SAND2223C

MASTER

**Parametric Integer Linear Programming:
A Synthesis of Branch and Bound with Cutting Planes**

By

Suzanne L. K. Rountree
Department 4723
Sandia National Laboratories
Albuquerque, NM 87112

Billy E. Gillett
Computer Science Department
University of Missouri - Rolla
Rolla, Missouri 65401

DISCLAIMER

This book was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency Thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

ABSTRACT

A branch and bound algorithm is designed to solve the general integer linear programming problem with parametric right-hand-sides. The right-hand-sides have the form $b + \theta d$ where b and d are conformable vectors, d consists of nonnegative constants, and θ varies from zero to one.

The method consists of first determining all possible right-hand-side integer constants and appending this set of integer constants to the initial tableau to form an expanded problem with a finite number of family members. The implicit enumeration method gives a lower bound on the integer solutions. The branch and bound method is used with fathoming tests which allow one family member possibly to fathom other family members. A cutting plane option applies a finite number of cuts to each node before branching. In addition, the cutting plane method is invoked whenever some members are feasible at a node and others are infeasible. The branching and cutting process is repeated until the entire family of problems has been solved.

1. Introduction

Parametric integer linear programming problems (PILP) have been solved by implicit enumeration, branch and bound, and cutting plane techniques. However, very little has been done with combinations of the above on general integer problems. In addition, several sources indicate that a synthesis of branch and bound with cutting plane should merit attention [1,3,18].

The problem to be addressed is a continuous pure PILP of the following form:

$$\begin{aligned} \max \quad & \sum_{j=1}^n c_j x_j \\ \text{S.T.} \quad & \sum_{j=1}^n a_{ij} x_j \leq b_i + \theta d_i \quad 1 \leq i \leq m \\ & x_j \geq 0, \text{ integer} \quad 1 \leq j \leq n, \end{aligned} \quad (1.1)$$

with $0 \leq \theta \leq 1$ and d_i integer constants, for $1 \leq i \leq m$.

Pioneering work in PILP by Roodman [16,17] and Piper and Zoltners [14,15] using the implicit enumeration method was restricted to the 0-1 case. Later, Marsten and Morin [11] considered the continuous parametric right-hand-side case using a modified branch and bound method. They had to maintain a list of all candidate problems and two bounding functions. To solve parametric problems, they extended their bounding test to compare the two bounding functions rather than a single lower bound and a single upper bound. Their work also was restricted to the 0-1 case.

In addition, Marsten and Morin [10] presented a hybrid algorithm combining dynamic programming with branch and bound. Dynamic programming provided another dimension, fathoming-by dominance, to the branch and

bound technique. It also eliminated branch and bound backtracking by controlling the search strategy. Holm and Klein [8,9] developed a method for general PILP and mixed PILP based on cutting plane algorithms using Gomory cuts. They saved information from the source rows of their cuts to be applied to any changed problems. They did not attempt to solve several members simultaneously, as might be necessary in the face of uncertain right-hand-sides. Bailey [1], on the other hand, did enumerate all family members using the transformation theorem of Noltemeier [13]. Bailey [1] employed a cutting plane technique on pure integer PILPs with a solution success rate near 50%. He felt the largest deterrent to problem solution with cutting plane techniques was round-off error, which increased with problem dimension and the number of different right-hand-sides.

A synthesis of the better features of branch and bound with cutting plane indicates promise. The enumeration of all family members in combination with strong fathoming tests affords the possibility of one member yielding a stronger lower bound which in turn causes another member to be fathomed.

With the dual cutting plane technique, the first feasible solution is the optimal solution. No suboptimal solutions are generated in the process. On the other hand, in a maximization problem utilizing branch and bound, any lower bound could serve as a sub-optimal feasible solution in the event of premature termination. In addition, the linear programming relaxation yields upper bounds to strengthen the fathoming tests. Unfortunately, there is a storage problem as the size of the branch and bound tree increases. Thus, the fusion of branch and bound with cutting plane could lead to smaller storage requirements and more effective cuts

than required by pure branch and bound and cutting plane methods, respectively.

