A LINEAR PROGRAMMING APPROACH TO THE CUTTING STOCK PROBLEM - PART II*

by

P. C. Gilmore
R. E. Gomory

* This paper appeared in Operations Research, v. 11, no. 6, Nov.-Dec. 1963, pp. 863-888. See following page for abstract and ONR contract support information.

A LINEAR PROGRAMMING APPROACH TO THE CUTTING STOCK PROBLEM - PART II*

by

P. C. Gilmore
R. E. Gomory

International Business Machines Corporation
Thomas J. Watson Research Center
Yorktown Heights, New York

ABSTRACT: In this paper, the methods for stock cutting outlined in an earlier paper in this journal are extended and adapted to the specific full-scale paper trim problem. The paper describes a new and faster knapsack method, formulation changes, and experiments such as one giving waste as a function of stock length. The methods developed are also applicable to a variety of cutting problems outside of the paper industry.

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

Research Paper
RC-949
May 27, 1963

I. INTRODUCTION

In an earlier paper [1] we outlined a column generating procedure to overcome one of the basic difficulties associated with the cutting stock problem, the problem of too many cutting patterns, or in linear programming terms the problem of too many columns. In this paper we describe the adaptation and application of this method to the specific problem of paper trim. The changes that were required to make the application possible were of two kinds, changes in the algorithm itself and changes in formulation, and are described below. The final algorithm can be used in many other situations such as the slitting of steel rolls, cutting of metal pipe, cellophane roll slitting, etc.

(a) Algorithm Changes. The large size of many of the paper industry problems has required a whole series of modifications which were needed to get the calculation done more rapidly or even to do it at all on the small computers that are usually available in a paper mill. For this purpose we evolved a new and rapid algorithm for the knapsack problem, which proved to be markedly superior to dynamic programming, developed ways of cutting down the size of the knapsack problem, methods of speeding convergence of the linear programming calculation, and a reasonable cutoff. The new calculations are sufficiently rapid that real problems can now be solved and with a considerable saving in paper. Programs written by

W. E. Winans[1] and Darrell Murff[2] are being used in a routine manner at paper mills on computers installed specifically to deal with the paper trim problem.[3]

(b) Changes in the Formulation. Special circumstances in the paper industry provide limits on the number of cutting knives, bring in machine balance considerations, and introduce a tolerance or indeterminancy in the customers' orders. We will show how these conditions can be dealt with as modifications of the knapsack problem, modifications of the linear programming formulation, or by a programming formulation involving a rational objective function.

Finally, we will give some experiments, made possible by the completed algorithm, showing the economy that can be obtained from the use of more than one stock length, indicating the extreme sensitivity of the waste to the exact stock length available, and showing the effect of limiting the number of cutting knives.

---

[1] IBM, Green Bay, Wisconsin.

[2] IBM, Tyler, Texas.

[3] The parent program for these 1620 programs was a 7090 and 7094 FORTRAN program written by Mrs. C. S. Wade of the T. J. Watson Research Center. The 7094 program was used in the tests in this paper. We are greatly indebted to Mrs. Wade for excellent programming and for many helpful suggestions.

II. CHANGES IN THE ALGORITHM

The cutting stock problem in mathematical form is the problem of minimizing $\sum_j x_j$, subject to

$$\sum_j a_{ij} x_j \geq N_i$$

where N_i, $i = 1, \ldots, m$ is the number of rolls of length ℓ_i demanded, x_j is the number of times the jth cutting pattern is used, and $a_{i,j}$ is the number of rolls of length ℓ_i produced each time the jth way of cutting up a roll is used.

If there are different length parent rolls available to be cut up, their costs may be different. In this case, the objective function becomes $\sum_j c_j x_j$, where c_j is the cost of the parent roll from which the jth cutting pattern is cut.

The difficulty of the problem lies in the large number of cutting patterns (columns) that may be encountered. For example, with a standard roll of 200" and demands for 40 different lengths ranging from 20" to 80", the number of cutting patterns can easily exceed ten or even 100 million. And this size problem is encountered in practice. Thus, we are faced with a linear programming problem involving ten or 100 million columns.

In an earlier paper [1], we outlined a column generating procedure that in principle enables one to avoid this difficulty in the sense that one calculates alternately with an $m \times m$ matrix (the basis inverse) and a subroutine that produces the next column when

needed, and never with the entire list of columns.

To produce the next column, one has to find that column whose scalar product with the current linear programming prices is maximal. Since any string of nonnegative integers a_i, $i = 1, \ldots, m$ is a column provided only that

$$\sum_{i=1}^{i=m} \ell_i a_i \leq L,$$

where L is the standard stock length and ℓ_i the demanded lengths, the maximization problem is

$$(1) \qquad \begin{aligned} &\text{maximize} \quad \sum_{i=1}^{i=m} b_i a_i \\ &\text{subject to} \quad \sum_{i=1}^{i=m} \ell_i a_i \leq L. \end{aligned}$$

Here the b_i are the current linear programming prices, and the a_i must be nonnegative integers. Following Dantzig [4], we call (1) the knapsack problem. If the maximum $\bar{M}$ in (1) is greater than the cost c of the standard stock length L, an improving column has been found, if $\bar{M} \leq c$, no improvement is possible and the cutting stock problem has been solved. If there are several standard stock lengths L_i with costs c_i, several problems like (1) must be solved.

In [1] we outlined a method which consisted of using a short-cut method for a solution followed by dynamic programming in the

event the short-cut method did not produce a solution. Unfortunately, even dynamic programming is often too ponderous for the size of problem generated by the paper industry. Consider a problem involving a 200" parent roll and orders specifying lengths to 1/8 of an inch as is customary. The fundamental recursion of the dynamic programming calculation is

$$F_{s+1}(x) = \max_{r} \{r\, b_{s+1} + F_s(x - r\ell_{s+1})\}, \tag{2}$$
$$0 \leq r \leq [x/\ell_{s+1}]$$

where $F_{s+1}(x)$ has the intuitive interpretation of being the value of the most valuable combination that can be fitted into a knapsack of length x if only the first $s+1$ lengths can be used.

In carrying out this recursion, most of the time will be spent on the inner loop of the calculation which will involve

(1) multiplying b_{s+1} times r,

(2) fetching and adding $F_s(x - r\ell_{s+1})$ to $r\, b_{s+1}$,

(3) subtracting the current maximum of the expression in parentheses in Equation (2) from the sum in Step (2),

(4) making a conditional transfer depending on whether or not this difference is negative,

(5) adjusting an index and returning to Step (1).

