial solution without underlines.

Step 1: If (P_S) is obviously devoid of a feasible solution with value less than $\bar{z}$, go to Step 4. If (P_S) has an obvious optimal solution with value less than $\bar{z}$, then replace $\bar{z}$ by this value, store the optimal solution as an incumbent, and go to Step 4. If any free variable must obviously take on a particular binary value in order for (P_S) to have a feasible solution with value less than $\bar{z}$, then augment S on the right by $\underline{j}$ ($-\underline{j}$) for each variable x_j that must take on the value 1 (0).

Step 2: Add a new surrogate constraint and/or delete one or more current surrogate constraints, or do neither.

Step 3: Augment S on the right by $\pm j$ for some free variable (or several free variables) x_j.

Step 4: Locate the rightmost element of S that is not underlined. If none exists, terminate; otherwise, replace the element by its underlined complement and drop all elements to the right. Return to Step 1.

There is a wide variety of possible mechanisms for implementing Steps 1 and 3. Many can be found in, or adapted from, Refs. 1, 4, 9,

10, 14, 15, 21, 22, and 26. The possibilities are further multiplied because the conditional instructions of Step 1 can be executed in any order or even in parallel. It is important to observe that many of the possible mechanisms, and perhaps most of the ones that are relatively inexpensive computationally, essentially apply to the constraints taken one at a time. At Step 1, for example, a prominent role is often played by tests for binary-infeasibility and for conditional binary-infeasibility, with each constraint being considered individually. A constraint is said to be binary-infeasible if it has no binary solution, and is said to be conditionally binary-infeasible if its binary feasibility is conditional upon certain of the variables taking on particular binary values. It is easily verified that $\beta + \Sigma_j \alpha_j x_j \geq 0$ (> 0) is binary infeasible if and only if $\beta + \Sigma_j \text{ Max } \{0, \alpha_j\} < 0$ (≤ 0); and $\beta + \Sigma_j \text{ Max } \{0, \alpha_j\} - |\alpha_{jo}| < 0$ (≤ 0) implies $x_{jo} = 0$ or 1, according as $\alpha_{jo} < 0$ or $\alpha_{jo} > 0$, in any binary solution of $\beta + \Sigma_j \alpha_j x_j \geq 0$ (> 0).

This leads naturally to the desire to introduce surrogate constraints at Step 2 that are "strong" in the sense that such tests are effective when applied to them.

III. COMPUTING STRONGEST SURROGATE CONSTRAINTS BY LINEAR PROGRAMMING

Since at any given stage of the calculations only a subset of the variables is free, the "strength" of a surrogate constraint must be defined relative to the current partial solution S. The special role played by conditional and unconditional binary-infeasibility (see Sec. II) suggests that "strength" be defined in accordance with how near a constraint is to being binary-infeasible.

Definition. The surrogate constraint $\mu^1(b + Ax) + (\bar{z} - cx) > 0$ is said to be stronger relative to S than the surrogate constraint $\mu^2(b + Ax) + (\bar{z} - cx) > 0$ if the maximum of the left-hand side of the first constraint is less than the maximum of the left-hand side of the second constraint, the maxima being taken over binary values of the free variables.

(For purposes of comparison, the corresponding definition Glover implicitly uses [15] seems to be: the surrogate constraint $\mu^1(b + Ax) \geq 0$ is said to be stronger relative to S than the surrogate constraint $\mu^2(b + Ax) \geq 0$ if the maximum of $(\bar{z} - cx)$ subject to the first constraint is less than the maximum of $(\bar{z} - cx)$ subject to the second constraint, the maxima being taken over binary values of the free variables.)

Finding a strongest surrogate constraint is, then, the problem of minimizing, over all $\mu \geq 0$, the expression

$$(1) \qquad \text{Max}\{\mu(b + Ax) + (\bar{z} - cx) \,|\, x_j = 0 \text{ or } 1,\ j \notin S \text{ and } x_j = x_j^S, j \in S\}$$

or, upon rearrangement, the expression

$$(2) \qquad \sum_{i=1}^{m} \mu_i b_i^S + \bar{z} - z^S + \text{Max} \left\{ \sum_{j \notin S} \left(\sum_{i=1}^{m} \mu_i a_{ij} - c_j \right) x_j \,\middle|\, x_j = 0 \text{ or } 1,\ j \notin S \right\},$$

where we have introduced the notation

$$z^S = \sum_{j \in S} c_j x_j^S \quad \text{and} \quad b_i^S = b_i + \sum_{j \in S} a_{ij} x_j^S.$$

