DISCUSSION PAPER NO. 93

THE STRENGTHENED GOMORY MIXED INTEGER CUT FOR THE ALL INTEGER PROGRAM

By

Avinoam Perry*

July 1974

* Assistant Professor, Graduate School of Management, Northwestern University Evanston, Illinois.

ABSTRACT

A strengthened mixed integer cut for solving the all integer program may be derived from a combination of the Gomory cut [3] and the modified Dantzig cut [1]. Experiments with this method employing three cut selection rules indicate a strong potential of this approach. This paper presents a derivation of the strengthened mixed integer cut, a description of the three suggested rules for selecting a cut among several alternatives, and a summary of computational experience.

I. Introduction

In this paper we derive a strenthened mixed integer cut for solving the all integer program. This cut is a combination of Gomory's cut [3] and the modified Dantzig cut [1]. Experiments with this method employing three cut selection rules indicate a strong potential of this approach. This paper presents a derivation of the strengthed mixed integer cut, a description of the three suggested rules for selecting a cut among several alternatives, and a summary of computational experience. To facilitate the presentation we review the derivation of the Gomory cut [3] and a different cut we call the complementary Gomory cut. We show that a combination of these two cuts produces the modified Dantzig cut [1]. We also show that a combination of these three cuts yields a strengthened mixed integer cut.

II. Deriviation of the Strengthened Mixed Integer Cut

a. The Gomory Cut [3]

Consider the following L.P. problem

$$\max \quad \sum_j c_j x_j$$

$$\text{s.t.} \quad \sum_j B_{ij} x_j + t_i = B_{io}$$

$$x_j,\ t_i = \text{Integer}$$

The optimum solution to this problem has the following form:

$$(1) \quad x_i = B_{io} + \sum_k B_{ik}\, t_{ik} - \sum_{k'} B_{ik'}\, t_{ik'}$$

where: x_i is a basic variable

t_i is a nonbasic variable

B_{io} is the value of the basic variable x_i at the current optimum solution

k is the set of all negative coefficients of the nonbasic variables t_{ik}

k' is the set of all positive coefficients of the nonbasic variables $t_{ik'}$

B_{ik} is the coefficient of the nonbasic variable t_{ik}

$B_{ik'}$ is the coefficient of the nonbasic variable $t_{ik'}$.

$(x_i,\ t_{ik},\ B_i,\ B_{ik},\ B_{ik'} \geq 0)$

(1) may be redefined as follows:

$$(2) \quad x_i = [B_{io}] + b_{io} + \sum_k ([B_{ik}] + 1)t_{ik} - \sum_k (1-b_{ik})t_{ik} - \sum_{k'} [b_{ik'}]t_{ik'} - \sum_{k'} b_{ik'}\, t_{ik'}$$

where: $[B_{io}]$, $[B_{ik}]$, $[B_{ik'}]$ are the integer part of B_i, B_{ik}, $B_{ik'}$ respectively, and b_i, b_{ik}, $b_{ik'}$ are the fractional part of B_{io}, B_{ik}, $B_{ik'}$, respectively. $(b_{io},\ b_{ik},\ b_{ik'} \geq 0)$.

From the integrality requirement on t_{ik} and $t_{ik'}$ it follows that:

(3) $[B_{io}] + \sum_{k}([B_{ik}] + 1)t_{ik} - \sum_{k'} [B_{ik'}]t_{ik'}$ is an integer.

From the integrality requirement on x_i it follows that:

(4) $b_{io} - \sum_{k} (1-b_{ik})t_{ik} - \sum_{k'} b_{ik'}t_{ik'}$ is an integer.

but $0 < b_{io} < 1$

$0 < b_{ik} < 1 \qquad 0 < b_{ik'} < 1$

$0 < 1-b_{ik} < 1 \qquad 0 < 1-b_{ik'} < 1$

Hence, (4) may be represented as:

(5) $b_{io} - \sum_{k} (1-b_{ik})t_{ik} - \sum_{k'} b_{ik'}t_{ik'} \leq 0$

which is the Gomory cut for the all integer program.

b. The Complementary Cut

A redefinition of (1) yields the following expression of the current solution to the L. P. problem.

