#17

ALL-INTEGER INTEGER PROGRAMMING ALGORITHM

by

Ralph E. Gomory

International Business Machines Corporation
Research Center
Yorktown Heights, New York

ABSTRACT: An all-integer integer programming algorithm is described and a numerical example is given.

Research Report
RC-189
January 29, 1960

The purpose of this paper is to describe a new method of integer programming which differs from its predecessors in two main points:

(1) It is an all-integer method, that is, if the coefficients in the original matrix are integers all coefficients remain integer during the whole calculation.

(2) It is a uniform procedure closely resembling the ordinary dual simplex method with the difference that the pivot element is always a -1. The cycle of maximizing, adding an inequality, etc. characteristic of [2] has been eliminated.*

We will use the notation of [2] so that we regard the linear programming problem as the problem of maximizing in non-negative variables

$$z = a_{o,o} + \sum_{j=1}^{j=n} a_{o,j} \quad (-t_j) \tag{1}$$

subject to the restrictions

$$\begin{aligned} x_1 &= a_{1,o} + \sum_{j=1}^{j=n} a_{i,j} \quad (-t_j) \\ &\cdot \\ &\cdot \\ &\cdot \\ x_m &= a_{m,o} + \sum_{j=1}^{j=n} a_{m,j} \quad (-t_j) \end{aligned} \tag{2}$$

---

* I would like to acknowledge the stimulus of some ideas advanced by M. D. McIlroy of the Bell Telephone Laboratories.

$$
\begin{aligned}
t_1 &= -1 \quad (-t_1) \\
&\cdot \quad\quad \cdot \\
&\cdot \quad\quad\quad \cdot \\
&\cdot \quad\quad\quad\quad \cdot \\
t_n &= \quad\quad\quad -1 \quad (-t_n)
\end{aligned}
$$

or in matrix form to maximize z subject to

$$X = A^O T^O \qquad A^O = (\alpha_0, \alpha_1, \ldots, \alpha_n)$$

$$X = \begin{bmatrix} z \\ x_1 \\ x_2 \\ \cdot \\ \cdot \\ \cdot \\ t_1 \\ \cdot \\ \cdot \\ \cdot \\ t_n \end{bmatrix} \qquad T^O = \begin{bmatrix} 1 \\ -t_1 \\ -t_2 \\ \cdot \\ \cdot \\ \cdot \\ \\ \\ \\ \\ -t_n \end{bmatrix} \tag{3}$$

We will assume throughout that the columns α_j, $j \neq 0$ are lexicographically positive, i.e. that the problem is dual feasible. If they are not this can easily be arranged, see for example [3].

In the ordinary dual simplex method we would perform a series of Gaussian eliminations on the matrix A^O appearing in (3) in order to introduce new sets of non-basic variables. By doing this we continually form new expressions such as

$$X = AT \tag{4}$$

where the vector T is the vector of the new non-basic variables and A is the result of transforming A^0 by Gaussian eliminations. The pivot element in these steps is chosen by the dual simplex rule with the result that the columns of the matrices always remain lexicographically positive. When α_0, the column of constants, has only non-negative entries (except possibly in the first position), the solution to the ordinary linear programming problem has been obtained.

The procedure we will describe is very close to this. However, instead of constantly introducing new non-basic variables from among the original variables of the problem, new ones are created as we go along. These new variables will be added by introducing them first as basic variables in a new equation adjoined to the bottom of the matrix A appearing in (4) and then doing Gaussian elimination to make the new variable non-basic.

We will now turn to the derivation of these new relations. We give a derivation which includes the relations used in [1] and [2] as well as those used here.

Let us consider a typical equation

$$x = a_0 + \sum_{j=1}^{j=n} a_j \quad (-t_j) \tag{5}$$

or

$$0 = a_0 + \sum_{j=1}^{j=n} a_j \quad (-t_j) + 1 \quad (-x)$$

which is one of the equations appearing in (4). We will represent every coefficient a_j appearing in (5), as well as the 1, in the form $b_j \lambda + r_j$ where

b_j is an integer, r_j is a remainder, and λ is a positive number to be determined later. That is

$$a_j = b_j \lambda + r_j = \left[a_j/\lambda\right] \lambda + r_j \qquad j = 0, \ldots, n$$