Now for any $\mu \geqq 0$ we have

$$(3) \qquad \text{Max} \left\{ \sum_{j \notin S} \left(\sum_{i=1}^{m} \mu_i a_{ij} - c_j \right) x_j \,\middle|\, x_j = 0 \text{ or } 1,\ j \notin S \right\}$$

$$= \text{Max} \left\{ \sum_{j \notin S} \left(\sum_{i=1}^{m} \mu_i a_{ij} - c_j \right) x_j \,\middle|\, 0 \leq x_j \leq 1,\ j \notin S \right\}$$

$$= \text{Min} \left\{ \sum_{j \notin S} w_j \,\middle|\, w_j \geq 0 \text{ and } w_j \geq \sum_{i=1}^{m} \mu_i a_{ij} - c_j, j \notin S \right\},$$

where the second equality follows from the Dual Theorem of linear programming. Using (3) in (2), we obtain the following linear program:

$$\underset{\mu, w}{\text{Minimize}} \quad \sum_{i=1}^{m} \mu_i b_i^S + \bar{z} - z^S + \sum_{j \notin S} w_j$$

$$(LP_S) \qquad \text{subject to} \quad w_j \geq \sum_{i=1}^{m} \mu_i a_{ij} - c_j, \ j \notin S$$

$$w_j \geq 0, \ j \notin S \text{ and } \mu \geqq 0.$$

Denote the infimal value of (LP_S) by $v(LP_S)$. It is easy to see that the desired result is now at hand.

Proposition: Let S be an arbitrary partial solution. Then (LP_S) is necessarily feasible, and

A. $v(LP_S) \leq 0 \Leftrightarrow$ there exists a binary-infeasible surrogate constraint;

B. $v(LP_S) > 0 \Rightarrow$ there does not exist a binary-infeasible surrogate constraint, but any optimal μ in (LP_S) yields a strongest surrogate constraint relative to S.

The usefulness of (LP_S) is further enhanced by the fact that it is precisely the dual of the continuous version of (P_S) (in which $0 \leq x_j \leq 1$ replaces $x_j = 0$ or 1). Consequently, with the help of the Dual Theorem of linear programming, one can verify that $\bar{z} - v(LP_S)$ is a lower bound on the optimal value of (P_S), and the optimal dual variables of (LP_S) are optimal in (P_S) if they are all integers.

These results show not only how strongest surrogate constraints can be calculated by linear programming, but also that the aim of Step 1 can be accomplished at no extra computational cost in the course of attempting to construct a strongest surrogate constraint at Step 2. More specifically, one would set out to construct a strongest surrogate constraint by executing simplex iterations on (LP_S) until one of the following mutually exclusive events occurs:

(a) the value of the objective function of (LP_S) becomes ≤ 0;
(b) an optimal solution of (LP_S) is obtained, and $v(LP_S) > 0$ and the optimal dual variables are all integers;
(c) an optimal solution of (LP_S) is obtained, and $v(LP_S) > 0$ but not all of the dual variables are integers.

In event (a), a binary infeasible surrogate constraint must exist, and consequently one may go to Step 4; in event (b), the optimal solution of (P_S) is given by the optimal dual variables of (LP_S), so one should replace $\bar{z}$ by the new value and the incumbent by the new solution and go to Step 4; in event (c), a strongest surrogate constraint is obtained from the optimal μ in (LP_S).

Restarting techniques primal with respect to (LP_S) can conveniently be used to take advantage of the results of previous calculations each time Step 2 is executed. Since we do not always optimize (LP_S), some of these techniques are pre- as well as post-optimality. The Revised Simplex format is convenient in this regard. Keep in mind that the columns of the w_j are just the negatives of the unit vectors associated with the corresponding slack variables. One consequence is that the w_j variables can be treated logically rather than algebraically, so that (LP_S) is reduced to essentially m nontrivial variables and as many constraints as free variables. The other important consequence is that it is easy to write down a basic feasible solution to (LP_S) for any S; in fact, there is an obvious and simple procedure for modifying any basic feasible solution for (LP_S) until it becomes basic feasible for ($LP_{S'}$), where $S' \neq S$. This avoids the need for restarting techniques that are dual with respect to (LP_S).