The main objective of this paper is to present a branch and bound algorithm for the PILP problem where right-hand-side changes are known a priori. A cutting plane option is available to decrease the number of nodes required by the branch and bound tree. Such an algorithm is needed, especially with uncertainty in right-hand-side data, to generate a set of related family members and observe the effect of changes in the right-hand-sides.

2. Development of the Synthesized Algorithm

In this section, detail steps are provided for the synthesized PILP algorithm given in section 3. For reference, each description is followed by its corresponding step number. The notation used follows that of Taha [18] with minor adaptations to indicate specific family members.

The problem to be considered is a monotone increasing family where the feasible regions are nested supersets of each other. Starting with a problem of the form (1.1), with $0 \leq \theta \leq 1$ and $d_i \geq 0$, the θ 's are determined which make the right-hand-sides integer.

The continuous pure PILP problem is transformed into a set of finite discrete ILP problems of the form:

$$\begin{aligned} \max \quad & \sum_{j=1}^n c_j x_j \\ \text{S.T.} \quad & \sum_{j=1}^n a_{ij} x_j \leq b_i + t_i^k \quad 1 \leq i \leq m \\ & x_j \geq 0, \text{ integer} \quad 1 \leq j \leq n. \end{aligned}$$

For each θ , append the greatest integer of $b + \theta d$ to the original tableau as its associated family member's right-hand-side (RHS). Thus, it is desirable to solve simultaneously K problems with $k=1,2,\dots,K$. (Step 1)

Apply the column dual simplex method. Pivot until the family member with $\theta = 0$ is at continuous linear programming (LP) optimality. Save the current LP results, noting that the LP optima are upper bounds (UBs) on the integer optima. (Step 2)

All family members may not reach LP optimality simultaneously. Therefore, it is possible that some members may have different bases at their LP optimality. This situation occurs when some constraints are inactive for certain family members and active for others. To resolve this situation, family members with the same optimal LP basis are grouped together. The currently infeasible members are saved for later consideration. To avoid carrying along members not currently under consideration, the problem is split into feasible and infeasible members. The algorithm works with the feasible members only until their optimal integer solutions are obtained. Then, the algorithm returns to the infeasible members and continues pivoting until LP optimality is attained for the previously infeasible members. (Step 3)

Still considering the member with $\theta = 0$, use implicit enumeration to solve the 0-1 unit hypercube problem [18] around the LP solution to obtain a lower bound (LB) which holds for all family members. If the implicit enumeration results in an infeasible solution, decrease all nonzero LP solutions values by one unit and re-solve the embedded 0-1 problem. The implicit enumeration routine is invoked a maximum of two times. Large problems have many vertices to consider and therefore may not warrant the expense of implicit enumeration. (Step 4)

Nodes are always examined before branching to see if any members can be fathomed by updated LB values. Integer solutions in the current tableau are candidates for improved LBs. Note that a node is fathomed only when all of its members have been fathomed. (Step 6)

Next, consider the branch and bound portion of the algorithm. Use

a variation of Tomlin's P^* penalties to choose the branch node and variable. A penalty gives a lower bound on the exact amount of change in the objective function for the particular member i that yielded the penalty P^* . Thus, it is possible to fathom that particular member if the penalty will decrease the objective function value at the current node, $UB^q(\theta_1)$, to an amount less than or equal to the current LB (i.e. if $UB^q(\theta_1) - P^* \leq LB(\theta_1)$). Branch on the node with the largest P^* value [19,5]. Let f_k be the minimum of the fractional parts of the RHSs for variable k . The minimum fraction allows for maximizing the smallest amount of change for P^* . Let $P^* = \max \{P_k^*\}$ where $f_k > 0$, and $P_k^* = \min\{(z_j - c_j)f_k/\alpha_{kj}^*\}v^k$ where $j \in NB$, the set of subscripts for the nonbasic variables. α_{kj}^* is calculated as in Taha [18].

