LARGE AND NONCONVEX PROBLEMS IN LINEAR PROGRAMMING[1]

BY

R. E. GOMORY

Reprinted from

Proceedings of Symposia in Applied Mathematics, Volume XV

EXPERIMENTAL ARITHMETIC, HIGH SPEED COMPUTING AND MATHEMATICS

LARGE AND NONCONVEX PROBLEMS IN LINEAR PROGRAMMING[1]

BY

R. E. GOMORY

Introduction. Recent work, based on the ideas of Dantzig and Wolfe [1], Ford and Fulkerson [2], and Wolfe [3], has greatly extended the range of problems that can be approached by linear programming. Systems of linear inequalities so large that the coefficient matrices cannot even be effectively written down can, in some cases, be dealt with by implicit methods and optimal solutions can be attained. Examples of these are Dantzig [4], Gilmore and Gomory [5], Gomory and Hu [6], and Dzielinsky and Gomory [7]. Recently, a connection with mixed integer programming problems has been established by Benders [8]. It is the purpose of this paper to review these developments in a unified way showing that they are all special cases of an extremely simple extension to the basic calculations that must be gone through in performing G. B. Dantzig's simplex method.

If we take as our basic problem the problem of maximizing (or minimizing)

$$z = cx,$$

subject to the restrictions on the vector x

$$\begin{aligned} Ax &\leqq b, \\ x &\geqq 0, \end{aligned} \tag{1}$$

we can transform to equations in non-negative variables in two ways. Introducing a non-negative vector for the differences (slacks) of the two sides in (1), we can write either

$$\begin{pmatrix} 1 & 0 & -c & 0 \\ 0 & I & A & b \end{pmatrix} \begin{pmatrix} z \\ s \\ x \\ -1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \tag{2}$$

or alternatively

$$\begin{pmatrix} z \\ s \\ x \end{pmatrix} = \begin{pmatrix} 0 & c \\ b & -A \\ 0 & I \end{pmatrix} \begin{pmatrix} 1 \\ x \end{pmatrix}. \tag{3}$$

In both cases, all variables except z are required to be non-negative.

[1] This research was supported in part by the Office of Naval Research under Contract No. Nonr 3775(00), NR 047040.

Reprinted with permission of the publisher
The American Mathematical Society from the
Proceedings of the Symposium on the Interactions
between Mathematical Research and High-Speed Computing
Vol. XV, pp. 125-140, Copyright © 1963

Whichever form is used, call the matrix of coefficients $\bar{A}$. Then, in using any form of the simplex method, we perform repeatedly the following basic steps.

(A) Select a column of the current coefficient matrix.

(B) Select a row of the current coefficient matrix.

(C) Do a pivot step (Gaussian elimination) using as pivot element the element at the intersection of the chosen row and column. Go back to Steps (A) and (B).

As is well known, the Gaussian elimination Step (C) need not be performed on the entire matrix $\bar{A}$ or its transform. If we consider (2), the effect on $\bar{A}$ of the Gaussian eliminations (column eliminations) is equivalent to multiplying on the left by a nonsingular square matrix P. It is, therefore, enough at each repetition of the Steps (A), (B) and (C) to perform (C) on the current P. The effect of the eliminations on any particular element $\bar{a}_{ij}$ of $\bar{A}$ can then be found by multiplying by the updated P. Using this procedure (the revised simplex method) on an $m \times n$ matrix enables one, at least in Step (C), to perform the arithmetical operations only on an $m \times m$ matrix. Generally, there is no arithmetical saving in this as the information required in Steps (A) and (B) requires doing the arithmetic that can be saved in (C).

Similarly, if (3) is used, the pivot steps are row eliminations which are equivalent to multiplication on the right by a nonsingular P. In this case, it is an $n \times n$ P that needs to be processed. Again, because of (A) and (B), there is no arithmetical saving, so let us look at (A) and (B).

In the primal simplex method, the order of operation is (A), (B), (C). The column selection operation (A) is very simple. One selects the column for which the first entry in the transformed $\bar{A}$ is minimal. Note that it is this row which contains the objective function. To find this first entry directly if P is on the left, one forms the scalar product of the first row P' of P with the column of $\bar{A}$, thus doing an amount of arithmetic roughly equivalent to the saved Gaussian elimination.

The row selection is very nearly the same, but a ratio is involved. If we call the elements of the transformed $\bar{A}$ the $q_{i,j}$, give the chosen column index j_0 and the column corresponding to the original constants b, the index 0, then the row selection is to find the i for which q_{i,j_0} is positive and for which $q_{i,0}/q_{i,j_0}$ is minimal.