It should be noted that the developments of this section, although unique in terms of motivation and details of use, are nevertheless related in certain respects to independent work by other authors. Land and Doig [19], Dakin [7], Roy, Bertier and Nghiem [23], Hervé [18] and others have also described algorithms that use the continuous approximation to discrete programs like (P_S); but they make little or no use of the dual variables (μ and w). The dual variables μ are, however, used by Balas in his "filter algorithm" [2] to determine his "filter constraint" -- a close relative of the type of surrogate constraint employed here. He solves the continuous approximation to (P_S) only once, with S empty, rather than periodically. Recently, Glover [16] has illuminated the relationship between Balas' filter constraint and the strongest surrogate constraint of this study (as presented in an earlier draft), and has suggested possible extensions. Spielberg [24] has also independently used an approach akin to the present one, in the special context of the simple plant location problem. He has suggested in a private communication to the author that his "gain function" is analogous to our surrogate constraint.

IV. COMPARATIVE COMPUTATIONAL EXPERIENCE

IMPLEMENTATION

A simple version of the implicit enumeration procedure has been implemented and extensively tested on an IBM 7044. It is of completely general applicability, and takes no advantage of special problem structures. Step 1 uses just the simple tests for conditional and unconditional binary-infeasibility mentioned at the end of Sec. II; it recognizes an obvious optimal solution of (P_S), in the fashion of Balas [1], only by the trivial minimization of cx over binary values of the free variables while ignoring the constraints, and then testing the resulting solution for feasibility. Step 2 follows the outline and suggestions given at the end of Sec. III, except that a binary-infeasible surrogate constraint is added anyway in event (a), and in event (c) the non-integer dual variables are rounded in a simple attempt to discover a good feasible solution of (P_S). Step 3 uses a simplified version of Balas' augmentation rule: augment S by j_o, where j_o maximizes over all free variables the expression

$$\sum_{i=1}^{m} \text{Min } \{0,\ b_i^S + a_{ij}\}.$$

(This assumes, without loss of generality, that $c \geq 0$.)

The program was written entirely in Fortran IV for RAND's 32,000 word machine. The object program and its data are all-in-core, with all problem data treated as floating point, and is dimensioned to handle problems with up to 90 variables and 50 constraints (including surrogate constraints, if any).* The linear programming subroutine

*More than 32,000 words of primary storage are required if larger problems are to be solved; the number of words required is approximately $n^2 + n(3m + 18) + 9{,}000$.

is basically a Revised Simplex method with explicit inverse, the starting point having been a routine due to Clasen [6]. Restarting techniques were incorporated that use a labeling procedure rather than more conventional matrix manipulations. The labeling procedure is based on the observation that fixing a variable at the value 0 or 1 can be viewed as demanding equality in the appropriate inequality constraint among $0 \leq x_j \leq 1$, $j \in S$, in the continuous version of (P_S). This means that the corresponding dual variables (the w_j and slacks in (LP_S)) become unconstrained in sign; the appropriate variables are therefore labeled and treated as "unsigned." This procedure is easier to program than a more conventional one using matrix manipulations, and has the advantage of being economical in terms of core and setup time for the successive linear programs. It has the drawback, however, that (LP_S) (and therefore the explicit inverse) always has n rows, instead of only as many rows as free variables. Hence each pivot requires more work.

RESULTS

The code has been used to solve numerous test problems with up to 80 variables taken from the literature.* The number of iterations (executions of Step 1) and execution times (until termination, to the nearest hundredth of a minute) for most of these problems is presented in Table 1. We have omitted the problems too small to be of interest. Each problem was run twice: once skipping Step 2, so that no surrogate constraints were ever computed; and once with Step 2 fully implemented, so that an attempt was made to compute a new surrogate constraint each time, with only the last four surrogate constraints being kept and used. The columns corresponding to these runs are labeled "No LP" and "LP Every Time."

No prior information was used, such as an obvious initial feasible solution or upper bound on the optimal value of the objective function. Such information was usually available by inspection, but we did not wish to further confound comparability with other investigators'

*Refs. 1, 4, 11, 17, 21, 22, and 25.

computational results, which are reproduced in Table 1 for easy reference. These other investigators are: Bouvier and Messoumian [4], whose problems are randomly generated without any special structure at all; Fleischmann [9], whose "economic" problems are highly structured; Lemke and Spielberg [21], whose problems B and D2 are attributed to M. Sidrow of Texaco, and problem C to W. Arcuri of IBM; Petersen [22], whose problems are of the Lorie-Savage capital budgeting variety; and Woiler [26] who ran, among other problems, a number of Haldi's fixed charge problems [17] and some of the "IBM test problems" also published by Haldi. Each of these investigators used a different adaptation of the implicit enumeration approach.