$$1 = \left[1/\lambda\right] + r$$

(6)

$$0 \leq r_j < \lambda \,, \quad 0 \leq r < \lambda \,.$$

$$0 < \lambda$$

where square brackets indicate integer part of. Substituting the expressions in (6) into (5) and assembling all the remainder terms except r_o on the left gives

$$\sum_{j=1}^{j=n} r_j t_j + rx = r_o + \lambda \left\{ \left[a_o/\lambda\right] + \sum_{j=1}^{j=n} \left[a_j/\lambda\right] (-t_j) + \left[1/\lambda\right] (-x) \right\} \tag{7}$$

Any non-negative integer values for x and the t_j which satisfy (5) will also satisfy (7) and will make the left hand side of (7) a non-negative number since the r_j are non-negative. Let us now look at the right hand side and especially at the contents of the curly bracket which we can write separately as s,

$$s = \left[a_o/\lambda\right] + \sum_{j=1}^{j=n} \left[a_j/\lambda\right] (-t_j) + \left[1/\lambda\right] (-x). \tag{8}$$

Clearly the value obtained by substituting the same x and t_j into (8) will be an integer since all the coefficients appearing are integers (though of any sign). However, s is not only integer but in fact non-negative, for suppose s were

a negative integer such as -1, -2, etc. Since $r_o < \lambda$, a negative integer s would make the entire right hand side negative. However we know that the left side is non-negative, so this is a contradiction. Thus the s introduced by equation (8) is a new non-negative integer variable.

Let us examine this first for the case $\lambda = 1$. Here $\left[1/\lambda\right] = 1$ so on substituting (5) into (8) we have

$$s = \left[a_o\right] + \sum_{j=1}^{j=n} \left[a_j\right] (-t_j) - \left\{ a_o + \sum_{j=1}^{j=n} a_j \ (-t_j) \right\}$$

or

$$s = -f_o - \sum_{j=1}^{j=n} f_j \ (-t_j) \tag{9}$$

where f_j denotes the fractional part of a_j. This equation was the starting point for the algorithm described in [1] and [2]. Its relation to the new method is described at the end of this paper.

We will now consider the case $\lambda > 1$. Here we have $\left[1/\lambda\right] = 0$ and (8) becomes

$$s = \left[a_o/\lambda\right] + \sum_{j=1}^{j=n} \left[a_j/\lambda\right] (-t_j)$$

or

$$s = b_o + \sum_{j=1}^{j=n} b_j \ (-t_j). \tag{10}$$

(10) represents a new equation which must be satisfied (with non-negative s) by any integer solution to the original linear programming problem.

Thus we can adjoin (10) to the bottom of (4) and consider it as a possible row in which to pivot. Before proceeding any further we must assemble certain facts needed for using (10).

In the dual simplex method the only equations, or rows, in which one may pivot are those in which the constant term is negative (this means an unsatisfied inequality). Furthermore, in our notation, only negative elements are eligible pivot elements. Thus a row is eligible for pivoting only if it starts with a negative constant and contains other negative elements. Clearly we have $a_j < 0 \Rightarrow b_j = \left[a_j/\lambda\right] < 0$ so that if the row appearing in (5) is eligible so is the row in (10) which is derived from it.

Thus we are assured that if there are any eligible rows left in (4) we can create from any one of them a new eligible row (10). If there are no eligible rows the problem has been completed or has no solution.

We will now try to adjust the new row so that the pivot element will become a -1. That this is possible can be seen from the fact that for λ sufficiently large, all negative b_j's become -1's while all others vanish. So for sufficiently large λ the pivot element can only be a -1.

However, as we shall see, we can do better than this. We need one observation.