There is one further difference between the minimizations involved in (A) and (B) in the primal simplex method. The minimization in (A), which did not involve a ratio, was not necessary, only desirable. It would have been sufficient to take any column in $P\bar{A}$ whose first entry was negative. The minimization in (B) is necessary to preserve feasibility. This distinction is especially important when we come to integer programming.

When doing the dual simplex method (which may also involve a P either on the left or right as desired), it is Step (A) which involves the ratio and in which minimization is necessary; Step (B) involves no ratio and minimization is desirable but not necessary. Row selection, Step (B), requires finding the minimal $q_{i,0}$, and column selection requires finding among $q_{i_0,j} > 0$, the one for which $q_{0,j}/q_{i_0,j}$ is minimal. The i_0 used is obtained from (B), and the order of operations is (B), (A), (C).

Note that in both (A) and (B) we are performing a minimization or maximization operation of the simplest sort. We simply look at all the numbers involved one after another and find the largest or smallest one. What is done in essence when dealing with large problems is to replace this simple maximization by a maximization algorithm. This may be a recursive computation, a network flow calculation, or even another linear programming problem. The objective function may be linear or involve a ratio of linear terms depending on whether the step is of the simple or ratio type.

If the problem has a moderate number m of rows but an enormous number n of columns, (A) is replaced by the maximization algorithm, and left multiplication is used. If the columns are moderate in number but the row count is too great, then (B) is replaced by an algorithm and right multiplication is used. Whether a primal or dual method is used determines whether the objective function in the algorithm is to be linear or rational. The point is that by doing the maximization operation other than by simply looking at all the numbers, one avoids the necessity for ever calculating most of the numbers involved.

Of course, this is only possible when the rows or columns are given in some systematic way, as is the case in the following examples.

Cutting stock example. In this example, we will use the primal simplex method, left multiplication, and a minimization algorithm with a linear objective form. The problem (see Eisemann [9], and Gilmore and Gomory [5]) is as follows.

We assume that an unlimited stock of standard lengths of one material is available from which one is to cut lengths to fill a list of orders. An order consists of a request for a number N_i of pieces of length l_i, $i = 1, \cdots, n$, $l_i \leqq L$, the standard length. The problem is to fill the orders by cutting up the stocked lengths so that the waste, i.e., unused but cut up pieces, is minimized.

If we assume that it is possible to list every single way to cut up the standard length L into some collection of desired lengths l_i, then the problem can be written down as a linear programming problem. Specifically,

$$\begin{aligned} &\text{minimize } \sum_j x_j, \\ &\sum_j a_{ij} x_j \geqq N_i, \qquad i = 1, \cdots, n, \\ &x_j \geqq 0, \quad \text{all } j. \end{aligned} \tag{4}$$

The interpretation of (4) is as follows. x_j is the number of standard rolls to be cut up in the jth way, a_{ij} is the number of pieces of length l_i obtained each time a standard length is cut up in the jth way, and the inequalities indicate that the total number of each length l_i produced is at least as great as the demand. Minimizing the sum of the x's, i.e., minimizing the number of standard rolls to be used, is equivalent to minimizing waste.

This formulation has two drawbacks. One is that the x's of the solution, in order to be interpretable as numbers of standard rolls, must be integers. Although this is a real difficulty in problems involving small N_i and hence small x's, it is not

the difficulty that is the subject of this paper. In problems with large N_i, rounding to the nearest integer seems to be an adequate procedure. The other difficulty, and this is the subject here, is the enormous number of columns in the resulting matrix $\bar{A}$. There is one such column for every way to cut up a standard length, and even for a moderate number n of different l_i, say twenty or thirty, the number of different ways of cutting can be so large as to even defy writing down. (The exact number depends on the relative sizes of the l_i and L.)

Nevertheless, one can go through Steps (A), (B) and (C) and solve the problem without ever writing down these cutting patterns. Let us assume that we have done the preliminary problem of getting a feasible basis; i.e., the matrix is in a form where it has a unit submatrix of order $(n + 1)$ and a non-negative b. This can be done without any appreciable arithmetic effort and is described in detail in [5]. We emerge from this preliminary stage with a transforming matrix P, which would, if the multiplication were carried out, transform the $\bar{A}$ corresponding to (4) into feasible form. After this, we are ready to carry out Step (A). We are to select the column A_j of $\bar{A}$ for which $p_1 A_j$ is maximal. p_1 is the top row of P, and we are maximizing rather than minimizing because the problem itself is a minimization. Now the columns of $\bar{A}$ contain a -1 as the first entry, but after that the entries $a_{i,j}$ are restricted only by the condition