The results summarized in Table 1 indicate that use of the imbedded linear program (LP_S) dramatically reduces the number of required iterations, often by several orders of magnitude; and that this reduction is more than enough to pay for the time spent working on (LP_S), since total execution times were greatly reduced in almost every case.

The present algorithm is evidently very efficient relative to the others; but differences in programming and machine speed make it inadvisable to hazard a quantitative estimate of the apparent improvement.

For comparison with various cutting-plane algorithms for the Haldi and IBM problems, see [25]; such a comparison, although by no means unfavorable to the present algorithm, is somewhat prejudiced because upper bounds on variables have been handled here by binary representation rather than directly.

A rather conspicuous feature of the algorithm with the imbedded linear program is that it not only tends to find near-optimal solutions quickly, but also tends to verify an optimal solution relatively promptly after one is found. Exceptions are IBM 5 and 9 (in each case an optimum was found in 0.01 min.) and most of the Bouvier and Messoumian problems (where about 90 percent of the computing time was spent verifying an optimum).

OPPORTUNITIES FOR IMPROVEMENT

The results presented in Table 1 are subject to further improvement. Substantial improvement could often have been gained simply by exercising the basic options of the experimental program in a

Table 1

COMPARATIVE COMPUTATIONAL EXPERIENCE

Problem Designation	Problem Size: 0-1 Var x Constraints	No LP[a] Iterations	No LP[a] (7044) Min	LP Every Time[a] Iterations	LP Every Time[a] (7044) Min	Other Algorithms Min	Machine	Version	Ref
Bouvier and Messoumian [4]									
15	20 x 20	515	0.48	21	0.09	0.47	7044		4
16	20 x 20	1,897	1.69	89	0.62	2.07	7044		4
17	20 x 23	889	1.06	115	0.64	0.85	7044		4
18	20 x 23	569	0.59	1	0.04	0.97	7044		4
22	25 x 20	4,267	4.88	143	0.92	3.27	7044		4
23	27 x 20	6,565	8.08	171	1.18	7.10	7044		4
24	28 x 20	> 8,117	>10.00	281	1.93	15.20	7044		4
Fleischmann [11]									
I-35	35 x 8	1,009	0.50	4	0.02[b]	0.04[cd]	7094	Ref. 10	9,10
I-50	50 x 11		>10.00	9	0.06[b]	0.24[cd]	7094	Ref. 10	9,10
I-60	60 x 11	>10,367	>10.00	3	0.04	1.68[d]	7094	Ref. 10	9,10
I-80-1	80 x 11	> 8,557	>10.00	1	0.03	8.95[d]	7094	Ref. 10	9,10
I-80-2	80 x 11	> 8,337	>10.00	19	0.20	8.28[d]	7094	Ref. 10	9,10
Lemke and Spielberg [21]									
B	35 x 28	1,995	2.39	35	0.19	5.5	360/40	DZIP1	21
C	44 x 12	2,061	1.90	447	1.38	5	360/40	DZIP1	21
D2	74 x 37	> 3,838	>10.00	>517	>10.00	>30	7094	DZIP1	21
Petersen [22]									
4	20 x 10	893	0.46	27	0.04	0.06[e]	7094	R1&R2	22
5	28 x 10	13,387	8.85	181	0.24	0.16[e]	7094	R1&R2	22
6	39 x 5	>18,857	>10.00	143	0.43	1.18[e]	7094	R1&R2	22
7	50 x 5	>15,577	>10.00	115	0.46	9.55[e]	7094	R1&R2	22
Haldi [17]									
II-7[f]	20 x 4	459	0.10	63	0.03	0.02	360/40	DZIP1	21
"	"	"	"	"	"	0.36	B5500	LAR-MAX	26
II-8[f]	20 x 4	511	0.10	79	0.05	0.40	B5500	LAR-MAX	26
II-10	30 x 10	769	0.41	75	0.06	----	-----	-------	
IBM [17]									
1[g]	21 x 7	435	0.14	17	0.01	0.13	B5500	LAR-MAX	26
2[g]	21 x 7	369	0.11	27	0.02	0.17	B5500	LAR-MAX	26
3[g]	20 x 3	217	0.04	17	0.01	0.04	B5500	LAR-MAX	26
4[g]	30 x 15	>13,995	>10.00	57	0.13	6.67	7094	DZIP1	21
5[g]	30 x 15	>13,858	>10.00	365	1.90	50.00	7094	DZIP1	21
6	31 x 31	> 7,038	>10.00	209	2.00	>50.00	7094	DZIP1	21
9[h]	15 x 35	551	0.45	107	0.44	---	----	-----	

[a]Except for the Haldi and IBM problems, the initial partial solution was taken to be the empty one. In the Haldi problems, the initial partial solution consisted of all variables at the value 0. In the IBM problems, each of the original variables was set at the value 1.