In programming this, we seem to require 10 elementary machine

orders (orders such as add), so we will allot to this calculation the time 10T. (Reasonable T's are for the 7090, 4.5 microseconds, for the 7094, 4 microseconds, and for the 1620, 470 microseconds.[1]) This loop must be gone through $[L/\ell_{s+1}]$ times and then repeated 8L times if L is in inches and order lengths are given to 1/8" as is customary. This provides $F_{s+1}(x)$ for all $x \leq L$. This routine is then repeated m times if there are m lengths, in order to solve one knapsack problem. A reasonable average figure for $[L/\ell_s]$ is 5 for paper problems that we have seen, so the time per knapsack may be estimated as $10 \times 8 \times 5$ LmT. With an L of 200 and m = 30, we get 2.4×10^6T. If we find the linear programming calculation requires 2.5 m simplex steps, which is both a conventional estimate and about what we have actually observed, we get a time for the knapsack parts of one problem of $1000\ LTm^2$. For the thirty length problem, this is 16×10^7 T.

Because of the necessity of using the tape in doing the backtracking part of the dynamic programming calculation, our actual dynamic programming runs have exceeded twice this estimate.[2]

---

1 This is a weighted average of the add and multiply times actually needed for the loop.

2 The actual runs on the 7090 for a single dynamic programming calculation were 18 seconds with 25 lengths and L = 150, and 24 seconds with 40 lengths and L = 150. In the first problem, the tape time was about 7 seconds, in the second about $10\frac{1}{2}$ seconds.

However, even this estimate gives about a 13 minute run on the 7090 and a run of 1357 minutes on the 1620 for a typical 30 length, 200 inch roll problem.

These figures show the need for a faster method for the knapsack problem if paper industry problems are to be dealt with. A method related to [6] and which has proved to be about five times as fast as dynamic programming for our set of problems, is described next.

Knapsack Method

In what follows, we will denote the maximum of the cost c and of the previously defined $\bar{M}$ by M. If there are several standard stock lengths L_i, it is necessary to solve a series of knapsack problems like (1) with L replaced by L_i. We will denote the corresponding maxima of c_j and $\bar{M}_j$ by M_j. The algorithm we have employed for the knapsack problem will first be described in a form which permits one to calculate all the M_j simultaneously. Following the description and justification of the algorithm, we will show how it can be modified to calculate $\max \{M_j - c_j\}$ with possibly less computation.

The steps of the calculation are as follows:

(1) Reorder the variables $a_1, a_2, \ldots, a_m$ so that $b_1/\ell_1 \geq b_2/\ell_2 \geq \ldots \geq b_m/\ell_m$, and reorder the stock lengths so that $L_1 > L_2 > \ldots > L_k$. Introduce a variable a_{m+1} with coefficients $b_{m+1} = 0$ and $\ell_{m+1} = 1$.

Let λ and β be the m-vectors with coefficients $\ell_1, \ell_2, \ldots, \ell_m$ and $b_1, b_2, \ldots, b_m$ respectively. For an s-vector $(\alpha)_s$ of nonnegative integers $a_1, a_2, \ldots, a_s$, where $1 \leq s \leq m$, by $\lambda \cdot (\alpha)_s$ is meant $\sum_{i=1}^{s} \ell_i a_i$ and by $\beta \cdot (\alpha)_s$ is meant $\sum_{i=1}^{s} b_i a_i$. A vector $(\alpha)_m$ is an extension of a vector $(\alpha)_s$, $s \leq m$, if the first s coefficients of $(\alpha)_m$ are just the coefficients of $(\alpha)_s$.

In the algorithm a sequence of vectors $(\alpha)_s$, for various values of s, satisfying $L_1 \geq \lambda \cdot (\alpha)_s$ is generated in lexicographically decreasing order, where $(\alpha^1)_{s_1}$ is lexicographically larger than $(\alpha^2)_{s_2}$ if and only if for some i, $1 \leq i < \min \{s_1, s_2\}$, $a_1^1 = a_2^2, \ldots, a_i^1 = a_i^2$, while $a_{i+1}^1 > a_{i+1}^2$. A simple test (Step (5)) permits one to ignore many of the possible vectors. The first vector $(\alpha)_m$ in the sequence is the lexicographically largest m-vector satisfying $L_j \geq \lambda \cdot (\alpha)_m$, for any j. That is,

(2) let $a_1 = [L_1/\ell_1]$, $a_2 = [(L_1 - \ell_1 a_1)/\ell_2], \ldots,$ and $a_m = [(L_1 - (\ell_1 a_1 + \ldots + \ell_{m-1} a_{m-1}))/\ell_m]$. Let $t = 1$ and let $M_j = c_j$, $j = 1, \ldots, k$.

This vector $(\alpha)_m$ is then tested to determine whether for it $\beta\cdot(\alpha)_m$ exceeds the current best values M_j for applicable j, and if so these current best values are redefined.

(3) For those j, $t \leq j \leq k$, for which $L_j \geq \lambda\cdot(\alpha)_m$ and $\beta\cdot(\alpha)_m > M_j$, redefine M_j to be $\beta\cdot(\alpha)_m$.

(4) Let s be the largest i, $1 \leq i \leq m$, such that $a_i \neq 0$.

Thus $(\alpha)_s$ has among its coefficients all the non-zero coefficients of $(\alpha)_m$ and its last coefficient is non-zero. The lexicographically largest m-vector lexicographically smaller than $(\alpha)_m$ necessarily has its sth coefficient one less than a_s; that is, it is an extension of a vector $(\alpha^1)_s$ which differs from $(\alpha)_s$ only in having $a_s - 1$ as its sth coefficient. In the first part of (5), $(\alpha)_s$ is redefined to be $(\alpha^1)_s$.