Let J be the set of indices j, $j \neq 0$, for which $a_j < 0$. Then if the dual simplex rule applied to (10) gives a pivot element $b_{j'} = -1$, we must have

$$\alpha_{j'} = \min_{j \in J} \alpha_j. \tag{11}$$

That is to say that if the pivot does turn out to be a -1 the pivot column can only be the (lexicographically) smallest column having a negative entry in the row (5). To see this we simply apply the usual pivot selection rule which says that the pivot column is that column $\alpha_{j'}$ for which $\left(\frac{-1}{b_{j'}}\right) \alpha'_j = \min_{j \in J} \left(-1/b_j\right) \alpha_j$. Hence

$$\frac{-1}{b_{j'}} \alpha_{j'} \leq -1/b_j \alpha_j \qquad \text{all } j \in J \tag{12}$$

if $b_{j'} = -1$, and b_j is a negative integer

$$\alpha_{j'} \leq -1/b_j \alpha_j \leq \alpha_j$$

so

$$\alpha_{j'} = \min_{j \in J} \alpha_j$$

Since the b_j do not enter into the choice of pivot column the same column will be chosen as pivot column for all those λ which have the property that they produce from (5) a row (10) with pivot element -1. Let us consider two such λ's, λ_1 and λ_2. The result of pivoting on the rows of type (10) will be to add $\left[b_o/\lambda_1\right] \alpha_{j'}$ or $\left[b_o/\lambda_2\right] \alpha_{j'}$ to the column of constants (the zero column). Clearly the zero column, and hence the objective function, will be decreased more if the smaller λ is used since the pivot column is the same in both cases and only the coefficient is changed.

We can summarize this by saying that the requirement for choosing λ is

(a) It should produce a pivot of - 1

and

(b) the λ used should be as small as possible.*

Using these two requirements we will now select λ. If $\alpha_{j'}$ is the smallest column α_j with $j \in J$ and α_j is another column of the same set then if $\alpha_{j'}$ is to have a -1 in the new row and be chosen by the simplex rule as pivot column we must have, for all $j \in J$, $\left(-1/b_j\right) \alpha_j \geq \alpha_{j'}$. Here b_j is a negative integer. Let μ_j be the largest integer for which $\left(1/\mu_j\right) \alpha_j \geq \alpha_{j'}$. (That there are some integers fulfilling this last inequality is clear since it is satisfied by 1).** In order for $\alpha_{j'}$ to be our pivot column, then, we must have

---

* This choice of λ is the one which, subject to condition (a), produces the biggest change in the zero-column. It does not necessarily result in replacing the original row (5) by a derived row (10) that is as strong an inequality as is possible or is necessarily either stronger or weaker than the original. For example from, $x = -4 - 3(-t_1) - 5(-t_2)$, we derive for $\lambda = 2$, $s = -2 - 2(-t_1) - 3(-t_2)$, for $\lambda = 3$, $s = -2 - 1(-t_1) - 2(-t_2)$, for $\lambda = 4$, $s = -1 - 1(-t_1) - 2(-t_2)$. Since the variables on the left are required to be non-negative these represent the inequalities $3t_1 + 5t_2 \geq 4$, $2t_1 + 3t_2 \geq 2$, $t_1 + 2t_1 \geq 2$, $t_1 + 2t_1 \geq 1$. The second of these is weaker than the first, the third stronger, the last weaker again.

** If the relation is satisfied for all integers, the α_j will never be chosen in preference to $\alpha_{j'}$ and the λ_j of (14) may be taken as zero.

$$-b_j = -\left[a_j/\lambda\right] \leq \mu_j \tag{13}$$

and the smallest λ fulfilling (13) is

$$\lambda_j = -a_j/\mu_j . \tag{14}$$

We note that λ_j is not necessarily an integer.

In order to fulfill conditions such as the above for all columns α_j, $j \in J$, we must have λ at least as great as

$$\lambda_{min} = \max_{j \in J} \lambda_j . \tag{15}$$

This choice of λ leads to the selection of $\alpha_{j'}$ as pivot and for $\alpha_{j'}$ we have $\mu_{j'} = 1$, hence $\lambda_{min} \geq -a_{j'}/\mu_{j'} \geq -a_{j'}$ so $\left[a_{j'}/\lambda_{min}\right] = b_{j'} = -1$.

We can summarize the procedure for obtaining the minimal λ in these four steps

a) select the smallest α_j with $j \in J$, this will be the pivot column α_{jo}.

b) for each α_j, $j \in J$, find the largest integer μ_j such that $\left(1/\mu_j\right) \alpha_j \geq \alpha_{jo}$.

c) set $\lambda_j = -a_j/\mu_j$.

d) find $\lambda_{min} = \max_{j \in J} \lambda_j$.