[b]Average for five slightly different problems of the same size.

[c]Average for ten slightly different problems of the same size.

[d]These times [11] are much better than the times announced in [9] because of subsequent modifications [10].

[e]These times [20] are much better than those announced in [22] because Petersen's program was converted to a faster machine. The R1 & R2 modifications are the best of several that Petersen developed.

[f]This problem was converted to 0-1 form using a six-place binary expansion for each original activity variable. This seems to be the same conversion as was used in [21] and/or [26].

[g]Problems 1 through 5 were converted to 0-1 form by binary representation of the variables using an upper bound of 7, 7, 31, 3, and 3 in each problem, respectively. This seems to be the same conversion as was used in [21] or [26].

[h]This problem had 50 constraints originally, but the last 15 have been eliminated here since they only specify that the variables be binary.

discriminating rather than uniform manner. It is often more economical, for example, to employ the imbedded linear program less frequently than at every opportunity. Lemke and Spielberg's problem C was solved in 0.45 instead of 1.38 minutes when (LP_S) was used only every sixteenth time. Another possibility is to take advantage of an obvious upper bound on the optimal value of a given problem. When $\bar{z}$ was initially set at an upper bound for IBM 6 that was in error by 20 percent, the solution time was 1.25 instead of 2.00 minutes. A third possibility for improvement involves making a more appropriate choice of the initial partial solution. More plausible initial partial solutions than the empty one are often obvious when the problem data are inspected. This was so for IBM 6, and its use reduced the solution time from 2 minutes to less than 4 seconds. Another source of good initial solutions for some problems is the linear programming solution of the continuous approximation to (P). If we take the initial partial solution to be empty, the dual variables of (LP_S) the first time around yield this solution. Taking the next partial solution to consist of the naturally integral variables causes the algorithm in effect to examine all possible roundings of the continuous solution first. An option to begin in this way -- we call this the "LP start" -- has proven effective for many problems (e.g., Lemke and Spielberg C is solved in 0.44 minute, and IBM 6 in 0.34 minute).

Finally, we mention that a very important source of improvement would be to use tests at Step 1 that are more powerful than the simple binary-infeasibility test. The dramatic improvements Fleischmann [9] and Petersen [22] achieved over the original Balasian algorithm testify to the untapped potential here. Some of the more powerful tests are of general applicability, while others capitalize on special problem structures (remember that we have not taken any advantage here whatever of special structure in the test problems). Obviously one can take advantage of structure not only by introducing specialized tests at Step 1, but also by exploiting it in the LP subroutine that is used to solve (LP_S), since (LP_S) inherits the structure of (P). It remains also to implement various augmentation rules for Step 3 other than Balas'. For example, combining Balas' augmentation rule with a

restriction to the free variables corresponding to fractional dual variables of (LP_S) seems to generally improve performance -- for Lemke and Spielberg C, it reduced the solution time to 0.60 minute.

V. INFLUENCE OF PROBLEM SIZE

The ultimate practical usefulness of any integer programming algorithm depends on the crucial question, "How fast does solution time increase with problem size?" The number of variables is perhaps the main determining factor for implicit enumeration algorithms, since the number of possible solutions of (P) is 2^n. If solution times tend to increase exponentially with the number of variables,* as has been suggested of Balas' algorithm,** then there is little hope of ever being able to solve really large problems directly.

Probably this important question can be answered only within the context of specific problem classes, and even then only with a specifically parameterized population of problems of differing sizes. We shall summarize our experimental investigations in the range of 30-90 variables for three important classes of problems: set covering, optimal routing, and knapsack with several constraints. The main conclusion which emerges is that, for the problems run, the imbedded linear program mitigates approximately exponential dependence of computing time on the number of variables to what appears to be low-order monomial dependence -- approximately linear in the first two cases, and cubic in the third. If these results are at all indicative, then the present algorithm can be expected to cope routinely with quite large structured integer linear programs.