The variation from Tomlin's method is in choosing the branch direction. Examine the element α_{kj} which yielded the P^* penalty. If $\alpha_{kj} < 0$, take the up branch first. Otherwise, take the down branch. If it is not possible to take both branches (i.e. if one tableau has an all positive row so that no variable can enter the basis), do not save the current tableau. (Step 7)

When adding an up or down Dakin [2] branch, all fractional values remain the same except for the RHS values. The parametric up and down constraints are as follows, with a different RHS for every member i :

$$-\sum_{j \in NB} \alpha_{kj} x_j \leq -f_k^i \quad (\text{down})$$

$$\sum_{j \in NB} \alpha_{kj} x_j \leq f_k^i - 1 \quad (\text{up})$$

Thus, an up (down) branch on variable k can be added to all members simultaneously. The specific form of the branch is $x_k \leq [x_{ki}^*]$ or $x_{ki} \geq [x_{ki}^*] + 1$. For example, even though a fixed constraint such as $x_2 \geq 2$ is not added to every family member, the up constraint for variable x_2 is added to all

members. (Step 8)

Once feasibility has been attained for a member, check for an integer solution. Check after each branch and each cut. If the solution is integer, use it to improve LBs for all larger members. Then fathom each member i whose LP optimum is within one unit of its lower bound (i.e. if $UB(\theta_i) + E < LB(\theta_i) + 1$, where E is an epsilon factor). Thus, the optimal integer solution for θ_i has been found. Note that a LB for one member can be used to update a LB for a member with a larger θ and subsequently fathom that new member. (Step 9)

Also check to see if the current node can yield a better feasible solution than the current LB for a particular member i . If $UB^q(\theta_i) + CRIT < LB(\theta_i) + 1$, where $CRIT$ is an epsilon factor, there is no feasible solution better than the current LB for member i . The member is fathomed for the current node and is omitted from all children of the current node. (Step 10)

All members are examined after each branch. If an unfathomed member is infeasible, the cutting plane method is invoked to cut toward feasibility. Thus, infeasible solutions are cut off while moving toward feasibility. Cuts are based on the first feasible unfathomed member. Whenever no feasible unfathomed member is found, pivot by invoking the column dual simplex method. Continue examining nodes and members until the entire tree has been traversed and all optimal integer solutions have been found. (Step 11)

When all nodes have been fathomed, check to see if splitting occurred on the original LP solution. If it did not, then all family member's integer solutions have been found. Otherwise, solve the original LP tableau for the remaining members (Step 12)

The cutting plane option allows cuts to be applied to a node to obtain smaller UBs for that node. If the cutting plane option is desired, add a finite number of cuts before branching on a node. Add dual fractional

Gomory [6] cuts based on the first feasible unfathomed member. Let f_k be the fractional part of that first feasible unfathomed member for variable k . Then $P = \max\{P_k\}$ where $f_k > 0$, and $P_k = \min\{(z_j - c_j)f_k/f_{kj}\} \forall k$ where $j \in NB$. Again, pivot rather than cut, if the first unfathomed member is infeasible. If the node becomes fathomed, return to step 6 to examine the other nodes before branching. (Step 5)

The fathoming tests used along with frequent examination of the family members cause members to be fathomed quickly. In addition, family members can provide better lower bounds and fathom closely related members.

3. SPILP Algorithm

STEP 1: Calculate θ s and set up right-hand-sides.

STEP 2: Find the LP solution. Save results as UBs on members.

STEP 3: Split problem if not all members are feasible.

STEP 4: Use implicit enumeration to solve the LP problem for the unit hypercube around the LP solution for $\theta = 0$. Save results as LBs on all members. If results are infeasible, adjust the LP values and re-solve. Do not repeat.

STEP 5: If the cutting plane option is desired, add the indicated number of cuts.