Step b) can usually be accomplished for each column by a single division.*

We are now in a position to describe the algorithm.

Assume an all-integer starting matrix A^o which is dual feasible. We choose a row having a negative constant term (if there are none the problem has been solved), for this row we choose λ_{min} and the pivot column by the four steps given above. We create a new row of type (10) using this λ_{min} and adjoin this to the bottom of the matrix A. We now perform Gaussian elimination on the new row, i.e., we introduce s as a new non-basic variable, we drop the new row and then repeat this process. Because this pivot element is -1, the matrix remains in integers. A numerical example is attached as an appendix.

We will next show that for certain rules of choice of row this process is a finite one.

We will assume that the problem has some integer solution X' which gives the objective function a value z_o.

---

* If α_j and α_{jo} both begin with non-zero terms $a_{o,j}$ and $a_{o,jo}$, then if $a_{o,jo}$ does not divide $a_{o,j}$, $\mu_j = \left[a_{o,j}/a_{o,jo}\right]$. If $a_{o,jo}$ does divide $a_{o,j}$, then $\mu_j = a_{o,j}/a_{o,jo}$ if $\alpha_j \geq \left(a_{o,j}/a_{o,jo}\right) \alpha_{jo}$, and $\mu_j = \left(a_{o,j}/a_{o,jo}\right)$ -1 otherwise. If the two columns begin with unequal numbers of zeros (α_{jo} must have more) μ_j is arbitrarily large and $\lambda_j = 0$. If both columns have p zeros, the procedure is as above with $a_{p,j}$ and $a_{p,jo}$ substituted for $a_{o,j}$ and $a_{o,jo}$. Also it is worth noting that μ_{jo} is always 1.

At any stage of the calculation we have the variables x represented as a constant column plus a sum of non-negative columns times non-negative variables (preceded by minus signs)

$$X = \alpha_o + \sum_{j=1}^{j=n} \alpha_j (-t_j) \tag{16}$$

Since any solution must give the t_j non-negative values we see that any solution X is lexicographically equal or less than α_o. Furthermore α_o is decreased lexicographically after each step since a negative multiple of one of the columns is added to it. Thus we have a descending sequence of α_o' s

$$\alpha_o > \alpha_o^1 > \alpha_o^2 \ldots > X' \tag{17}$$

bounded below by the assumed solution X'. Let us suppose that we had an infinite sequence of this sort. Since we are dealing with all-integer vectors α_o, α_o^1, α_o^2, ... the components of the vectors change by integer amounts. Let us consider the first component (the objective function). If this decreased indefinitely it would eventually get below z_o the first component of X'. This is a contradiction. Consequently the first component can only decrease strictly for a finite number of steps and then must remain stationary thereafter at some fixed value $z' \geq z_o$. From this point on the second component must be non-increasing. There are now two possibilities:

(a) The second component may reach some final value and remain fixed thereafter, or

(b) it might decrease indefinitely.

If (a) occurs we can move on to the third component which presents the same two alternatives. If alternative (a) occurs for each component we will have a fixed α_0 after a finite number of steps. Since α_0 will decrease strictly as long as there are negative elements in the constant column this means that there are no more such elements and that the problem has been solved. A rule that will guarantee finiteness, then, is one that can exclude possibility (b).

Let us consider the first component for which alternative (b) occurs. After a certain point the component becomes negative and remains negative. Thus its row is eligible for selection as a row (5). However it is never selected, for if it were selected, and a row of type (10) generated from it, the pivot column $\alpha_{j'}$ would have a strictly negative coefficient $a_{j'}$ in the row in question and, upon pivoting, the constant term of the row would be strictly increased. This would contradict the assumed non-increasing character. Thus any rule which, if the constant term of a row goes negative and remains negative, will sooner or later select that row, will exclude possibility (b), and give a finite algorithm.

Examples of such rules are

(1) Always select the first row (from the top) having a negative element.

(2) Select the rows by a cyclic process, i.e., on the first step look at the first row and then if it does not have a negative constant look at its successors, on the second step look at the second row and then its successors, etc.

(3) If the rows are chosen at random a finite process will result with probability 1.

As an example of a rule not covered by this finiteness proof we cite the ordinary simplex selection rule, that is choose the row with the largest negative constant term.