SET COVERING

Set covering problems*** were randomly generated**** to have 30 constraints, a density of 0.07, and no column dominance. Five samples each were generated with 30, 40, ..., 90 variables. Each problem was

---

*That is, if the solution time is proportional to some constant greater than unity raised to the n^{th} power.

**See Castellan [5], Fulkerson [12], and the results below.

***Balinski [3] has given an excellent discussion of the structure and applications of set covering problems.

****Professor Melvin Breuer, USC, generated the problems incidentally to certain other investigations.

run with an empty initial partial solution, (LP_S) used at every opportunity, and with the last four surrogate constraints retained. The computational results, presented in Table 2, suggest that the solution times are increasing approximately linearly with the number of variables.

Table 2

SET COVERING PROBLEMS (30 CONSTRAINTS)

No. 0-1 Variables	Avg Solution Time For 5 Problems (Min)	Avg No. Iter
30	0.03	3
40	0.07	6
50	0.08	4
60	0.14	6
70	0.15	8
80	0.21	6
90	0.17	4

An attempt was also made to run some of the problems without the imbedded linear program. The two attempted 30-variable problems were solved in 1.6 and 1.2 minutes, but the three attempted 40-variable problems each exceeded their time limit of 16 minutes. A sequence of smaller set covering problems with 15 constraints and density 0.2 was then run. The results are summarized in Table 3. When plotted on semilog paper, it seems that times are increasing exponentially with the number of variables.

Table 3

SET COVERING PROBLEMS (15 CONSTRAINTS): IMBEDDED LINEAR PROGRAM NOT USED

No. 0-1 Variables	Avg Solution Time For 4 Problems (Min)	Avg No. Iter
15	0.02	44
20	0.07	137
25	0.42	689
30	1.13	1657
35	3.59	4744

Finally, a representative selection of the above problems was converted to have a modified form of objective function which is supposed [3, p. 306] to be relatively difficult for cutting-plane algorithms: each cost coefficient was set at the number of ones in its column. Experiments indicate that this kind of weighted set covering problem is only slightly more difficult when the imbedded linear program is used.

OPTIMAL ROUTING IN NETWORKS

Consider a network with nodes connected by L links, with link i having capacity c_i, and having K distinguished source-sink pairs. Associated with each source-sink pair is a required flow d_k, and a collection of routes along which flow is permitted. Let the set J_k index the routes permitted for source-sink pair k, and y_j be the flow at unit price p_j through route j for $j \in J_k$. Designate the link-route matrix by A; that is, a_{ij} is 1 if link i is on route j, and is 0 otherwise. The problem is to meet the demands at minimum cost by integral flow over permissible routes, without exceeding the link capacities:

$$\text{Minimize} \sum_{k=1}^{K} \sum_{j \in J_k} p_j y_j \text{ subject to}$$

$$\sum_{k=1}^{K} \sum_{j \in J_k} a_{ij} y_j \leq c_i, \quad i=1, \ldots, L$$

$$\sum_{j \in J_k} y_j \geq d_k, \quad k=1, \ldots, K$$

$$y_j \geq 0, \; y_j \text{ integer.}$$

The particular numerical example used was adapted from the model and data of [8]. In the language of this reference, there are 9 air

bases (nodes) and 9 origin-destination pairs (K = 9) requiring various tonnages (d_k) of air cargo shipments. A total of 45 permissible routes are allowed for the shipments, each of which involves a subset of 32 given segments (links). Each segment is taken to contribute a cost per unit shipped proportional to its air mileage, and to have a capacity of 3 units of air cargo. Hence each y_j is bounded above by 3, and therefore admits binary representation by two binary variables; a total of 90 binary variables is required.

The problem was solved in full 90 variable form and also with some of the routes eliminated. In each case the initial partial solution was empty, (LP_S) was used at every opportunity, and the last four surrogate constraints were retained. The solution times are given in Table 4. As with the set covering problems, solution times seem to increase not much faster than linearly with the number of variables.

Table 4

OPTIMAL ROUTING PROBLEMS (41 CONSTRAINTS)[a]

No. 0-1 Variables	Solution Time (Min)	Iterations
36	0.06	15
48	0.08	15
60	0.14	23
72	0.20	31
84	0.24	31
90	0.26	31

[a]Due to the method of problem generation, some of the constraints turn out to be trivial in the problems with less than 90 variables.