(6) $x_i = [B_{io}] + b_{io} + \sum_{k} [B_{ik}]t_{ik} + \sum_{k} b_{ik}t_{ik} - \sum_{k'}([B_{ik'}] + 1)t_{ik'}$

$+ \sum_{k'} (1-b_{ik'})t_{ik'}$

From (6) it follows that:

(7) $b_{io} + \sum_{k} b_{ik}t_{ik} + \sum_{k'} (1-b_{ik'})t_{ik'}$

is an integer.

Hence:

(8) $b_{io} + \sum_{k} b_{ik}t_{ik} + \sum_{k'} (1-b_{ik'})t_{ik'} \geq 1$

is a valid cut.

c. The Modified Dantzig Cut

Addition of the Gomory cut and its complement yields the modified

Dantzig cut.

Gomory:	$-b_{io} + \sum_{k} (1-b_{ik})t_{ik} + \sum_{k'} b_{ik'}t_{ik'}$	≥ 0
Complement:	$b_{io} + \sum_{k} b_{ik}t_{ik} + \sum_{k'} (1-b_{ik'})t_{ik'}$	≥ 1
Modified Dantzig	$0 + \sum_{k} t_{ik} + \sum_{k'} t_{ik'}$	≥ 1

d. The Convexity Cut Approach [2]

The Gomory cut, its complement, and the modified Dantzig cut may be represented as a special case of the general convexity cut:

$$(9) \quad \sum_{j} (1/t_j^*)t_j \geq 1$$

where: $1/t_j^*$ is the coefficient of the nonbasic variable t_j, and t_j^* may be interpreted as a measure of the depth of the cut along the j's dimension.

The convexity form of the Gomory cut is formulated as:

$$(10) \quad \sum_{k} \frac{1-b_{ik}}{b_{io}} t_{ik} + \sum_{k'} \frac{b_{ik'}}{b_{io}} t_{ik'} \geq 1$$

and the convexity form of the Complementary cut is formulated as:

$$(11) \quad \sum_{k} \frac{b_{ik}}{1-b_{io}} t_{ik} + \sum_{k'} \frac{1-b_{ik'}}{1-b_{io}} t_{ik'} \geq 1$$

where:

$$t_{ik}^* = \frac{b_{io}}{1-b_{ik}} \quad \text{and} \quad t_{ik'}^* = \frac{b_{io}}{b_{ik'}} \quad \text{for the Gomory cut}$$

$$t_{ik}^* = \frac{1-b_{io}}{b_{ik}} \quad \text{and} \quad t_{ik'}^* = \frac{1-b_{io}}{1-b_{ik'}} \quad \text{for the complementary cut}$$

It follows that if a coefficient t_i^* derived from (10) is less than 1, then its complement derived from (11) is greater than 1 and, therefore the complementary cut is deeper along that dimension.

It is possible to derive a cut which contains coefficients $t_i^* \geq 1$ only, by an arbitrary selection of coefficients t_i^*, from either Gomory

or its complement, in such a way that all t_i^*'s are greater than or equal to 1. (This approach implies that if a nonbasic variable t_i becomes positive in the final integer optimum solution, then, it must be greater than or equal to 1. This requirement is not implied directly by the Gomory or the complementary cut.)

e. The Strengthened Mixed Integer Cut Of The All Integer Program

Let $Q \subset k$ and $R \subset k$ and let $Q \cap R = \phi$. Let $Q' \subset k'$ and $R' \subset k'$ and let $Q' \cap R' = \phi$ where k and k' are the sets defined in (1). We can express (1) as:

$$(12) \quad x_i = B_{io} + \sum_Q B_{iQ}t_{iQ} + \sum_R B_{iR}t_{iR} - \sum_{Q'} B_{iQ'}t_{iQ'} - \sum_{R'} B_{iR'}t_{iR'}$$

The assignment of a variable $t_i \in k$ to either Q or R, or the assignment of a variable $t_i \in k'$ to either Q' or R' is arbitrary.