In the test in the second part of Step (5) below, one determines if it is possible for any extension of $(\alpha)_s$ to lead to an improvement of at least one of the current maximums M_j. A necessary condition for an improvement of M_j to be possible, assuming that $L_j \geq \lambda\cdot(\alpha)_s$, is that an improvement result when the integer restriction on a_{s+1} is relaxed and a_{s+1} set equal to $(L_j - \lambda\cdot(\alpha)_s)/\ell_{s+1}$. Because of the ordering chosen for the variables, the condition for an improvement is that

$\beta \cdot (\alpha)_s + b_{s+1} (L_j - \lambda \cdot (\alpha)_s)/\ell_{s+1} > M_j$. This necessary condition is tested for in (5); if it holds for some j one transfers to (7) and calculates the lexicographically largest extension $(\alpha)_m$ of $(\alpha)_s$ which satisfies $L_t \geq \lambda \cdot (\alpha)_s$, where t is the smallest integer j for which the condition holds for the jth stock length; if it fails to hold for any j, then a successor of $(\alpha)_s$ in the sequence of vectors, if there is one, is calculated in (6).

A successor to $(\alpha)_s$ is calculated in (6) only when $s > 1$. Then the successor is the lexicographically largest $(s-1)$-vector which is lexicographically smaller than $(\alpha)_s$. This successor is chosen because from the failure of the necessary condition in (5), one can conclude that it is the lexicographically largest $(s-1)$-vector which may have an extension improving at least one of the current maximums M_j.

(5) Redefine a_s to be $a_s - 1$ and let t be the smallest j, $1 \leq j \leq k$, such that $L_j \geq \lambda \cdot (\alpha)_s$ and such that $(L_j - \lambda \cdot (\alpha)_s)\, b_{s+1} > (M_j - \beta \cdot (\alpha)_s)\, \ell_{s+1}$, and go to (7). If there is no such j then go to (6).

(6) Redefine s to be the largest i, $1 \leq i \leq s-1$, such that $a_i \neq 0$, and go to (5). If there is no such i, then the current values of M_j, $j = 1, \ldots, k$, are the maximums to be found.

(7) Let $a_{s+1} = [(L_t - \lambda \cdot (\alpha)_s)/\ell_{s+1}], \ldots, a_m = [(L_t - (\lambda \cdot (\alpha_s) + \ell_{s+1} a_{s+1} + \ldots + \ell_{m-1} a_{m-1}))/\ell_m]$, and go to (3).

That the final values of the maxima M_j are the desired maxima when the algorithm terminates in Step (6) is clear from the order in which the m-vectors are generated and then tested in (3) for possible improvement of the M_j. For the first vector tested in (3) is the lexicographically largest vector $(\alpha)_m$ which can satisfy $L_1 \geq \lambda\cdot(\alpha)_m$ and $\beta\cdot(\alpha)_m > c_j$, for some j, and each succeeding vector tested in (3) is the lexicographically next largest m-vector which can satisfy $L_t \geq \lambda\cdot(\alpha)_m$ and $\beta\cdot(\alpha)_m > M_j$, for some j.

In order to determine the cutting pattern $a_1, a_2, \ldots, a_m$ which actually maximizes M_j, it is only necessary to keep a record for each j of how the current maximum has been obtained and update these records each time an improvement is made in (3) in the current maximums.

The modification that can be brought to the algorithm to compute $\max \{M_j - c_j\}$ rather than M_j, $j = 1, \ldots, k$, can now be simply described. The modification is only in Step (3), which should be replaced with:

(3') For those j, $t \leq j \leq k$, for which $L_j \geq \lambda\cdot(\alpha)_m$, let $\Delta M_j = \beta\cdot(\alpha)_m - M_j$, and let $\Delta M = \max \{0, \Delta M_j / j = t, \ldots, k\}$. Redefine M_j to be $M_j + \Delta M$, for $j = 1, 2, \ldots, k$.

Identical Prices

In our knapsack discussion so far we have tried to speed up the calculation by using a more rapid knapsack method. Another approach is to speed up through cutting down on the size of knapsack problem that must be solved. We have been able to do this because of the phenomenon of identical prices.

Clearly, identical prices, if they occur, can be exploited in the knapsack problem. For if two lengths ℓ_i and ℓ_j, $\ell_i < \ell_j$, receive identical linear programming prices, then the density b_i/ℓ_i associated with ℓ_i is greater, and since ℓ_i is shorter, it can always be substituted for ℓ_j in (1). So it is unnecessary in the maximization to consider ℓ_j or the variable a_j at all. Thus we need deal only with the smaller knapsack problem in which, among lengths having identical prices, only the shortest length is considered.

Normally in linear programming one would not expect identical prices any more than one would expect duplication of the values of the primal variables. Nevertheless, we see from Figure 1, which gives the size of the knapsack problem actually solved, that such duplication occurs extensively. The behavior shown in the figure is typical of a low waste problem. Duplication is even greater in high waste problems. In our problem set identical prices reduced the average size of the knapsack problems from 30 to 18.2 variables. What happens is that lengths that are nearly the same receive exactly the same price.

Although we do not really understand this phenomenon, we will advance a rather unsatisfactory partial explanation. Consider a stage in the linear programming calculation at which the average waste per roll is still two or three inches. This could be either early in the calculation before a nearly optimal answer is reached, or at any stage of a problem containing considerable irreducible waste. Consider any cutting pattern in the basis and let us suppose it cuts non-zero quantities a_{i_1}, a_{i_p} of the lengths $\ell_{i_1}, \ell_{i_2}, \ldots, \ell_{i_p}$. If ℓ'_{i_1}, etc. are lengths near the ℓ_{i_j}, then, because of the available waste, the same pattern, but with the ℓ'_{i_j} substituted for the ℓ_{i_j} will still fit in the knapsack. If the same price is given to the ℓ'_{i_j} as to the ℓ_{i_j}, the new pattern will price out to zero just as the old one did. This is consistent with its being in the basis and being used.

However poor the explanation, there is no doubt about the reality of the phenomenon or of the substantial help it gives in cutting down the calculation.

Test Problems

In the following sections, we will often have occasion to refer to our test problems. We had four series of test problems, Series A1, A2, A3 and A4. Each series contained one 40 length, one

35 length, one 30 length, one 25 length and one 20 length problem. In addition, one 50 length problem A5-1 was used in some experiments. The results on number of pivots, running times (on the IBM 7094) and final waste in the answer are given in Column A of Figure 2. These results were obtained with the program containing the new knapsack method, exploiting identical prices, and using the various devices mentioned below. In order to obtain computing times under usual conditions, a cutting knife limitation, described in III below, was imposed on these runs. The limitations for the Series A1 to A5 are respectively 7, 5, 5, 9, and 7. Results when some of these devices are omitted will be described in the appropriate sections.