STEP 6: Examine nodes. Check for integer solutions. (For detail, see steps 9 and 10.) Fathom members when possible. Fathom the node when all family members at that node have been fathomed. When all nodes have been fathomed, go to step 12.

STEP 7: Branch on node with max P^* . Use Tomlin's penalties to choose the node and variable. Copy node only if it is possible to take both an up and down branch from that node.

STEP 8: Add the up or down constraint and pivot until the first unfathomed member is feasible.

- STEP 9: Check for an integer solution. In integer, use it to improve LBs. For each member, if $UB(\theta_1) + E < LB(\theta_1) + 1$, the member is completely fathomed. The optimal integer solution has been found.
- STEP 10: Check for no feasible solution better than the current $LB(\theta_1)$. For the current member, if $UB^q(\theta_1) + CRIT < LB(\theta_1) + 1$, the member is fathomed for the current node.
- STEP 11: Increment to the next unfathomed member. If feasible, go to step 9. Otherwise, add cuts based on the first feasible unfathomed member. If the first unfathomed member is infeasible, then pivot. Go to step 9. When all members have been examined, go to step 5.
- STEP 12: Check to see if all members have been solved. If so, print all optimal solutions and stop. Otherwise, solve the original LP for the remaining members. Update flags to indicate a new branch and bound tree for the unsolved members. Set number of nodes to one. Go to step 5.

5. Computational Results

The SPILP algorithm was implemented on an Amdahl 470/V7 (Model u34), an IBM compatible machine, under OS/VS2 R3.7 (MVS) with JES2. The first fifteen test problems used the FORTRAN IV G1 compiler and were run with a load module. Thus, the CPU times given are for data input-output and execution only. The IBM 6 problem was not run from the load module. Consequently, its CPU times include some load time. All compile and link times are omitted.

A special storage framework was utilized to suit the occurrence of partial member fathoming without complete node fathoming. The rows of the simplex tableaus were stored as columns, with all tableaus attached. As members were fathomed at specific nodes, flags were set to omit fathomed

members from future tableaus and to decrease node size. In addition, the original tableau was stored as the first tableau.

A solution was considered to be integer whenever the objective function and all solution values were within a tolerance of 0.0005. Furthermore, E , the epsilon criterion given in the algorithm, was set at 10^{-7} ; and CRIT was set at 0.0005.

The SPILP algorithm was tested on ten fixed-charge problems of Haldi [7], where up to half of the original variables were binary. The 0-1 restrictions were omitted since they were not requisite for problem solution. In addition, six IBM test problems were solved. The core size required for matrix storage for the Haldi problems and the first five IBM problems was 488K bytes with a node storage matrix of size 500×100 . The largest of these problems had 15 original variables and 15 constraints. The largest problem attempted, IBM 6, required a core size of 1656K bytes for matrix storage with a node storage matrix of size 2000×100 .

Two different types of problems were solved. The first, including Haldi 1 through 4 and 9 and IBM 1 through 6 (excluding 3), have fairly small right-hand-sides (21 or less). Changes of up to 50% of each of the RHSs were examined. The second type of problem, including Haldi 5 through 8 and 10 and IBM 3 have much larger RHSs (from 80 to 210). Changes of up to 10% of the largest RHS were examined to compare the branch and bound method with and without the cutting plane option. All components of the d-vector were equal to reduce the number of family members generated. The computational results are given in Tables 1, 2 and 3.

6. Conclusions

This paper has presented the synthesized parametric integer linear programming (SPILP) algorithm for general ILP problems. It combined branch and bound and cutting plane methods, whereas previous general methods used

cutting planes [1,8,9] or combined branch and bound with dynamic programming [10].