(12) may be expanded to the following form:

$$(13) \quad x_i = [B_{io}] + b_{io} + \sum_Q [B_{iQ}]t_{iQ} + \sum_Q b_{iQ}t_{iQ} + \sum_R ([B_{iR}] + 1)t_{iR}$$
$$- \sum_R (1-b_{iR})t_{iR} - \sum_{Q'} [B_{iQ'}]t_{iQ'} - \sum_{Q'} b_{iQ'}t_{iQ'}$$
$$- \sum_{R'} ([B_{iR'}] + 1)t_{iR'} + \sum_{R'} (1-b_{iR'})t_{iR'}$$

From the integrality requirements on all x_i's and t_i's it follows that:

$$(14) \quad b_{io} + \sum_Q b_{iQ}t_{iQ} - \sum_R (1-b_{iR})t_{iR} - \sum_{Q'} b_{iQ'}t_{iQ'} + \sum_{R'} (1-b_{iR})t_{iR}$$

is an integer which must be either ≥ 1 or ≤ 0.

If (14) is ≥ 1 then:

$$(14a) \quad b_{io} + \sum_Q b_{iQ}t_{iQ} + \sum_{R'} (1-b_{iR'})t_{iR'} \geq 1 \qquad \text{is true}$$

If (14) is ≤ 0 then:

$$(14b) \quad b_{io} - \sum_R (1-b_{iR})t_{iR} - \sum_{Q'} b_{iQ'}t_{iQ'} \leq 0 \qquad \text{is true}$$

The equivalent convexity cut form of (14a) is:

$$(14c)\quad \sum_{Q} \frac{b_{iQ}}{1-b_{io}}\, t_{iQ} + \sum_{R'} \frac{1-b_{iR'}}{1-b_{io}}\, t_{iR'} \geq 1$$

The equivalent convexity cut form of (14b) is:

$$(14d)\quad \sum_{R} \frac{1-b_{iR}}{b_{io}}\, t_{iR} + \sum_{Q'} \frac{b_{iQ'}}{b_{io}}\, t_{iQ} \geq 1$$

At least one of (14c) or (14d) must be true for (14) to be true, and since by definition (14c) and (14d) are ≥ 0 we have:

$$(15)\quad \sum_{Q} \frac{b_{iQ}}{1-b_{io}}\, t_{iQ} + \sum_{R} \frac{1-b_{iR}}{b_{io}}\, t_{iR} + \sum_{Q'} \frac{b_{iQ}}{b_{io}}\, t_{iQ'} + \sum_{R'} \frac{1-b_{iR'}}{1-b_{io}}\, t_{iR} \geq 1$$

which is the strengthened mixed integer cut of the all integer program.

We may now summarize the derivation of the strengthened cut.

1. Solve the L.P. by ignoring the integrality requirements.
2. If the current optimum is all integer, stop, otherwise go to 3.
3. Derive a Gomory cut in the convexity cut form: $\sum_{j} (1/t_j^*)t_j \geq 1$

   where: $t_j^* = b_o/b_j$ for all $j \in k'$

   $t_j^* = b_o/(1-b_j)$ for all $j \in k$
4. If there is any $t_j^* < 1$ replace it by its complement. For example:

   if $b_o/b_j < 1$ replace it by $(1-b_o)/(1-b_j) > 1$

   if $b_o/(1-b_j) < 1$ replace it by $(1-b_o)/b_j > 1$
5. The new cut is $\sum_{j} (1/t_j^*)t_j \geq 1$ where $t_j^* \geq 1$.

The strengthened cut for the all integer program is a special case of the strengthened cut for the mixed integer program. Since the optimum value of the objective function is an integer, and the lexicographic dual simplex is applied to retain primal feasibility; a method employing the strengthened cut for the all integer program must converge in a finite number of steps. (The mathematical proof is the same as the one for the Gomory mixed integer cutting plane method [4].)

III. Experimental Rules for Selecting Cuts Among Several Alternatives

Suppose there are two possible source rows from which we derive two constraints

$$\sum_j (1/t^*_{ij}) t_{ij} \geq 1$$

$$\sum_j (1/t^*_{kj}) t_{kj} \geq 1$$

respectively. Comparing these two constraints, if we find that for every j, $t^*_{ij} \geq t^*_{kj}$, then we select row (i) as the source row because the cut derived from row i is deeper than the cut derived from row k. Row k is dominated by row i, therefore, we may exclude it from our considerations. Suppose there is one row (e) for which there are several $t^*_{ij} \geq t^*_{ie}$, but, at the same time, there are several $t^*_{ij} \leq t^*_{ie}$. The decision regarding the better possible source row in this case is much more complicated and, in fact, we have to make decisions which affect the short run only, namely, our decision is based on the expected outcome of the following pivot step. This decision procedure is outlined in the following algorithm which is referred to as Version 1.