Problem A2-3, a 30 length problem is exhibited in Figure 3, together with its solution in order to give some idea of the range of demand, range in the lengths demanded, etc. in the test problems. The test problems have been made up by amalgamating and selecting from a class of paper industry problems.

Median Method

The whole purpose of the knapsack calculation is to find a column representing a cutting pattern that will yield an improvement; in fact, by maximizing the knapsack calculation we find the column that will yield the most improvement per unit increase in the associ-

ated variable x_j (call this amount s_j). However, this criterion pays no attention at all to the question of how big x_j will be when the pivot step is completed, and the improvement in the objective function depends on this as the improvement is the product $x_j s_j$. We have found that much too often in our calculations a large s_j led only to a small improvement, while a much smaller s_j, but associated with a large change in x_j, led to a large improvement. Although it is computationally tedious to try and do anything about this in the general case of linear programming, in our situation there is a simple interpretation that leads to an intuitively obvious method of improvement.

To see this we must observe that in our problems there is often a very wide range of demand for the different lengths ℓ_j (see Example 1). For simplicity, let us imagine that we can divide the lengths into two groups, the high demand group and the low demand group. Now, if in doing our knapsack calculation we pay no attention to this distinction (which is what we have described doing so far), we are (with high probability) going to have some high and some low demand lengths represented in the cutting pattern that is generated. If the smallest demand among the lengths represented in the cutting pattern is N_i, then we are almost certain to have the increase in x_j bounded by N_i, for otherwise, by the use of this one cutting pattern alone, we will already be producing more of this length than is actually needed.

Consequently, any cutting pattern involving low demand lengths can only be used a small number of times, x_j must be small, and so cannot affect the objective function very strongly. The remedy is of course to actually divide the lengths into a high demand and a low demand group and sometimes do a knapsack problem restricted to the high demand group. In this way, the cutting pattern generated will at least have the possibility of being used a large number of times. What we have done is to divide demands into two groups, those having a demand above the median demand going into one group, those having a demand below the median demand going into the other. Then at every second pivot step, we confine the knapsack calculation to the variables of the high demand group. (If the confined knapsack problems could yield no improvement, we would revert to the large one.) The value of this device, which yielded a considerable improvement both on running time and on number of pivots, can be seen by comparing Columns A and B of Figure 2. On problems with long running times, it is of considerable assistance; on problems that run quickly anyway it does little, as one would expect from the foregoing explanation. Its performance on Series A4 is particularly striking.

Maximization

One obvious question to ask is if it is in fact worth while to go through the maximization procedure on the knapsack problem, or whether it is better to take the first improving column encountered in the knapsack algorithm and use that instead. One would certainly expect to have to take more simplex steps if this last procedure were adopted, but since most of the time spent in the calculation is spent doing the knapsack problem and not doing the Gaussian elimination step, a substantial saving in time in the knapsack subroutine could easily overbalance this. With this in mind, we omitted the maximization with the result shown in Column C of Figure 2. These computations seem to show a clear advantage in using the maximization algorithm. The advantage in reducing the number of pivots is quite large; the saving in running time is not as striking but still considerable, again especially on the longer run problems.

A Connection with Integer Programming

A further experiment was tried in which neither maximization nor the median device was used. This was done not so much in the hope of obtaining a faster calculation (this seemed unlikely), but rather to explore what happens in a linear programming situation involving millions of columns when an improving column is chosen more or less blindly.

This is almost identical with the situation encountered in integer programming. In integer programming [2] [3], a dual simplex method is used and there is a vast array of unwritten inequalities in the problem. (These are the extra inequalities that reduce the feasible space to the convex hull of the feasible lattice points.) No maximization over the unwritten inequalities (rows) is possible, nor is there anything corresponding to the median method. Since the dual method is used, rows play the same role that columns do in a primal method. Now, for many types of integer programming problems, there is a problem sensitivity unknown in ordinary linear programming. Some problems run easily, others drone on and on. We were interested in seeing if this was a characteristic of integer programming or whether it could occur in any linear programming situation involving a blind choice from a vast list of rows or columns.

The data we obtained did in fact show a sensitivity similar to integer programming data. (Figure 2, Column D.) The A1 series averaged over 20 m pivots for an m rowed problem against a conventional simplex estimate of 2.5 m. The A2 series averaged less than 4.4 m, while the A4 series was given an average of over 27 m pivots and only one problem of the five was complete at that point. Also problem A5-1, a fifty-row problem, required 2,123 pivot steps to complete.

These results suggest that the peculiarities of integer programming are not special to it but that they are to be expected in large linear programming problems where there is no effective method of choosing among the multitude of rows in the dual simplex case or among columns in the primal case.

Cutoff

Finally, we found that the calculation tended to slow down toward the end. Due to the large number of patterns that are implicitly there to be considered, there can be a very long tail to the calculation in which slightly better combinations keep being introduced but yield very little improvement. This is something we have encountered only in low waste problems; those with intrinsically high waste seem to stop quite quickly. This stage of the calculation also tends to take a long time per pivot, for with the low remaining waste, most of the identical prices have disappeared, and the knapsack problems to be solved are full size. Both the low rate of decrease of waste and the growth of the knapsack problems are shown for a typical problem in Figure 1. Consequently, it seemed worthwhile to develop a cutoff which would halt the calculation when it became too slow and unrewarding. The criterion used was to stop the calculation if ten pivot steps did not produce 1/10 % reduction in waste. This criterion, which seems to give very good results, was used in the experiment whose data appear

in Figure 4. The reduction in running time obtained was quite striking; the whole A1 series of five problems took a total of 2.28 minutes instead of 13.95 minutes with an increase in waste that averaged .26%.

We now turn to the formulation changes required by the special circumstances of the paper industry.

III. EXTENSIONS OF THE FORMULATION

Cutting Knife Limitation

Often the number of pieces into which a roll can be cut is limited by the fact there are only some fixed number R of cutting knives available (six is a typical number). This fact must be taken into account in the knapsack subroutine so that patterns involving more than R cuts will not be considered. The modification of the knapsack routine for the purpose is simple and is as follows: the components of the vectors $(\alpha)_m$ calculated in Steps (2) and (7) of the knapsack algorithm are calculated as indicated in the order $a_1, a_2, \ldots,$ except that after having calculated a_r one determines whether the inequality $\sum_{i=1}^{r} a_i < R$ is satisfied. If it is, one calculates a_{r+1}; if it is not and $\sum_{i=1}^{r} a_i > R$, one reduces a_r by the amount $\sum_{i=1}^{r} a_i - R$, and stops.