The rules of choice we are currently using in computations have evolved partly through a trial and error process. Our first attempt was to use the old tried and true simplex rule of choice described above even though we lacked a finiteness proof for it. We chose at each step the row with largest negative constant and formed the new row from it. The idea, of course, was that a large negative constant in the original row would, most likely, lead to a reasonably large constant in the new row. This approach was an almost complete failure. This rule did not seem to solve any but the very smallest problems. As an extreme example we can cite one 7 variable 7 inequality problem which we later solved in 10 steps but which with this rule ran for 1200 pivots before being taken off the machine.

An approach which does seem to be effective, however, is to focus attention not on the rows but rather on the columns and to construct a new row to make the pivot column as large as possible. There seem to be two reasons why a column criterion makes more sense here than a rule such as seeking for a large negative constant. One reason is that in this all-integer method an integer multiple of the pivot column is subtracted from the zero column. Thus the amount of progress is at least as great as the size of the pivot column. This is in contrast with the ordinary simplex method where

it is possible for a large new column to be brought in at a very small level. Secondly, when dealing with degeneracies, a common feature of integer programming problems, it is the degree of degeneracy of the columns that makes the essential difference. That is, considering any two columns such as

$$\alpha_1 = \begin{bmatrix} 0 \\ 0 \\ 7 \\ -3 \\ 2 \\ 1 \end{bmatrix} \qquad \alpha_2 = \begin{bmatrix} 0 \\ 2 \\ -1 \\ -4 \\ 2 \\ 4 \end{bmatrix}$$

any multiple of α_2 is to be preferred to any multiple of α_1 so that here everything is determined by the column and the size of the constant term is secondary.

Consequently the following approaches seem plausible for this form of integer programming.

First rank the columns of the matrix as 1, 2, 3, 4, etc. in order of descending size, denote the rank of j^{th} column by C (j). Then assign to each eligible row the rank $R(i) = \max_{j \in J} C(j)$. This rank, R(i), is the rank of the column that would be chosen as pivot column if the i^{th} row were used to generate a new row. We then choose a row i_o by $R(i_o) = \min_{\text{eligible } i} R(i)$.

A still stronger approach is the following which we are currently programming. First obtain R(i) values not only for the eligible rows but for all rows. Then choose a row i_o either as above or by some other rule such as the largest negative criterion and form the appropriate new row (10). Next

carry out the pivot operation only on the column of constants, i.e., more exactly compute (in a separate place) the numbers $a_{i,o} + b_o a_{i,j_o}$. Consider the rows i for which this expression is negative. These are the rows whose constants would be negative after the pivot step. If among these rows there are some with rank strictly lower than $R(i_o)$ select one, say i_1. If a_{i_1,j_o} times the row of the b_j's is added to the i_1 row a new row is formed with constant term $a_{i_1,o} + b_o a_{i_1,j_o}$ which is negative. This new row has a rank strictly smaller than $R(i_o)$ since its entry in the j_o column is zero and its entries in columns having rank $> R(i_o)$ are non-negative. This last follows from the definition of rank which shows that both rows involved have this property and from the fact that a_{i_1,j_o} is ≥ 0 because $R(i_1) < R(i_o)$. This process can now be iterated, with the newly created combined row playing the role of the original row i_o, and being used to create new b_j's, etc.* The basic idea here is to use the fact that in attempting to satisfy the inequality represented by the row i_o we violate the inequality represented by the row i_1 to revise the new row and improve the rank of the pivot column.

Although one's first reaction to this sort of scheme is that the computation involved in these selection methods is excessive a more detailed examination shows that the amount of work required can be expected to be much less than that required for a full pivot step. Thus if the number of pivot steps is reduced substantially these methods will be worthwhile.

---

* In this process only the last row generated is actually used in pivoting. Thus it is unnecessary to choose an optimal λ except when dealing with this last row. For all others $\lambda = -b_{j_o}$ will suffice to provide a - 1 in the smallest column.

The method we have used in our code so far has been only a very crude approximation of the above. The rule divides into two parts (a) if all the relative cost coefficients, i.e., the $a_{0,j}$, $j \neq 0$ are non-zero, that is to say that we are in a completely non-degenerate situation, we have used the old simplex rule choosing the row with the largest negative constant. If (b) some of the $a_{0,j}$ are zero we form the function N(j) which is a count of the number of zeros at the head of each column before a positive number is encountered. We then use N(j) which is an approximation to the ranking of the columns as the C(j) of the first method described above.