When the algorithm ran without the imbedded linear program on the two smaller problems, it tended to find an optimal solution just about as quickly, but took much longer to verify optimality--8.0 minutes for the 36-variable problem, and in excess of the 16-minute time limit for the 48-variable problem.

KNAPSACK WITH MULTIPLE CONSTRAINTS

A "knapsack" problem with 5 constraints and 82 variables was constructed from the largest problem in [22]. The latter problem, of the Lorie-Savage capital budgeting variety with 50 alternative projects subject to 5 capital constraints, was augmented by 32 similar projects. Smaller problems were then constructed by randomly selecting subsets of the 82 projects. The right-hand side was kept in constant proportion to the row sums of the resulting coefficient matrix in an attempt to preserve the relative degree of problem difficulty. Problems were generated in this way with 30, 40, ..., 80 variables, with four samples at each size. Each problem was run using an "LP start" (see Sec. IV), with (LP_S) used at every opportunity, and with the last six surrogate constraints retained. The results are given in Table 5.

Table 5

KNAPSACK (5 CONSTRAINTS)

No. 0-1 Variables	Avg Solution Time For 4 Problems (Min)	Avg No. Iterations
30	0.09	79
40	0.27	139
50	0.44	164
60	0.55	189
70	1.11	293
80	1.85	361

Without the imbedded linear program, the results given in Table 6 were obtained.

Table 6

KNAPSACK (5 CONSTRAINTS):
IMBEDDED LINEAR PROGRAM NOT USED

No. 0-1 Variables	Avg Solution Time For 4 Problems (Min)	Avg No. Iterations
15	0.02	178
20	0.06	402
25	0.36	1,857
30	1.59	7,064
35	9.74	38,385

With the imbedded linear program, solution times seem to increase approximately as the third power of the number of variables. Without the imbedded linear program, however, a semilog plot suggests that times are going up exponentially with the number of variables.

VI. EXTENSION TO THE MIXED INTEGER CASE

Extension to the mixed integer problem is natural and completely straightforward. Suppose that (P) is of the form

$$
\begin{aligned}
&\text{Minimize} \quad cx + dy \\
&\text{Subject to} \quad Ax + Dy + b \geqq 0 \\
&\qquad\qquad 0 \leq x_j \leq 1 \text{ and integer, } j = 1, \ldots, n \\
&\qquad\qquad y \geqq 0,
\end{aligned} \tag{4}
$$

where d and y are n_1-vectors and D is an m x n_1 matrix. A partial solution is still defined in terms of a subset of the x_j, but a "completion" now involves a choice of y as well as a choice of the free x_j variables. The analog of (P_S) is the mixed program:

$$
\begin{aligned}
&\text{Minimize} \quad z^S + \sum_{j \notin S} c_j x_j + dy \\
&\text{subject to} \quad b_i^S + \sum_{j \notin S} a_{ij} x_j + \sum_{j=1}^{n_1} d_{ij} y_j \geq 0, \; i = 1, \ldots, m \\
&\qquad\qquad 0 \leq x_j \leq 1 \text{ and integer, } j \notin S \\
&\qquad\qquad y_j \geq 0, \; j = 1, \ldots, n_1,
\end{aligned} \tag{5}
$$

and the procedure of Sec. II remains valid with surrogate constraints of the form

$$\mu(b + Ax + Dy) + (\tilde{z} - cx - dy) > 0.$$

The concept of binary-infeasibility is modified to account for the presence of y in the obvious way, and so is the definition of "strength." The analog of (LP_S) is

$$\underset{\mu,w}{\text{Minimize}} \quad \sum_{i=1}^{m} \mu_i b_i^S + \bar{z} - z^S + \sum_{j \notin S} w_j$$

(6)

$$\text{Subject to} \quad w_j \geq \sum_{i=1}^{m} \mu_i a_{ij} - c_j, \; j \notin S$$

$$\mu D \leqq d.$$

$$\mu \geqq 0$$

$$w_j \geq 0, \; j \notin S.$$

One uses (6) in a manner precisely analogous to the use of (LP_S). The only possible complication is that (6) may be infeasible, i.e., $\{\mu \geqq 0 : \mu D \leqq d\}$ may be empty. To avoid this possibility, we can assume that the minimand of the continuous version of (4) (i.e., without the integrality requirement) is bounded from below. This assumption can be enforced, if necessary, by adding an additional constraint such as $\Sigma_{j=1}^{n} y_j \leq M$ or $cx + dy \geq -M$, where M is a suitably large positive number. Then it is easy to show that (6) is infeasible only if (4) is also infeasible, in which case the entire procedure terminates.