Machine Balance Problem

In this problem, we envisage several paper machines of different lengths being available to meet the demand. However, the demands cannot all go on one machine, even if it gives the least paper trim, since its production during the given time period is limited. Also, sometimes one may want to balance the production from the available machines so that one machine does not stand idly by while the others run, etc. These problems are all essentially alike and can be handled by slight variations on the previous formulation. We first consider the problem where the production (number of rolls) from each machine is limited. If there are P machines, with the rth machine capable of producing Q_r rolls of its length L_r, the linear programming problem becomes

$$\min \ \Sigma_{j,r} c_r x_{j,r}$$

$$\text{subject to} \quad \Sigma_{j,r} a_{i,j,r} \geq N_i$$

$$\text{and} \quad \Sigma_j x_{j,r} \leq Q_r .$$

When c_r is the cost (or length) of a roll from the rth machine, $x_{j,r}$ is the number of times the jth pattern is used on the rth machine, and $a_{i,j,r}$ is the number of rolls of length ℓ_i produced by this activity. This is a matrix, which, with one row for the objective function, has $(m + P + 1)$ rows and, hence, a $(m + P + 1) \times (m + P + 1)$ inverse. Its general appearance is as follows:

$$
\left(\begin{array}{cccccc|c}
c_1\ c_1 \cdots c_1 & & c_2\ c_2 \cdots c_2 & & c_3\ c_3 \cdots c_3 & & \geq N_1 \\
 & A_1 & & A_2 & & A_3 & \vdots \\
 & & & & & & \geq N_m \\
\hline
1\ 1 \cdots 1 & & & & & & \leq Q_1 \\
 & & 1\ 1 \cdots 1 & & & & \leq Q_2 \\
 & & & & 1\ 1 \cdots 1 & & \leq Q_3
\end{array}\right)
$$

The blocks A_r contain all cutting patterns that fit the length L_r. The top row of the $(m + P + 1) \times (m + P + 1)$ inverse now includes prices for the production limitation inequalities, as well as for the ordered lengths. Again, we try to find a column whose scalar product with this inverse top row will be positive. For columns taken from the rth block looking for this column is the problem

$$
\begin{aligned}
&\text{maximize} && c_k - \Sigma_i \Pi_i a_i + \bar{\Pi}_k \\
&\text{subject to} && \Sigma \ell_i a_i \leq L_k,
\end{aligned}
\tag{3}
$$

where $\bar{\Pi}_k$ is the price associated with the kth production limitation and the $a_1, \ldots, a_m$ are nonnegative integers.

The desired column is produced by taking the max over k of the result obtained from P different knapsack problems (3). Once the column is produced, the calculation proceeds exactly as in the simple stock cutting case.

Machine balance; i.e., equality between the amounts used by various machines, or lower as well as upper bounds on the production used from each machine, can be dealt with in the same manner.

Customer Tolerances

One of the special features of the paper industry is the fact that customer orders often do not have to be filled exactly. If a customer orders a quantity N_i of length L_i, he will accept a quantity within a certain range of N_i. A range of plus or minus 5% is very common. With the quantities to be produced confined to ranges rather than to exact amounts, the objective function must be changed, and generally ends up being a rational function. We will follow through the case where all cutting patterns (rolls) are given the same cost, the case where roll costs are different is no more difficult, and we will see that the column generating method can still be carried out and still involves a knapsack problem.

Although all rolls have the same cost, it is no longer adequate to take total waste as the objective function to be minimized, for generally total waste will go up as larger quantities are produced (to take advantage of the range inequalities), even though the percentage waste (which measures the efficiency of the operation) may go down. Consequently, we must introduce percentage waste which is no longer a linear, but rather a rational, objective function. Also,

we must introduce a waste explicitly into the cutting pattern vector. Formerly, we could assign the length of a roll as the cost of a cutting pattern, and by minimizing the total number of rolls required to fill the orders we automatically minimized waste. This is no longer possible when we deal with ranges and with percentage waste. Instead, we must give to the jth cutting pattern a waste w_j, where

$$w_j = L - \Sigma_i a_{ij} \ell_i \quad . \tag{4}$$

Waste, which will enter into the new objective function, is now linearly dependent on the entries in the cutting pattern. In spite of these changes, we will see that our methods still go through.

If the amount produced of length ℓ_i is to be between N_i' and N_i'' $(N_i' < N_i'')$, the new formulation is in the notation of [1].

$$\text{minimize} \qquad \zeta = z_1 / z_2 = \frac{\Sigma w_j x_j}{\Sigma x_j}$$

$$\text{subject to} \qquad \Sigma_j a_{ij} x_j - s_i = N_i' \quad \text{all } i$$

$$\text{and} \qquad 0 \leq s_i \leq (N_i'' - N_i').$$

As far as the equations are concerned, this formulation is no different from the earlier one except for the presence of an upper bound $N_i'' - N_i'$ on s_i. As methods of handling upper bounds without explicitly writing them are well known [5], we will not discuss this here and will pay no further attention to the upper bound restriction.

We now turn to the rational objective function.

The rational objective function in linear programming seems to be of some current interest. The method described here, which we show lends itself to a column generating technique, seems to be closest to that of Martos [7,8]. For other approaches, see Charnes and Cooper [9], Dinkelbach [10] and Dorn [11].

It is easy to show by direct differentiation that a rational function will have the property of uniformly increasing or decreasing along any line not passing through a point for which the denominator vanishes. More precisely, if at a point P the derivative of a rational function in the direction L through P is v, then the derivative in the direction L at any point of L will have the same value v unless L passes through a point of denominator zero.

It follows that the ordinary simplex method can be used to maximize a rational function (in a domain where the denominator doesn't vanish) for all we need to do is go from vertex to vertex in the usual way, inquiring at each vertex if any of the lines (edges) leading from it to a neighbor is a direction which improves the objective function. If no line yields any improvement, the calculation is finished. Only the testing for lines of improvement is different, but we will see that in our case it can again be reduced to a knapsack calculation.