$$\sum_i a_{i,j} l_i \leqq L, \tag{5}$$

which says that the lengths produced by cutting up one standard length should not exceed the standard length. Any set of n non-negative integers $a_{i,j}$ satisfying (5) will be a cutting pattern and will appear somewhere in $\bar{A}$. Consequently, the column selection problem is the following maximization problem: find non-negative integers $x_1, \cdots, x_n$ such that

$$\sum x_i l_i \leqq L, \qquad i = 1, 2, \cdots, n, \tag{6}$$

and such that $p_1 \cdot (-1, x_1, \cdots, x_n)$ is maximal. Writing $p_1 = (\Pi_0, \Pi_1, \cdots, \Pi_n)$, we can disregard the effect of Π_0 so this is equivalent to maximizing

$$v = \sum \Pi_i x_i, \qquad i = 1, 2, \cdots, n. \tag{7}$$

Now to maximize (7) subject to (6) is the problem known as the knapsack problem [10], the name coming from the following interpretation. The restriction (6) is taken as being the weight restriction on the contents of a knapsack. Items of weight l_i can be put into the knapsack until the weight limit is reached. Each ith object is worth an amount Π_i and the problem is to get the greatest total value into the knapsack without exceeding the weight limitation.

To find the largest possible value of v, one computes recursively the function $v_i(y)$ (dynamic programming). The interpretation of $v_i(y)$, in knapsack terms, is that it is the greatest value that can be put in a knapsack of capacity y if one is allowed to put in nonzero amounts of the first i items only. Clearly,

$$v_1(y) = \Pi_1[y/l_1],$$

and $v_n(L)$ is the largest possible value of c, the one we are looking for. The recursion for $v_i(y)$ is

$$v_i(y) = \max \{\Pi_i x_i + v_{i-1}(y - l_i x_i)\},$$

for

$$0 \leqq x_i \leqq [y/l_i].$$

If the functions $v_i(y)$ are kept after being computed, it is an easy matter to work back, after finding $v_n(L)$, and get the values of x_i, $i = 1, \cdots, n$, that gave that value. These values of x then form the column which, together with a -1 in the first position, has been selected in Step (A).

Now that the column has been selected, we multiply it by P to obtain the corresponding column of $P\bar{A}$, and we need only this transformed column and the transform of the column containing b to carry out (B) in the usual way. We then do (C) transforming P only. With the new P we again go through the Steps (A), (B), (C) with the selection process of (A) being done by a new knapsack problem. This is iterated until finally the scalar product of the current p_1 with the chosen column is negative (positive). At this point, the usual simplex process has terminated and we have the solution. This calculation, though with an improved method for solving the knapsack problem, has been programmed for the IBM 7090 and problems as large as $n = 50$, $L = 200$, $10 \leqq l_i \leqq 80$, solved in a few minutes running time.

A network design problem. In this next example, we will deal with a matrix representing a great many inequalities, but only a comparatively small number of variables, i.e., many rows but only a few columns.

Multiplication will be from the right and the method described here will be primal. The algorithm used will involve a ratio minimization. A more detailed description is available in Gomory and Hu [6] where a dual method with its simpler linear minimization problem is also given. The problem is as follows.

A network of n nodes is to be designed to accommodate certain flows. The network consists of nodes N_i, $i = 1, \cdots, n$, and arcs A_{ij} connecting the nodes. With each arc is associated a non-negative number y_{ij} called the capacity. In such a network a pq-flow is a set of x_{ij} such that

$$0 \leqq x_{ij} \leqq y_{ij}, \qquad \text{all } i,j,$$

$$\sum_i x_{ij} - \sum_k x_{jk} = 0, \qquad j \neq p,q. \tag{8}$$

Intuitively, one thinks of a liquid flow, with the x_{ij} the amounts flowing through pipes having limited capacities. The equations (8) then state that liquid is conserved at the nodes with the exception of the two special ones p, the source, and q, the sink. The amount of flow is by definition,

$$f_{pq} = \sum_j x_{pj} = \sum_k x_{kq},$$

the inflow at the source or outflow at the sink. For a given network, the largest possible amount of pq flow is denoted by f_{pq}.