This extension has not been tested computationally.

Another extension that has not been tested computationally is the handling of general upper bounds on the integer variables directly rather than by conversion to the 0-1 case by binary representation. This can be done with a slightly modified form of the backtracking procedure used here. The induced changes should be obvious, such as a change in the coefficient of w_j in (LP_S) from 1 to the given upper bound on x_j. An alternative would be to keep the binary representation of upper bounded variables but to take advantage of the resulting special structure; in an expansion requiring 5 binary variables, for example, only one of the corresponding columns of the A matrix need be stored explicitly since the other 4 can be generated as needed from the stored column.

REFERENCES

1. 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.

2. -----, "Discrete Programming by the Filter Method," Operations Research, Vol. 15, No. 5, September-October 1967, pp. 915-957.

3. Balinski, M. L., "Integer Programming: Methods, Uses, and Computation," Management Science, Vol. 12, 1965, pp. 253-313.

4. Bouvier, B., and G. Messoumian, "Programmes linéaires en variables bivalentes, algorithme de Balas," Université de Grenoble, France, June 1965.

5. Castellan, J. W., "Political Apportionment by Computer," Brown University Computing Review, Vol. I, No. 2, pp. 5-24.

6. Clasen, R. J., Using Linear Programming as a Simplex Subroutine, The RAND Corporation, P-3267, November 1965.

7. Dakin, R. J., "A Tree-Search Algorithm for Mixed Integer Programming Problems," Computer Journal, Vol. 8, No. 3, October 1965, pp. 250-255.

8. Fetter, R. B., and R. C. Steorts, A Model for the Design and Evaluation of Air Cargo Systems, The RAND Corporation, RM-4801-PR, October 1966.

9. Fleischmann, B., "Computational Experience with the Algorithm of Balas," Operations Research, Vol. 15, No. 1, January-February 1967, pp. 153-155.

10. -----, "Programm zur Lösung linearer Optimierungsaufgaben mit Null-Eins-Variablen," Deutsches Rechenzentrum, October 19, 1966.

11. -----, private communication, November 30, 1966.

12. Fulkerson, D. R., Review 1015, Mathematical Reviews, Vol. 32, No. 1, July 1966, p. 170.

13. Geoffrion, A. M., "An Improved Algorithm for Integer Programming by Implicit Enumeration," August 23, 1965 (unpublished).

14. -----, Integer Programming by Implicit Enumeration and Balas' Method, The RAND Corporation, RM-4783-PR, February 1966, and in SIAM Review, Vol. 9, No. 2, April 1967, pp. 178-190.

15. 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.

16. -----, "Surrogate Constraints," Working Paper 68-5, Graduate School of Business, University of Texas, Austin, September 1967.

17. Haldi, J., "25 Integer Programming Test Problems," Working Paper No. 43, Graduate School of Business, Stanford University, December 1964.

18. Hervé, P., "Résolution des programmes linéaires a variables mixtes par la procédure S. E. P.," Metra, Vol. VI, No. 1, 1967, pp. 77-91.

19. Land, A. H., and A. G. Doig, "An Automatic Method of Solving Discrete Programming Problems," Econometrica, Vol. 28, 1960, pp. 497-520.

20. Laughhunn, D. J., private communication, October 31, 1967.

21. Lemke, C., and K. Spielberg, Direct Search Zero-One and Mixed Integer Programming, IBM Corporation, IBM Technical Report 39.008, New York Scientific Center, July 1966.

22. Petersen, C. C., "Computational Experience with Variants of the Balas Algorithm Applied to the Selection of R & D Projects," Management Science, Vol. 13, No. 9, May 1967, pp. 736-750.

23. Roy, B., P. Bertier, and P. T. Nghiem, "Programmes linéaires en nombres entiers et procédure S. E. P.," Metra, Vol. IV, No. 3, 1965.

24. Spielberg, K., An Algorithm for the Simple Plant Location Problem with Some Side Conditions, IBM Corporation, IBM Technical Report 320-2900, New York Scientific Center, May 1967.

25. Trauth, C. A., and R. E. Woolsey, Practical Aspects of Integer Linear Programming, Sandia Corporation Monograph, SC-R-66-925, August 1966.

26. Woiler, S., "Implicit Enumeration Algorithms for Discrete Optimization Problems," Ph.D. Dissertation, Department of Industrial Engineering, Stanford University, May 1967.