To prepare for the calculation, we adjoin to the matrix representing the equations

$$\Sigma_i a_{ij} x_j - s_j = N_i'$$

two new rows and two unit columns representing two new auxiliary objective functions z_1 and z_2.

$$z_1 - \Sigma w_j x_j = 0$$

$$z_2 - \Sigma x_j = 0.$$

Now the column corresponding to the jth cutting pattern will have an entry $-w_j$ in the top row and a -1 in the second row and the usual entries $a_{i,j}$ in the remaining rows.

At any point in the calculation we will have a $(m + 2) \times (m + 2)$ basis inverse whose top row will be of the form $(1, 0, \Pi')$ with Π' an m-vector. The scalar product of this row with the column c_j of the jth cutting pattern gives as usual $-\frac{dz_1}{dx_j}$. This is the negative of the rate of change of z_1 along the edge that would be traced out if x_j were increased, but all other nonbasic variables kept zero. Similarly, the second row is of the form $(0, 1, \Pi^2)$ and gives as its scalar product with c_j, $-\frac{dz_2}{dx_j}$.

Since $\zeta = z_1/z_2$

$$(5) \qquad \frac{d\zeta}{dx_j} = \frac{z_2 \frac{dz_1}{dx_j} - z_1 \frac{dz_2}{dx_j}}{z_2^2}$$

$$= \frac{1}{z_2^2} \left[z_1 (0, 1, \Pi^2) \cdot c_j - z_2 (1, 0, \Pi^1) \cdot c_j \right]$$

$$= \frac{1}{z_2^2} \left[w_j z_2 - z_1 + \Sigma_i \, (z_1 \Pi_i^2 - z_2 \Pi_i^1) \, a_{i,j} \right].$$

Substituting the expression (4) for w_j we get

$$\frac{d\zeta}{dx_j} = \frac{1}{z_2^2} \left[L \, z_2 - z_1 + \Sigma_i (z_1 \Pi_i^2 - z_2 \Pi_i^1 - \ell_i) \, a_{ij} \right].$$

If this expression is negative, increasing x_j will decrease the objective function, otherwise not. Setting

$$k = L \, z_2 - z_1$$

and

$$\overline{\Pi}_i = - (z_1 \Pi_i^2 - z_2 \Pi_i^1 - \ell_i)$$

we see that selecting the column that gives the most negative $d\zeta/dx_j$ is the problem of minimizing over j the expression

$$k - \Sigma_i \, \overline{\Pi}_i \, a_{ij}.$$

Since there is a column for any cutting pattern, this is equivalent to

maximizing over nonnegative integer a_i the expression

$$\Sigma \bar{\Pi}_i a_i$$

subject to $$\Sigma a_i \ell_i \leq L.$$

So column choice is again a knapsack problem. The only difference from our earlier calculation is that the $\bar{\Pi}_i$ now involved are a compound of the prices of the two auxiliary objective functions and of their current values z_1 and z_2.

Checking the slack variables for a possible improvement is done using (5) directly and presents no difficulties.

IV. EXPERIMENTS

A. Effect of Stock Length. The effect of parent stock length on waste is a question of considerable practical importance since companies having cutting problems can sometimes control the parent length either through ordering rolls or by deciding upon having certain size paper machines.

We had originally conjectured that in a larger problem with a great variety of lengths demanded and a tremendous list of cutting patterns available, that almost any large stock size would yield about the same percentage waste. This, however, was not the case for the problem chosen for our experiment. This problem was run repeatedly for stock lengths ranging from 168 to 260 inches, with the results

shown in Figure 5. The percentage waste varied rapidly with stock length, showed no particular pattern, and was sensitive to changes as small as one inch in the parent stock size. In changing the parent stock size from 216 inches to 217 inches, the waste dropped 1/2%. In changing from 215 inches to 216 inches, the waste dropped 1%. In changing from 186 inches to 187 inches, the waste dropped almost 4%. Although there was a general downward tendency with increasing stock length, there are many peaks. The waste minimum attained at 217 inches was not matched again before the 255 inch stock size.

B. Use of Multiple Stock Lengths. With the demonstrated sensitivity to parent stock length indicating that it is probably difficult to pick a good standard size in advance of the orders being known, one way out would seem to be to use a number of different parent lengths in the calculation. The high waste series, problems A2-1 through A2-5, were rerun with 1, 2, 3, and then 4 parent lengths available. The results can be seen in Figure 6. In all cases, the waste was sharply reduced. In fact, we have no problems that give high waste on multiple parent stock lengths. Although human planners find the multiple stock problem much more difficult than the single stock, the machine computation time (Figure 6) was only very slightly increased

when compared with the single stock calculation.[1] This suggests that the use of multiple parent lengths, when it is possible to have them available, may be a very good way to cut down waste.

C. Cutting Knife Limitation. In order to see if waste was greatly affected by the number of cutting knives, the problems were rerun with an unrestricted number of knives (see Figure 7). This restricted number available for the data of Figure 2 were 7 for A1, 5 for A2, 5 for A3, 9 for A4 and 7 for A5. Although cutting patterns using more knives did show up in the solutions, the final waste figures were unchanged with the exception of A3-5 where there was a drastic reduction.

V. CONCLUSION

The methods described here have enabled us to speed up the process described in [1] to the point where it has become an effective procedure for real problems. Modifications in the formulation have also allowed us to take into account many special factors of the paper industry situation. Experiments have shown the usefulness of multiple parent stock widths, the sensitivity of problems to the exact

---

[1] The times recorded here for multiple stock lengths can probably be improved upon, since the 7094 program did not use the knapsack algorithm for multiple stock lengths described earlier, but rather the following: after M_1 is determined, the calculation of M_2 is begun with M_2 set initially to $(M_1 - c_1) + c_2$ in Step (2) rather than to c_2, and similarly with $M_3, \ldots, M_k$. The column giving the largest improvement is then chosen.

single parent length available, and the insensitivity in most cases to the usual sort of cutting knife limitation.

REFERENCES

[1] P. C. Gilmore and R. E. Gomory, "A Linear Programming Approach to the Cutting Stock Problem", Operations Research 9, 849-859 (1961).

[2] R. E. Gomory, "All-Integer Integer Programming Algorithm" appearing in "Industrial Scheduling", edited by John F. Muth and Gerald L. Thompson, Prentice-Hall (1963).

[3] R. E. Gomory, "An Algorithm for Integer Solutions to Linear Programs" appearing in "Recent Advances in Mathematical Programming", edited by Robert L. Graves and Philip Wolfe, McGraw-Hill (1963).