The design problem involves constructing a network whose maximal flows f_{pq} all exceed a set of requirements r_{pq} for all node pairs p and q. This problem arises when one considers a completely time shared communication net. The flows then are communication flows and the net is used at different times to relay messages between different node pairs. Of course, this can always be done simply by constructing a network with enormous capacities $y_{i,j}$. Our problem then is to construct such a network with least cost. Suppose that each arc has a unit cost c_{ij}. Then the total cost of the network, the quantity to be minimized, is

$$C = \sum_{i,j} c_{ij} y_{ij}, \qquad \text{all } i,j.$$

To convert this into a linear programming problem is quite easy. One simply appeals to the max flow min cut theorem of Ford and Fulkerson which states that

$$f_{pq} = \min_A \left(\sum_{i,j} y_{ij} \right), \qquad i \in A, j \notin A,$$

where the A ranges over all node subsets that contain p but not q. In words, this says that the max flow equals the total (directed) flow carrying ability of the smallest cut where a cut is a collection of arcs whose removal splits the network into two components, one including a set of nodes A including the source, and the other containing the remaining nodes, among them the sink.

Using this fact, we can write down the condition that our network provide the flows $f_{pq} \geqq r_{pq}$. The conditions are for all pairs p,q

$$\sum_{i,j} y_{ij} \geqq r_{pq}, \qquad i \in A, j \notin A, \text{ all } A \ni p \in A, q \notin A.$$

The only difficulty is that there are $n(n-1)2^{n-2}$ such inequalities for an n node network. Once again, we have a large linear programming problem, but this time m, the number of rows, is enormous compared to the number of columns.

The situation here is a little more complicated than in the preceding example, and the description must be something of an outline. For a full explanation, see [6]. Essentially, Step (A) is done in the usual explicit manner. The problem is to do (B) implicitly and without being forced to write out all the equations. We will give a heuristic exposition here. Step (A) has found a column which, when added to the present solution, improves it. The problem being solved by Step (B) is to find out what multiple or weight of this new column can be added. The new column can be added to the old solution until the solution is so changed that some inequality is about to be violated. The ratio $q_{i,0}/q_{i,j_0}$ involved in Step (B) is what determines the amount of the column that can be added before the particular inequality represented by row i is violated. To do this implicitly in our case, we take the current values of the y_{ij} and add to them an indeterminate amount θ. These y_{ij} values then give us a network whose capacities depend on a single parameter θ. Using a variant of the labeling process of Ford and Fulkerson, we can obtain $f_{pq}(\theta)$ for this network, i.e., we can get the piecewise linear function that gives the max flow from p to q in this network as a function of θ.

When we have this, we can find the largest value θ_{pq} of θ for which $f_{pq} \geqq r_{pq}$. By the max flow min cut theorem, there must be for this value of θ a cut whose capacity is r_{pq} and decreases with increasing θ. This then is the first inequality to be violated of all those obtained from this particular pair pq of nodes. It also turns out that we obtain the inequality explicitly as a by-product of the labeling process. The process must be repeated for each pair pq, and the smallest of all the θ_{pq} is the sought for minimal ratio; the cut associated with it is the sought for row. Equipped with this row, one is then ready for Step (C).

As usual, Step (C) is carried out only on a square matrix. This time the matrix is square with a size determined by the number of columns.

We should add that in this network design problem, the labor can be cut down still more by further theoretical considerations. If all c_{ij} are 1, then the problem is solvable by a simple direct construction. If the problem is symmetrical, i.e., all arcs have the same capacity in both directions ($y_{ij} = y_{ji}$), as is often the case, it is possible to show, see (11), that from the $n(n-1)/2$ essentially different requirements that remain after symmetry is taken into account, there is a subset of only $n-1$ that dominate in the sense that if these $n-1$ requirements are met, all the others will automatically be met too. This subset of dominating requirements can be obtained easily. This means that in the course of Stage (B) of the calculation, the parametric network flow calculation need be made for only $n-1$ rather than $n(n-1)$ pairs of nodes.

Dantzig–Wolfe decomposition. In this example, we revert to left multiplication, a primal method, and a simple maximization. This time, the algorithm used is itself a simplex computation. The reference here is Dantzig–Wolfe [1].

Consider a linear programming problem

$$\text{minimize } cx,$$

$$\text{subject to } Ax = B,$$

in which we divide the equations into two groups, an upper one and a lower one so that we replace $Ax = b$ by

$$A_1x = b_1, \tag{9}$$

$$A_2x = b_2, \tag{10}$$

where A_1 has m_1 rows and A_2 has m_2. We now transform the problem into one with $m_1 + 1$ rows and a great many columns by the following reasoning. The only x's we need to consider as solutions to (9) are those that also solve (10). Let us suppose that the convex of solutions to (10) is bounded. Then we can, in principle, give a complete list of vertex solutions $x_1, x_2, \cdots, x_N$ to (10), and

any solution to (10) whatsoever is a weighted combination of these x_i with non-negative weights w_i totalling 1. Therefore, it is only necessary to consider x's of this form. So, upon substituting, the problem becomes:

$$\text{minimize } \sum_i (cx_i)w_i,$$

$$\text{subject to } \sum_i (Ax_i)w_i = b_1,$$

$$\text{and } \sum_i w_i = 1,$$

or, using $\bar{c}_i$ for cx_i and α_i for Ax_i, we have the linear programming problem:

$$\text{minimize } \sum_i \bar{c}_i w_i,$$

$$\sum_i \alpha_i w_i = b_1,$$

$$\sum_i w_i = 1,$$

which is a problem with fewer rows but an enormous number of columns.