Using this code we have had irregular but interesting results. For example, we have a series of four 15 variable 15 inequality problems arising from coding theory. The inequalities in the four problems are identical except that the constant terms in the inequalities are increased by adding 2 to get from one problem to the next. All coefficients of the variables remain the same. Three of the problems are solved in 17, 21, 23, pivot steps respectively. Of the fourth we know only that it requires, with this code, over 400. A similar series of three 32 x 32 problems involved 23 and 156 iterations (this last took 3 minutes on the 704) with a third problem unsolved after 200. Although in other sparser problems the results have been less irregular there are still plenty of failures. There are indications that the procedures described above are worth investigating.

We will now consider the relation of this all-integer method to the method of [1] and [2].

The derivation of the inequalities given earlier in this paper shows that both the algorithm described here and the algorithm of [1] and [2] are extreme cases. In [1] and [2] only inequalities of the $\lambda = 1$ type were used, while here only inequalities of the $\lambda > 1$ type were involved.* Inequalities of the $\lambda = 1$ type will give non-trivial eligible rows whenever there are non-integers present in the zero column whether there are any negative elements there or not. However, these inequalities are not available if we have an all-integer matrix. On the other hand the inequalities of $\lambda > 1$ type are available whenever there are negative elements in the zero column but fail if the zero column is all non-negative but possibly containing non-integers.

The various states of the zero-column with the corresponding available types of inequality are summarized in this table.

	Negative Constants	No Negative Constants
Some Non-Integers	$\lambda = 1,\ \lambda > 1$	$\lambda = 1$
All Integers	$\lambda > 1$	Solution

It is clearly possible to combine both classes of inequalities into algorithms having many interesting properties. For example, one could start by doing the regular simplex method to get somewhere near a solution or possibly to obtain a non-integer solution. Then from some point on one could pivot only on additional rows generated either for $\lambda = 1$ or $\lambda > 1$. One type or the

---

* Note that the derivation of the inequalities does not depend on the a_j being integers.

other is always available. Since the pivot element is either - 1 or a proper fraction - f_j the D -- number described in [2] will now be monotone decreasing. Another approach would be to set in advance a bound for D and to switch to additional rows whenever a pivot element of the ordinary simplex method would violate the bound on D. Operating with a fixed D means that round off problems can be eliminated. There are obviously many combined algorithms that are possible and we have no idea which are better than others.

APPENDIX

Example

Integer Programming

$$\min \quad z' = 10x_1 + 14x_2 + 21x_3$$

or

$$\max \quad z = -(10x_1 + 14x_2 + 21x_3)$$

Subject to

$$\begin{aligned} 8x_1 + 11x_2 + 9x_3 &\geq 12 \\ 2x_1 + 2x_2 + 7x_3 &\geq 14 \\ 9x_1 + 6x_2 + 3x_3 &\geq 10 \end{aligned}$$

The row marked with an arrow is used at each step

$j_o = 1$

$\mu_1 = [10/10] = 1 \quad \mu_2 = [14/10] = 1 \quad \mu_3 = [21/10] = 2$

$\lambda_1 = 2/\mu_1 = 2 \quad \lambda_2 = 2/\mu_2 = 2 \quad \lambda_3 = 7/\mu_3 = 7/2$

$$\lambda_{min} = \max (2, 2, 7/2) = 7/2$$

(1)

	1	$-x_1$	$-x_2$	$-x_3$
$z =$	0	10	14	21
$s_1 =$	-12	- 8	-11	- 9
$s_2 =$	-14	- 2	- 2	- 7 ←
$s_3 =$	-10	- 9	- 6	- 3
$x_1 =$	0	- 1	0	0
$x_2 =$	0	0	- 1	0
$x_3 =$	0	0	0	- 1
$\bar{s}_1 =$	- 4	- 1*	- 1	- 2
$\lambda =$	7/2			

(2)