[4] George B. Dantzig, "Discrete Variable Extremum Problems", Operations Research 5, 161-310 (1957).

[5] George B. Dantzig, "Upper Bounds, Secondary Constraints and Block Triangularity in Linear Programming", Econometrica 23, 174-183 (1955).

[6] J. F. Benders, A. R. Catchpole and C. Kuiken, "Discrete Variable Optimization Problems", The RAND Symposium on Mathematical Programming, 65, March 1959.

[7] B. Martos, "Hiperbolikus Programozas", Publications of the Math. Institute of the Hungarian Academy of Sciences, V, 383-406 (1960).

[8] B. Martos, "Hyperbolic Programming by Simplex Method", Deuxième Congrès Mathématique Hongrois, Budapest, 24, August 31, 1960, Akadémiai Kiadó, Budapest, VI, 44-48 (1961).

[9] A. Charnes and W. W. Cooper, "Programming with Fractional Functionals: I. Linear Fractional Programming", ONR Research Memo No. 50, Northwestern University, February 1962.

[10] W. Dinkelbach, "Maximierung eines Quotienten zwei Linearen Funktionen unter Linearen Nebenbedingungen", Zeitschrift für Wahrscheinlichkeitstheorie 1, 141-145 (1962).

[11] W. S. Dorn, "Linear Fractional Programming", IBM Research Report RC-830, November 1962.

Data	No. of Ordered Lengths	% Waste	A Max. Knapsack Median Method Pivots	A Max. Knapsack Median Method Time (Min)	B No Max. Knapsack Median Method Pivots	B No Max. Knapsack Median Method Time (Min)	C No Max. On Knapsack Median Method Pivots	C No Max. On Knapsack Median Method Time (Min)	D No Max. On Knapsack Median Method Pivots	D No Max. On Knapsack Median Method Time (Min)
A1-1	40	3.0474	148	1.34	194	2.64	333	1.47	541	2.07
-2	35	0.0184	233	4.50	347	9.24	560	6.30	1104	11.03
-3	30	0.0616	161	6.32	171	6.75	418	7.19	586	7.84
-4	25	0.6227	101	0.47	135	0.93	177	0.62	545	1.51
-5	20	0.0539	90	1.32	111	2.23	214	2.25	370	3.08
A2-1	40	4.7232	110	0.52	167	0.62	122	0.50	316	1.06
-2	35	7.8219	90	0.31	76	0.28	93	0.30	126	0.42
-3	30	8.6921	87	0.25	71	0.22	87	0.24	111	0.25
-4	25	9.6407	47	0.14	73	0.15	64	0.14	76	0.17
-5	20	5.1748	21	0.09	19	0.06	28	0.07	28	0.08
A3-1	40	5.0845	57	0.33	73	0.32	66	0.33	80	0.52
-2	35	0.4825	141	3.22	157	2.34	348	3.35	450	3.83
-3	30	0.2114	83	0.59	124	0.78	371	1.97	518	1.69
-4	25	0.1484	90	0.54	95	0.58	259	0.81	380	0.92
-5	20	2.7297	24	0.12	27	0.07	173	0.30	476	0.93
A4-1	40	0.3869	138	5.53	210	41.58*	637	10.65*	727	15.87*
-2	35	0.4326	125	4.20	208	7.79	648	4.58*	878	17.75*
-3	30	0.4494	92	1.24	140	4.52	489	3.62	817	19.23*
-4	25	0.1415	98	2.51	141	6.30	415	3.66	878	9.19
-5	20	0.5490	67	1.74	77	1.57	536	5.13	786	18.10*
A5-1	50	0.4862	291	14.51	540	16.97*	980	12.51*	2123	19.87

Figure 2 - Data on runs to test methods of computation. * indicates a problem that was not run to completion. The percentage waste for these runs are given in the supplemental table.