We now use an algorithm for Step (A). In Step (A), we maximize the scalar product of the top row

$$p_1 = (\Pi_0, \cdots, \Pi_{m_1+1}) = (\Pi_0, \overline{\Pi}, \Pi_{m_1+1})$$

of the transforming matrix with the column vector $(\bar{c}_i, \alpha_i, 1)$, the maximization extending over all i. But

$$\max_i p_1 \cdot (\bar{c}_i, \alpha_i, 1) = \max_i p_1(cx_i, A_1x_i, 1) = \max_i (\Pi_0 c + \overline{\Pi} A_1) \cdot x_i + \Pi_{m_1+1},$$

and since maximizing a linear function over all vertices is equivalent to maximizing over the convex body, this last is equivalent to

$$\max (\Pi_0 c + \overline{\Pi} A_1) \cdot x + \Pi_{m_1+1},$$
$$\text{subject to } A_2x = b_2,$$

which is an ordinary linear programming problem. Thus, by means of this technique, a problem of m_1 plus m_2 rows is converted into an m_1 row problem with an m_2 row problem being solved as the column selection routine.

There are many applications of this form of decomposition. It lends itself particularly well to solving the inequality systems that arise from models of large, but only slightly interacting systems. Examples of these are large firms with semi-autonomous divisions. Many of the activities of these divisions do not affect the others. Production in the various divisions may depend on different labor supplies and may go to different markets. However, there are certain inputs for which the divisions may compete; they generally draw on the same fund of capital for expansion; they compete for a share of a total company

budget which has a limitation on the total. Because of these interactions, the matrices which result from these company models often have this form

$$\begin{pmatrix} A_{01} & A_{02} \cdots A_{0,n} \\ A_{11} & 0 & 0 \\ 0 & A_{22} & 0 \\ 0 & 0 & A_{n,n} \end{pmatrix}.$$

All the A_{ij} are blocks of coefficients. The $A_{0,j}$ are the coefficients of activities which affect the entire firm; the $A_{i,i}$, $i > 0$, contain the coefficients of activities whose effect is felt only within the ith division. If the firm has many divisions, the number of rows in $A_{0,2}$ will be small in comparison with the total in A_{11}, A_{22}, etc. Consequently, if the problem is split into an upper part $A_1 = (A_{0,1}, \cdots, A_{0,n})$ and an A_2 consisting of the rest of the matrix, the A_1 part will be easily solved because it is small and the A_2 part because it consists of a number of parts, the A_{ii}, which (now that A_1 is removed) do not interact at all and can be dealt with as small separate problems.

Nonconvex problems. Convex optimization problems form the usual subject matter of linear programming. This is inevitable since the simplex method, the heart of the subject, is a gradient method, although a very elegant and economical one, and is therefore only capable of discovering local optima. It is, however, possible to bring nonconvex problems into this framework by convexifying them. A prototype of this approach is the method used for integer programming problems in [12; 13].

The integer programming problem is

$$\begin{gathered} \text{maximize } cx, \\ \text{subject to } Ax \leqq b,\ x \geqq 0, \\ \text{and all components of } x \text{ integers.} \end{gathered} \tag{11}$$

Here, we are maximizing over a very nonconvex set, the lattice points within the convex $Ax \leqq b$, $x \geqq 0$. The convexification process involves adjoining to the system (11) additional inequalities satisfied by integer solutions to the inequalities of (11) but not necessarily by all non-integer solutions. In principle, if enough of these were added, the convex hull of the lattice points satisfying (11) could be obtained, and the problem would become an ordinary one of linear programming. Actually, not all of the inequalities giving the convex hull are available at the start and the methods used attempt to add inequalities on a more selective basis. The source of the additional inequalities is as follows. Consider one of the inequalities of (11)

$$\sum a_{ij}x_j \leqq b_j. \tag{12}$$

From any one of these inequalities, one obtains

$$\sum [a_{ij}/\lambda]x_j \leqq [b_j/\lambda], \tag{13}$$

where the square bracket means integer part of, and λ is an arbitrary number $\geqq 1$.