Version 1

Step #1. From every possible source row derive a constraint of the type:

$$\sum_j (1/t^*_j) t_j \geq 1$$

These constraints are represented in a matrix form $T = 1/t^*_{ij}$

Step #2. Search across the rows to find $\max_i t^*_{ij}$ for every column in matrix T.

Step #3. Derive a dummy constraint

$$\sum_j \max_i (1/t^*_{ij}) t_j \geq 1$$

Step #4. Search for $\min_j [(\max_i t^*_{ij}) \cdot (z_j - c_j)]$

Step #5. Select the one source row which has the property of step 4.

We may modify version #1 to include some long run considerations. This procedure is outlined in the following algorithm which is referred to as version two.

Version 2

Apply the first 5 steps of version #1.

Step #6. For every possible candidate (row) in matrix T count the number of zero coefficients.

Note: a zero coefficient implies $1/t^*_j = 0$ or $t^*_j = \infty$

Step #7. Select the one row which has the largest number of zero coefficients.

Step #8. If the row selected in step 7 has more zero coefficients than the row selected in step 5, replace the new row with the old row, otherwise select the row chosen in step 5.

Version 3

Version #3 uses a heuristic rule which has a relatively high probability of selecting deep cuts among several alternatives and, nevertheless, employs less computer space and less searching time.

Define: $f_j = (b_j)$ or $(1-b_j)$

then: $t^*_j = (1-b_o)/f_i$ or b_o/f_i

If the numerator $1-b_{io}$ or b_{io} is a relatively large fractional number then, on the average, t^*_{ij} will be large.

Based on this assumption we derive the following procedure:

Step #1. Select those restricted basic variables for which $b_o \neq 0$ and calculate

$$\max_i \left| b_{io} - 0.5 \right|$$

Step #2. The one row which fulfills the property of step 1 is selected as the source row.

Computational results from these three versions are presented in the next section.

Note: It is possible to use combinations of the above versions and to come up with more than just three ways of selecting source rows. This topic may be treated as a research project by itself.

IV. Computational Experience

The strengthened mixed integer cut for solving the all integer program was tested employing three codes: version 1 (V1), version 2 (V2), version 3 (V3).

The problems used for testing purposes are those developed and reported by J. Haldi [5] to test the LIP1 computer code. Further comparisons were made with respect to Trauth and Woolsey's [8] results with the LIP1, IPM3, and LIP2-2 codes.

The results are presented in tables no. 1 and 2 and are generally self-explanatory. All times were computed from the first executed instruction of the program to the end of the minimum output needed to interpret the results. All times are given in seconds. The word "iteration" refers to a single matrix pivot operation. All programs were run on the CDC6400 computer.

The first nine problems in the computational summary table are Haldi's fixed charge problems. They are followed by the IBM integer programming test problems [5]. Versions 1, 2, and 3 are denoted as V1, V2, and V3 respectively.

Table 1

Fixed Charge Problems

Code	V1		V2		V3		IPM3		LIP1		ILP2-2	
Problem	Time	Iter.	Time	Iter.	Time	Iter.	Time	Iter.	Time	Iter.	Time	Iter.
1	1.789	20	1.902	20	1.652	19	3.117	54	1.833	24	0.852	36
2	1.330	13	1.401	13	3.103	39	3.767	81	1.350	15	0.935	47
3	1.512	14	1.430	11	1.783	21	3.033	37	1.883	26	1.384	104
4	0.901	6	0.966	6	0.887	6	4.100	91	1.483	18	0.674	18
5	2.576	18	2.414	16	6.780	60		+7000	9.012	158		+7000
6	2.006	13	2.819	24	6.911	.77		+7000	7.567	123	3.273	311
7	2.548	18	2.497	16	3.007	37		+7000	7.833	159		+7000
8	2.241	14	2.310	14	2.475	21		+7000	6.417	126	3.033	306
9	1.278	9	1.282	9	1.206	9	5.183	118	3.233	42	3.598	293