The performance data collected from the solution of the test problems by the SPILP algorithm indicate that it is a highly successful algorithm. On occasion the procedure for solving a single problem proved to be inferior to expanding the problem to a family of problems and solving the family parametrically. With several members to solve, better penalties were chosen and easy-to-solve members helped fathom the more difficult-to-solve members. Furthermore, the few problems unsolved without the cutting plane option were readily solved with the cutting plane option. Thus, all problems attempted were solved by the SPILP algorithm. In general, the use of the cutting plane option before branching in the SPILP algorithm decreased the number of nodes examined and the number of nodes stored at one time and thus required less storage than without the use of the cutting plane option. However, the number of simplex iterations generally increased. Experimentation with the number of cuts to apply whenever an infeasible solution was found and the number of cuts to apply before branching on the larger problems further increased the efficiency of the algorithm.

Implicit enumeration and penalties were applied to each problem separately and in most cases gave greatly improved lower bounds as well as improved node and variable selections for a resultant reduction in the number of nodes.

Finally, the members from two parametric problems were solved independently and the results compared with the parametric family results. The CPU time increased significantly when members were solved separately rather than parametrically. Thus, the parametric solution method was the preferable method.

In conclusion, our results look extremely promising and indicate that the SPILP algorithm is an efficient algorithm.

Table 1

Results for Haldi Problems with Small Right-Hand-Sides

Problem	Split LP Tableau	Cutting Plane Option	Number of Nodes	Number of Iterations	Max. # Nodes Stored	% d is of b	Number of Members	CPU Time (sec.)
Haldi 1	no	yes	3	17	3	50%	16	.20
Haldi 1	no	no	7	12	4	50%	16	.21
Haldi 1	no	no	7	12	4	0%	1	.13
Haldi 2	no	yes	3	16	3	50%	17	.21
Haldi 2	no	no	8	13	4	50%	17	.22
Haldi 2	no	no	3	7	3	0%	1	.10
Haldi 3	no	yes	3	20	3	50%	11	.17
Haldi 3	no	no	6	10	4	50%	11	.17
Haldi 3	no	no	5	9	4	0%	1	.11
Haldi 4	no	yes	2	21	3	50%	8	.16
Haldi 4	no	no	8	13	4	50%	8	.18
Haldi 4	no	no	8	13	4	0%	1	.14
Haldi 9	yes	yes	11	55	5	50%	6	.33
Haldi 9	yes	no	20	41	6	50%	6	.36
Haldi 9	no	no	7	14	4	0%	1	.15

Table 2

Results for IBM Problems with Small Right-Hand-Sides

Problem	Split LP Tableau	Cutting Plane Option	Number of Nodes	Number of Iterations	Max. # Nodes Stored	% d is of b	Number of Members	CPU Time (sec.)
IBM 1	yes	yes	4	16	4	50%	3	.17
IBM 1	yes	no	6	12	5	50%	3	.17
IBM 1	no	no	1	6	2	0%	1	.09
IBM 2	yes	yes	4	16	4	50%	3	.20
IBM 2	yes	no	6	12	5	50%	3	.19
IBM 2	no	no	1	6	2	0%	1	.11
IBM 4	yes	yes	27	171	8	50%	5	6.60
IBM 4	yes	no	60	190	12	50%	5	6.98
IBM 4	no	no	1	14	2	0%	1	5.38
IBM 5	no	yes	36	387	8	50%	7	4.42
IBM 5	no	no	19	345	9	50%	7	3.96
IBM 5	no	yes	93	413	8	0%	1	4.95
IBM 6	yes	yes	415	13681	46	50%	11	248.61*
IBM 6	no	yes	91	897	24	0%	1	17.54*

*Problem not run from load module.