This new inequality, as the reasoning of [13] shows, will be satisfied by any non-negative integer solution to (12) and, thus, can be added to the problem. Weighted sums of inequalities can also be treated in this way, and one can show that by combining new and old inequalities, a set giving the convex hull of the lattice points is eventually obtained.

The actual algorithms do not do this, but rather produce new inequalities one at a time in such a way that the pivot element is always 1 and the integer character of the matrix is always maintained.

Essentially then, an integer problem such as (11) is equivalent to a linear problem with an enormous number of rows. Most of the rows are not given (they would define the convex hull of the lattice points) but have to be developed in the course of the calculation.

This has the following consequences.

(1) Because of the large number of rows, these problems are done by row transformations.

(2) Because the minimization over all rows is necessary for a primal simplex calculation, and all rows are not available, it has been difficult to develop a primal simplex algorithm. (Although one has very recently come into existence, it is complicated.)

(3) The dual simplex method can be used because it does not require complete row minimization. However, this inability to minimize provides a weaker algorithm even in the dual simplex case, and may be responsible for the erratic computational behavior found in computer runs using integer programming.

Decomposition of Benders. In this example, we will have right multiplication and a linear minimization problem. Actually, the method of Benders, in its simplest form, turns out to be the exact dual (in the linear programming sense) of the Dantzig–Wolfe decomposition. However, because we end up doing the dual simplex method, which is advantageous for integer programming, we are able to use Benders' method for problems involving integers and, hence, for other nonconvex problems. The reference here is Benders [8].

Let us now consider a problem in which the variables are split into two groups x and y, i.e.,

$$\begin{gathered} \max z = c_1x + c_2y, \\ \text{subject to } A_1x + A_2y = b, \\ x \geqq 0,\ y \geqq 0. \end{gathered} \tag{14}$$

Grouping z and y on the right, the problem can be restated as: find the vector (z,y), $y \geqq 0$ and with largest z such that the equations

$$\begin{aligned} c_1x &= z - c_2y, \\ A_1x &= b - A_2y, \end{aligned} \tag{15}$$

have a non-negative solution x.

Denoting the columns of the matrix

$$\begin{pmatrix} C_1 \\ A_1 \end{pmatrix}$$

by $c_1, \cdots, c_N$, and the vector on the right in (15) by v, we know by Farkas' theorem that the equations (15) are solvable for non-negative x if and only if every vector Π making a non-negative scalar product with the c_i also has a non-negative scalar product with v. Let us assume that a normalizing condition such as $\sum \Pi_i = 1$, together with the conditions $\Pi c_i \geqq 0$ all i, gives us a bounded polyhedron of values of Π. Then there is a finite list $\Pi^1, \Pi^2, \cdots, \Pi^N$ of extreme vectors Π^i such that all Π satisfying $\Pi c_i \geqq 0$ and $\sum \Pi_i = 1$ are convex combinations of the extreme Π. Then

$$\Pi^i \cdot v \geqq 0, \text{ all } i,$$

already is necessary and sufficient for the solvability of (15). Thus, (14) becomes

$$\begin{aligned} &\max z, \\ &\text{subject to } y \geqq 0, \text{ and} \\ &\Pi^i \cdot \begin{pmatrix} z - c_2 y \\ b - A_2 y \end{pmatrix} \geqq 0, \text{ all } i. \end{aligned} \tag{16}$$

Now we have a problem only involving the variables z and y but, just as in our second example, with an enormously long list of inequalities.

Now we need do our Gaussian eliminations only on a square matrix whose side is one more than the dimension of y. Column selection involves no difficulty. Row selection, if one uses the dual simplex method, consists of solving the linear programming problem:

$$\begin{aligned} &\min \Pi \cdot v, \\ &\text{subject to } \Pi \cdot \begin{pmatrix} C_1 \\ A_1 \end{pmatrix} \geqq 0, \end{aligned} \tag{17}$$

and v is the right-hand side of (15) that results from the current values of z and y. This process is exactly dual to the Dantzig–Wolfe decomposition. If the primal simplex method is employed, then Step (B) involves a ratio minimization. In this case, the selection algorithm is again of the linear programming type, but this time the objective function is rational. Although this is not as familiar a problem, it is one that presents no essential differences and is solved by very slight variations from the usual simplex procedure.