Table 2

Haldi's "IBM" Problems

Code	V1		V2		V3		IPM3		LIP1		ILP2-2	
Problem	Time	Iter.	Time	Iter.	Time	Iter.	Time	Iter.	Time	Iter.	Time	Iter.
1	1.503	8	1.512	8	1.488	8	2.300	8	1.866	11	1.087	11
2	2.711	23	2.785	23	2.6698	23	2.833	17	3.016	32	1.149	15
3	2.579	38	2.617	41	2.497	36	2.633	22	2.866	53	0.621	14
4	13.761	106	8.882	40	24.562	258	5.933	24	11.666	73	3.079	18
5	45.517	212	31.566	149	71.536	361	51.600	1144	66.483	351	26.184	842
9	281.174	409	351.002	841	508.066	982	633.313	6758	473.100	953	75.121	1105

V. Conclusions

Dual functional cutting plane methods are, in general, very sensitive to truncation and rounding errors. The final simplex tableau from which cuts are generated, consists of the following matrices and vectors:

$$x_B + B^{-1}At = B^{-1}b$$

$$c_B x_B + \sum_j (c_B B^{-1} A_j - c_j) = c_B B^{-1} b$$

Whenever the basis matrix B is inverted there is a roundoff error associated with its product B^{-1}. Cuts of the form $\sum_j (1/t_j^*) t_j \geq 1$ are derived from the current L.P. tableau where:

$$t_j^* = b_o/b_j \text{ or } (1-b_o)/b_j \text{ or } b_o/(1-b_j) \text{ or } (1-b_o)/(1-b_j)$$

where b_o is the fractional part of $B_i^{-1}b$ or $c_B B^{-1} b$ and b_j is the fractional part of $B^{-1}A_j$ or $c_B B^{-1} A_j - c_j$; therefore both b_o and b_j contain rounding errors. Every cutting plane, once derived, stays in the L.P. tableau as an additional constraint at least until its associated slack variable becomes positive. When new cuts are derived, some of them may be generated from an older cut which already contains rounding errors. Consequently, the rounding error is building up very quickly, and hence, the optimum integer solution may not be reached. One way of overcoming this computational problem is by introducing an artificial parameter ε (a very small number) which serves for truncation purposes. For example: whenever $[|b_j| \text{ or } |1-b_j|] \leq \varepsilon$ we set $b_j = 0$ and whenever $[|b_o| \text{ or } |1-b_o|] \leq \varepsilon$ we set $b_o = 0$. The magnitude of ε is extremely important and its optimal value may vary from one numerical problem to another. For example: suppose we derive $\varepsilon \geq |b_j| \geq 0$ and therefore set $b_j = 0$, the corresponding $t_j^* = \infty$ or $1/t_j^* = 0$. If the correct value for $|b_j|$ should have been > 0 and its corresponding t_j^* should have been finite, using the truncation

procedure we might have cut off a lattice point which might have proved to be the optimum integer solution to the problem. If on the other hand the arbitrarily selected value for ε is too small and the derived value for $b_j > \varepsilon$ where in fact it should have been = 0 then the derived cut is not optimal and the solution procedure may take more iterations than actually needed, but more iterations and more cuts increase the probability of severe rounding errors in the future. A slightly different aspect of the same problem is the case of b_o. When $b_o = 0$ we do not derive a cut from its corresponding row. If the arbitrarily selected ε is too small and the derived $b_o > \varepsilon$ where in fact it should be equal to 0, then we may derive a cut from a row which already has an integer solution. This cut will cut off a valid lattice point and may lead to a wrong solution to the problem. If, on the other hand, the arbitrarily selected ε is relatively large and the derived $\varepsilon \geq b_o > 0$ and therefore b_o is set equal to 0 where in fact it should be > 0, we miss a good opportunity to generate a deep cut from this source row and more steps may be needed to obtain the optimum solution to the problem.

We have now reached a stage where we can summarize our conclusions. Dangers associated with the fact that:

	ε is relatively small		ε is relatively large
1.	t_j^* is finite where in fact it should be infinite and more iterations are needed.	1.	A valid lattice point may be cut off by assuming $t_j^*=\infty$ where in fact t_j^* is finite.
2.	A valid lattice point may be cut off by using a source row at which $b_o>0$ where in fact $b_o=0$.	2.	More iterations may be needed because the best source row may be missed by assuming $b_o=0$ where in fact $b_o>0$.