		1	$-\bar{s}_1$	$-x_2$	$-x_3$
z	=	-40	10	4	1
s_1	=	20	- 8	- 3	7
s_2	=	- 6	- 2	0	- 3 ←
s_3	=	26	- 9	3	15
x_1	=	4	- 1	1	2
x_2	=	0	0	- 1	0
x_3	=	0	0	0	- 1
$\bar{s}_2$	=	- 2	- 1	0	- 1*
λ	=	3			

(3)

		1	$-\bar{s}_1$	$-x_2$	$-\bar{s}_2$
z	=	-42	9	4	1
s_1	=	6	-15	- 3	7
s_2	=	0	1	0	- 3
s_3	=	- 4	-24	3	15 ←
x_1	=	0	- 3	1	2
x_2	=	0	0	- 1	0
x_3	=	2	1	0	- 1
$\bar{s}_3$	=	-1	- 1*	0	0
λ	=	24			

(4)

		1	$-\bar{s}_3$	$-x_2$	$-\bar{s}_2$
z	=	-51	9	4	1
s_1	=	21	-15	- 3	7
s_2	=	- 1	1	0	- 3 ←
s_3	=	20	-24	3	15
x_1	=	3	- 3	1	2
x_2	=	0	0	- 1	0
x_3	=	1	1	0	- 1
$\bar{s}_4$	=	- 1	0	0	- 1*
λ	=	3			

(5)

		1	$-\bar{s}_3$	$-x_2$	$-\bar{s}_4$
z	=	- 52	9	4	1
s_1	=	14	-15	- 3	7
s_2	=	2	1	0	- 3
s_3	=	5	-24	3	15
x_1	=	1	- 3	1	2
x_2	=	0	0	- 1	0
x_3	=	2	1	0	- 1

Solution $z' = 52$

$x_1 = 1, \quad x_2 = 0, \quad x_3 = 2$

SAME EXAMPLE REDONE USING ROW COMBINATION

(1) The first step is the same since no higher ranking inequalities are made negative by the pivot step.

(2)

Column		Rank 1	2	3	
		1	$-\bar{s}_1$	$-x_2$	$-x_3$	Row
z	=	-40	10	4	1	Rank
s_1	=	20	- 8	- 3	7	2
s_2	=	- 6	- 2	0	- 3	3 ←
s_3	=	26	- 9	3	15	1
x_1	=	4	- 1	1	2	1
x_2	=	0	0	- 1	0	2
x_3	=	0	0	0	- 1	3
s_2	=	- 1	- 1*	0	0	

Selecting the only negative row we obtain as before the row (-2, -1, 0, -1). However its use would drive the lower ranking s_3 row negative. Combining 15 (-2, -1, 0, -1) + (26, -9, 3, 15) = (-4, -24, 3, 0) a row of rank 1. Applying $\lambda = 24$ gives (-1, -1, 0, 0). No iteration of the process is possible so this is used.

(3)

		1	$-\bar{s}_2$	$-x_2$	$-x_3$
z	=	-50	10	4	1
s_1	=	28	- 8	- 3	7
s_2	=	- 4	- 2	0	- 3 ←
s_3	=	35	- 9	3	15
x_1	=	5	- 1	1	2
x_2	=	0	0	- 1	0
x_3	=	0	0	0	- 1
$\bar{s}_3$	=	- 2	- 1	0	- 1*
λ	=	3			

(4)

		1	$-\bar{s}_3$	$-x_2$	$-\bar{s}_4$
z	=	-52	9	4	1
s_1	=	14	-15	- 3	7
s_2	=	2	1	0	- 3
s_3	=	5	-24	3	15
x_1	=	1	- 3	1	2
x_2	=	0	0	- 1	0
x_3	=	2	1	0	- 1

REFERENCES

1 RALPH E. GOMORY, "Outline of an Algorithm for Integer Solutions to Linear Programs," Bulletin of the American Mathematical Society, vol. 64, no. 5,(1958).

2 RALPH E. GOMORY, "An Algorithm for Integer Solutions to Linear Programs," Princeton-IBM Mathematics Research Project Technical Report No. 1, November 17, 1958.

3 G. B. DANTZIG, L. R. FORD, JR., and D. R. FULKERSON, "A Primal-Dual Algorithm for Linear Programs," in Annals Study 38 "Linear Inequalities and Related Systems," Kuhn and Tucker, Eds., Princeton University Press, 1956.