One valuable feature of this form of decomposition is that it is capable of handling problems that are partly integer ones. For example, we may restrict the variables z and y to be integers. This converts (14) into a problem involving some integer and some continuous variables. Ordinarily to solve this, one would have to add inequalities to (14) which would be regarded as a problem having a great many unwritten rows defining its convex hull. Multiplication on the right would be a matrix of size dim (x) + dim (y), and in the course of calculation A_1 and A_2 enter together and whatever special structure A_1 has alone gets hopelessly lost.

The system (16), however, involves only integer variables, and the special structure of A_1 is preserved as A_1 appears alone in (17) when doing Step (B). For example, plant location problems involving balancing the economies of scale inherent in a few large plants against the costs of transportation to a large number of ultimate destinations for the finished goods. The decision as to whether or not to build and incur certain fixed costs at a given location can be represented by an integer variable, the variable costs of production and transportation involve continuous variables. This results in a system (14) in which A_1 is large ($n \cdot m$ variables if there are n plants and m destinations) and of the special transportation matrix forms. A_2 contains only $n + 1$ variables. The system (16) then contains only $n + 1$ integer variables, multiplication is by an $(n + 1) \times (n + 1)$ matrix, and the large A_1 part is treated separately by special transportation problem methods. This approach to large nonconvex problems through this form of decomposition seems very promising.

Application to a model of A. S. Manne. As a final example of the use of these techniques, we will consider a production planning model due to A. S. Manne [14]. We consider a plant making a rather large variety of different products $p_1, \cdots, p_S$. The demand $d_{i,t}$ for each product in each time period is known for the next T periods. The plant can make many products simultaneously and would like, for reasons of economy, to arrange production so that the load on the labor force in the various periods is somewhat smoothed out.

A feasible production plan for an individual product is one that, if used, will produce the necessary amounts of the product on or before the times they are needed. Let the amount produced by the jth plan of production for the ith product in period t be $a_{i,j,t}$ and let the labor required to produce that amount be $b_{i,j,t}$.

Then we can write down the equations.

$$\sum_{i,j} b_{i,j,t} x_{i,j} = l_t + s_t, \qquad t = 1, \cdots, T, \tag{18a}$$

$$\sum_{j} x_{i,j} = 1, \qquad i = 1, \cdots, S, \tag{18b}$$

$$x_{ij} \geqq 0. \tag{18c}$$

The x_{ij} requires some interpretation. There is one for each feasible plan for each product. If the x_{ij} are integers (18b) and (18c), assure that they can only take on values 0 or 1, and, in fact, exactly one of the x_{ij} for fixed i will have value 1, the others will be zero. Thus, if the x_{ij} are integers and satisfy (18b) and (18c), they can be interpreted as picking out from the list of feasible plans for each product exactly one that is to be followed. With this interpretation of the x_{ij}, the (18a) are the equations of labor balance in each period, i.e., on the left, we have the labor required (in man hours, say) and on the right, the labor available split into the amount l_t available from the regular labor force plus the amount s_t

of overtime or extra labor required. The problem will be to find integer x_{ij} that satisfy the equations and minimize

$$\sum_{t=1}^{t=T} s_t.$$

In words, select a production plan for each product that involves the smallest total of overtime labor.

Quite aside from the problem of getting integer x_{ij}, we already have a formidable linear programming problem on our hands. There are $T + S$ rows, and while $T = 10$ is reasonable for many situations, S, the number of products may be a few hundred. The number of columns, however, is really overwhelming; there is one for each feasible plan for each product.

Leaving aside the problem of integer x_{ij} for the moment, let us tackle the problem using the methods for large linear programs (Dzielinsky and Gomory [7]). We first apply a Dantzig–Wolfe decomposition splitting the matrix into an upper part A_1 consisting of (18a) and a lower part A_2 consisting of (18b). We can now deal with a problem that has only $T + S$ rows and, hence, we will do our Gaussian eliminations on a $(T + S) \times (T + S)$ matrix. However, when we come to the column selection, we find that the linear programming problem to be solved is, in the notation of our previous section,

$$\begin{aligned} &\max \bar{p}_1 A_1 \cdot x, \\ &\text{subject to } A_2 x = b. \end{aligned} \tag{19}$$

Although this is apparently almost as large as the original problem, it can be solved much more easily. If we denote by $A_{i,j}$ the column of A_1 corresponding to the variable x_{ij}, we see that (19) splits up into a series of separate sub-problems each of the form

$$\max (\bar{p}_1 A_{i1}, \bar{p}_1 A_{i2}, \cdots, \bar{p}_1 A_{iN_i}) \cdot (x_{i1}, \cdots, x_{iN_i}),$$
$$\text{subject to } \sum_{j=1}^{j=N_i} x_{ij} = 1.$$