A cheap solution procedure to the problem may be one which will avoid cutting off a valid lattice point at the expense of introducing more iterations than needed. This may be done by introducing two different values for two different truncation parameters ε_1 and ε_2 where $\varepsilon_1 < \varepsilon_2$. ε_1 should serve as the truncating parameter of $|b_j|$ and ε_2 should serve as the truncating parameter of b_0, as a result when ε_1 is relatively small, danger # 2 may be eliminated and when ε_2 is relatively large danger # 1 may be eliminated.

Introduction of more than one truncating parameter may eliminate part of the rounding problem but not all of it. A solution which is reached by one code may not be the true optimum solution of the problem. Different versions of the same algorithm with different selection rules of the source row may lead to different solutions. Different versions of the same algorithm which vary only with respect to the truncating parameter may lead to different solutions or to different number of iterations needed for solving the problem. It is, therefore, important to remember that the solution to the problem while using the cutting plane approach is not always reliable and more than one code should be employed to solve the same problem until the solution value is consistently the same.

The results obtained in this research indicate that the approach employed in this paper may be preferred to Gomory's algorithm for integer solutions to linear programs [3]. This approach is better in the sense that the total number of iterations and the total computer time needed for solving a given problem is, in most cases, significantly lower. Also, the total number of cuts required to solve a given problem is, in most cases, smaller; a fact which decreases the possibility of severe rounding and truncating errors. Different selection rules affect the overall method

significantly. This fact has been shown by Martin [7] and by the results of this research. Version 2 is the most time consuming per iteration but, nevertheless, it produced most of the best results among all codes compared earlier.

These experiments indicate that a good rule for selecting a cut among several alternatives may not be based on the expected outcome of the following pivot step. A good rule selects a cut which cuts off larger portions of the original convex set. Version 2 selects a cut which maximizes the number of t_j^*'s = ∞ and therefore, it may be considered as a relatively good rule which takes into account long run considerations rather than the immediate outcome of the following pivot step. The rounding error problem associated with cutting plane methods limits their practical value. Large scale ILP problems which are usually solved by the branch and bound [6] procedure tend to produce better results if cuts are incorporated into the branch and bound algorithm in the form of penalties [9]. The algorithm discussed in this paper may be used as a subroutine in a branch and bound program, the objective of which is to establish improved penalties on the active branches and thereby, reduce the total number of steps required for reaching the optimum integer solution.

References

1. Bowman, V.J. and Nemhauser, G.L., "A Finiteness Proof for the Modified Dantzig Cuts in Integer Programming," Naval Research Logistics Quarterly 17, pp. 309-313 (1970).

2. Glover, Fred, "Convexity Cuts," Working paper, School of Business, The University of Texas, Austin, December 1969.

3. Gomory, R.E., "An Algorithm for Integer Solutions to Linear Programs," Princeton IBM Mathematical Research Report, November 1958, also in R.L. Graves and P. Wolfe (eds.) Recent Advances in Mathematical Programming, McGraw-Hill, New York, 1963, pp. 269-302.

4. Gomory, R.E., "An Algorithm for the Mixed Integer Problem," RAND Report P-1885, February, 1960.

5. Haldi, John, "25 Integer Programming Test Problems," Working Paper No. 43. Graduate School of Business, Stanford University.

6. Land, A. M. and Doig, A. G., "An Automatic Method for Solving Discrete Programming Methods", Econometrica, 28, 1960, pp. 454-461.

7. Martin, G. T., "An Accelerated Euclidean Algorithm for Integer Linear Programming," Recent Advances in Mathematical Programming McGraw-Hill, New York, 1963, pp. 311-318.

8. Trauth, C. A. Jr., and Wollsey, R. E., "Integer Linear Programming: A Study in Computational Efficiency", Management Science, Volume 15, No. 9, May 1969, pp. 481-493.

9. Tomlin, J. A., "An Improved Branch and Bound Method for Integer Programming", Operations Research, 19, 1971, pp. 1070-1074.