Table 3

Results for Problems with Large Right-Hand-Sides

Problem	Split LP Tableau	Cutting Plane Option	Number of Nodes	Number of Iterations	Max. # Nodes Stored	% d is of b	Number of Members	CPU Time (sec.)
Haldi 5	no	yes	7	33	4	10%	19	.40
Haldi 5	no	no	8	13	4	10%	19	.27
Haldi 5	no	no	8	13	4	0%	1	.14
Haldi 6	no	yes	5	38	3	10%	22	.38
Haldi 6	no	no	6	10	4	10%	22	.24
Haldi 6	no	no	6	10	4	0%	1	.12
Haldi 7	no	yes	1	4	2	10%	19	.13
Haldi 7	no	no	2	4	3	10%	19	.13
Haldi 7	no	no	1	3	2	0%	1	.08
Haldi 8	no	yes	1	4	2	10%	22	.13
Haldi 8	no	no	2	4	3	10%	22	.14
Haldi 8	no	no	1	3	2	0%	1	.08
Haldi 10	no	yes	28	428	11	10%	12	2.83
Haldi 10	no	no	16	98	6	10%	12	1.20
Haldi 10	no	no	9	38	3	0%	1	.68
IBM 3	no	yes	23	191	8	10%	21	.66
IBM 3	no	no	27	139	8	10%	21	.75
IBM 3	no	no	14	22	4	0%	1	.18

REFERENCES

- [1] M. G. Bailey and B. E. Gillett, "Parametric integer programming analysis: A contraction approach", Journal of the Operational Research Society, 31 (1980) 257-262.
- [2] R. J. Dakin, "A tree-search algorithm for mixed integer programming problems", Computer Journal 8 (1965) 250-255.
- [3] A. M. Geoffrion and R. E. Marsten, "Integer programming algorithms: A framework and state-of-the-art survey", Management Science 18 (1972) 465-491.
- [4] A. M. Geoffrion and R. Nauss, "Parametric and postoptimality analysis in integer linear programming", Management Science 23 (1977) 453-466.
- [5] R. E. Gomory, "An algorithm for the mixed integer problem", RM-2597, RAND Corporation, Santa Monica, California (1960).
- [6] R. E. Gomory, "Outline of an algorithm for integer solutions to linear programs", Bulletin of the American Mathematical Society 64 (1958) 275-278.
- [7] J. Haldi, "25 integer programming test problems", Working paper No. 43, Stanford University (December 1964).
- [8] S. Holm and D. Klein, "Parametric analysis for integer programming problems", Working paper 76-4, Florida International University (February 1976).
- [9] D. Klein and S. Holm, "Integer programming post-optimal analysis with cutting planes", Management Science 25 (1979) 64-72.
- [10] R. E. Marsten and T. L. Morin, "A hybrid approach to discrete mathematical programming", Working paper OR 051-76, Massachusetts Institute of Technology (March 1976).

- [11] R. E. Marsten and T. L. Morin, "Parametric integer programming: the right-hand-side case", Working paper OR 050-76, Massachusetts Institute of Technology (March 1976).
- [12] R. M. Nauss, "Parametric integer programming", Ph.D. dissertation, University of California, Los Angeles (1974).
- [13] H. Noltemeier, "Sensitivitätsanalyse bei diskreten linearen Optimierungsproblemen", in M. Beckmann and H. P. Kunzi, eds., Lecture Notes in Operations Research and Mathematical Systems, No. 30 (Springer-Verlag, New York, 1970).
- [14] C. J. Piper and A. A. Zoltners, "Implicit enumeration based algorithms for postoptimizing zero-one programs", Naval Research Logistics Quarterly 22 (1976) 791-809.
- [15] C. J. Piper and A. A. Zoltners, "Some easy post-optimality analysis for zero-one programming", Management Science 22 (1976) 759-765.
- [16] G. M. Roodman, "Postoptimality analysis in integer programming by implicit enumeration: the mixed integer case", Naval Research Logistics Quarterly 21 (1974) 595-607.
- [17] G. M. Roodman, "Postoptimality analysis in zero-one programming by implicit enumeration", Naval Research Logistics Quarterly 19 (1972) 435-447.
- [18] H. A. Taha, Integer Programming: Theory, Applications, and Computations (Academic Press, New York, 1975).
- [19] J. A. Tomlin, "An improved branch and bound method for integer programming", Operations Research 19 (1971) 1070-1075.