The solution is simply to set $x_{ij} = 1$ for the j value for which $\bar{p}_1 A_{ij}$ is maximal, and to set all other $x_{ij} = 0$. Thus, the problem reduces to that of finding, from among all possible feasible schedules for the ith product, the one for which $\bar{p}_1 A_{ij}$ is maximal. This can be done by a recursive (dynamic programming) calculation of the Wagner–Whitin type [15]. To see this, let β_i be the production function for the ith product. β_i gives labor required as a function of output so

$$\beta_i(a_{i,j,t}) = b_{i,j,t}.$$

Define recursively

$$\begin{aligned} c_{i,t}(y) &= \min_{x_t} \{-\Pi_t \beta_i(x_t) + c_{i,t-1}(y + d_{i,t} - x_t)\}, \\ c_{i,1}(y) &= -\Pi_1 \beta_1(d_{i,1} + y). \end{aligned} \tag{20}$$

$c_{i,t}(y)$ can be interpreted as the minimum cost of meeting demand for the ith product through period t and having an excess production of amount y at the end of the tth period. The Π_t, components of $\bar{p}_1$, are all nonpositive and $-\Pi_t$ can be interpreted as unit labor costs in each period under the present production plan. x_t is the amount produced each period. By use of the recursion (20), we find the new schedule with smallest labor cost. If β_i is concave, or especially if it is linear except for a jump at the origin (set up cost), the computational labor in (20) can be sharply reduced.

Thus, by a Dantzig–Wolfe decomposition followed by an application of the column generating technique of our first example, a problem originally calling for Gaussian eliminations on a $(T + S) \times (T + S)$ or 210×210 matrix, and the investigation of scalar products with columns as numerous as schedules, is reduced to Gaussian eliminations on a $(T + 1) \times (T + 1)$ or 11×11 matrix, and a string of S associated dynamic programming calculations.

We turn now to the question of integers. Fortunately, this tends to take care of itself. For in the system (18b) and (18c), there are $T + S$ equations and, therefore, at most, $T + S$ positive variables in the optimal solution. There must be at least one positive variable appearing with nonzero coefficient in each of the equations (18b). This accounts for S of the $T + S$ positive variables, so there are at most T values of i for which more than one x_{ij} is positive. If S is much larger than T, this means that almost always only one x_{ij} is positive for fixed i. Therefore, it must be integer with value 1. Thus, there are at worst T noninteger variables to be treated by some arbitrary rounding process.

References

1. George B. Dantzig and Philip Wolfe, *Decomposition principle for linear programs*, Operations Res. **8** (1960), 101–111.
2. L. R. Ford, Jr. and D. R. Fulkerson, *A suggested computation for maximal multi-commodity network flows*, Management Sci. **5** (1958), 97–101.
3. George B. Dantzig, *Linear programming and extensions*, Chapter 22, Princeton Univ. Press, Princeton, N.J. (to appear).
4. ———, *A machine-job scheduling model*, Management Sci. **6** (1960), 191–196.
5. P. C. Gilmore and R. E. Gomory, *A linear programming approach to the cutting-stock problem*, Operations Res. **9** (1961), 849–859.
6. R. E. Gomory and T. C. Hu, *An application of generalized linear programming to network flows*, J. Soc. Indust. Appl. Math. **10** (1962), 260–283.
7. Bernard P. Dzielinsky and Ralph E. Gomory, *Lot size programming and the decomposition principle*, Abstract for the meeting of the Econometric Society, Ann Arbor, Michigan, 1962.
8. J. F. Benders, *Partitioning in mathematical programming*, Thesis, Utrecht Univ., Utrecht, 1960.
9. Kurt Eisemann, *The trim problem*, Management Sci. **3** (1957), 279–284.
10. R. Bellman, *Some applications of the theory of dynamic programming—a review*, Operations Res. **2** (1954), 275–288.
11. R. E. Gomory and T. C. Hu, *Multi-terminal network flows*, J. Soc. Indust. Appl. Math. **9** (1961), 551–570.

12. R. E. Gomory, *An algorithm for integer solutions to linear programs*, Recent advances in mathematical programming, McGraw-Hill, New York, 1963.

13. ———, *All-integer integer programming algorithm*, Industrial scheduling, Prentice-Hall, Englewood Cliffs, N. J., 1963.

14. Alan S. Manne, *Programming of economic lot sizes*, Management Sci. 4 (1958), 115–135.

15. Harvey M. Wagner and Thomson M. Whitin, *A dynamic version of the economic lot size model*, Management Sci. 5 (1958), 89–96.

International Business Machines Corporation,
Yorktown Heights, New York