Variation of Percentage Waste and Knapsack Size with Number of Pivots Executed

x = waste, ⊗ = size

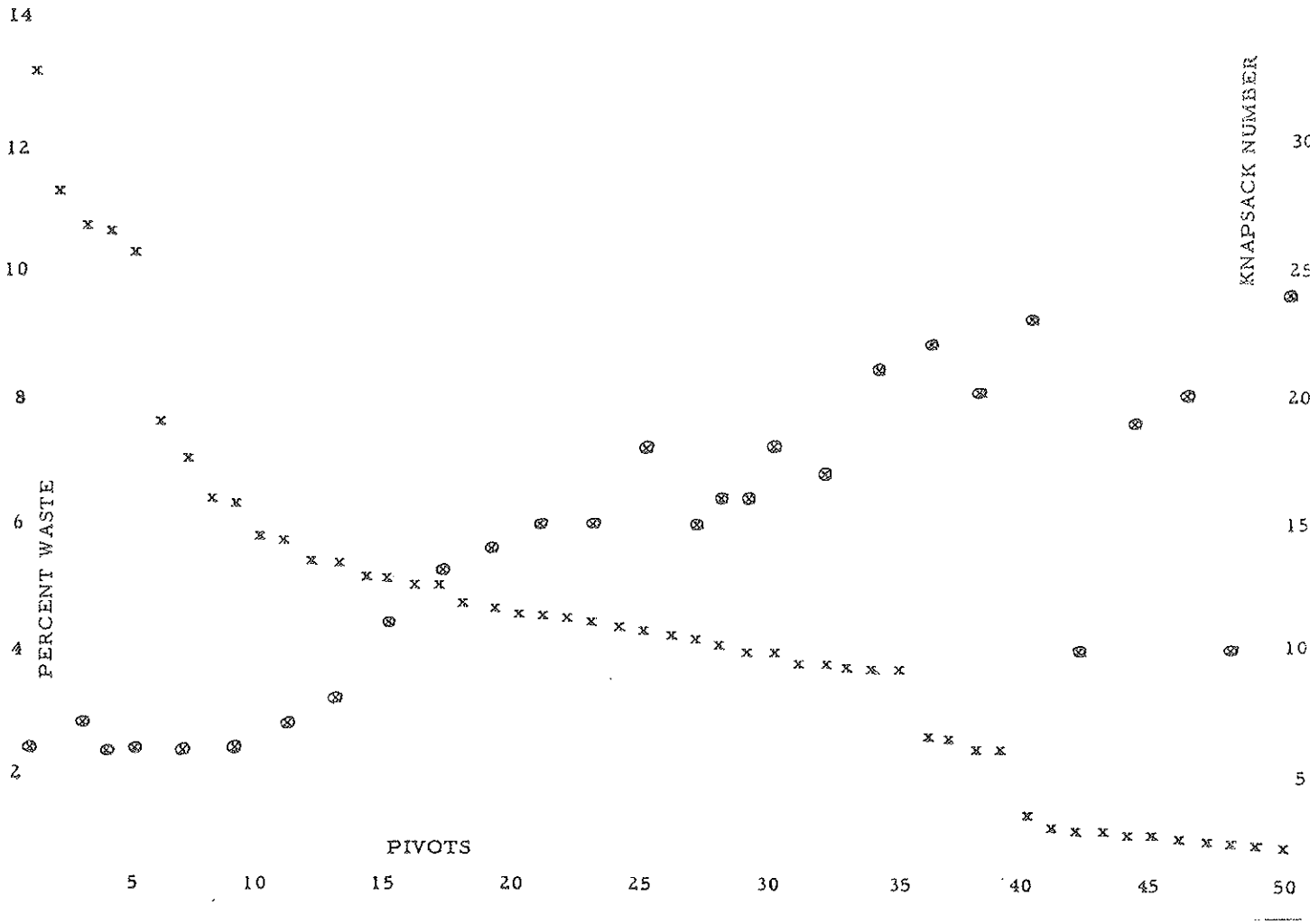


Figure 1 - Part 1

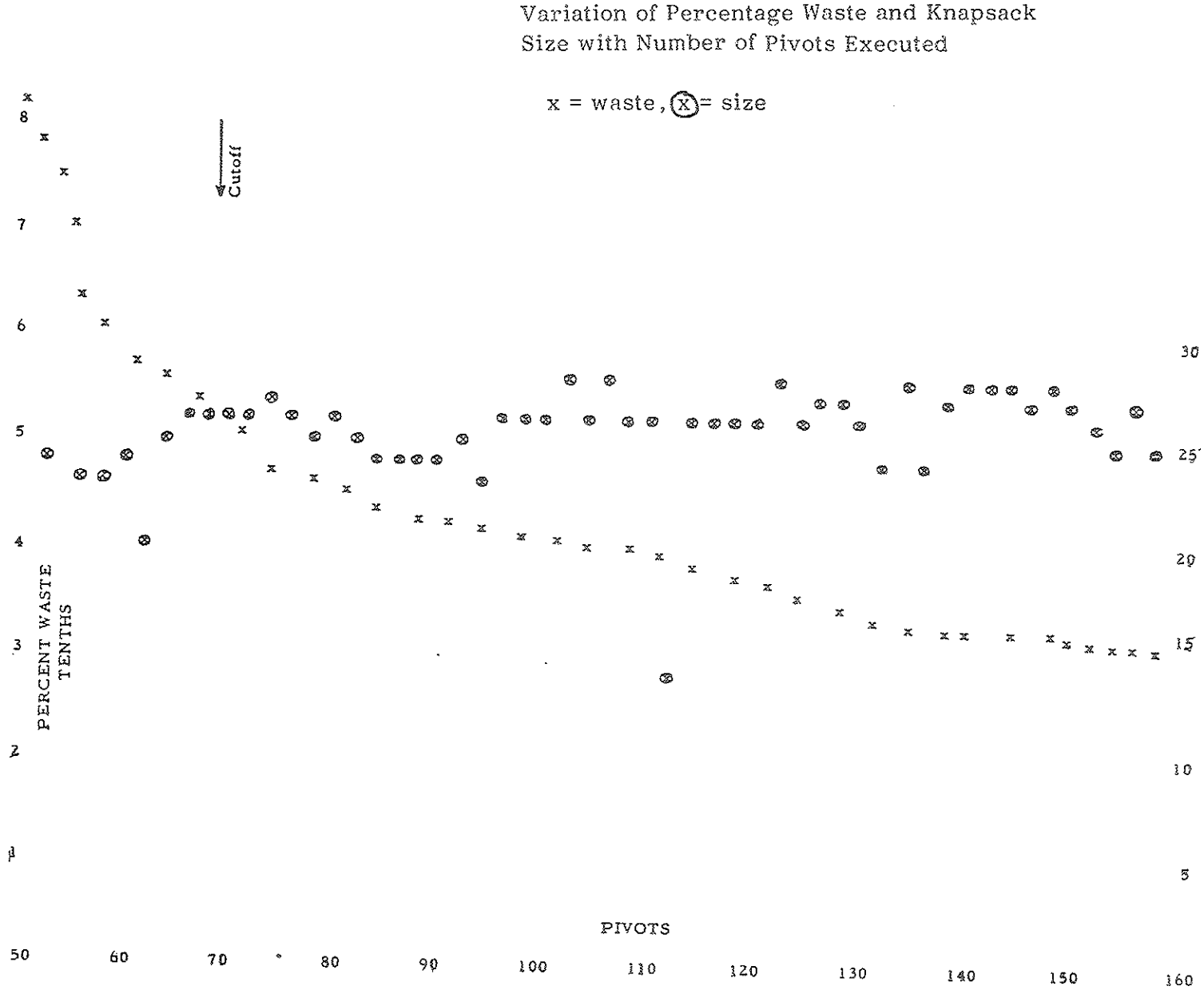


Figure 1 - Part 2

Supplemental Table of % Waste
for Incomplete Runs

Data	B	C	D
A4-1	0.3918	0.3921	0.9355
-2		0.4353	1.0887
-3			0.9747
-5			1.1868
A5-1	0.5459	0.4927	

Figure 2 - (Supplement)

Stock Length	Cost	Cutting Limit
218.00000	218.00000	20.00000

Order Length	Quantity
81.00000	4415.00000
70.00000	291.00000
68.25000	4765.00000
67.50000	4827.00000
66.75000	90.00000
66.00000	691.00000
64.00000	263.00000
63.75000	141.00000
63.00000	133.00000
60.00000	390.00000
56.25000	459.00000
56.00000	343.00000
52.50000	766.00000
52.00000	58.00000
51.75000	2736.00000
51.00000	212.00000
50.00000	720.00000
49.50000	133.00000
46.50000	529.00000
45.50000	185.00000
44.50000	94.00000
41.25000	393.00000
38.50000	47.00000
38.00000	95.00000
35.00000	411.00000
33.50000	36.00000
33.00000	273.00000
32.00000	56.00000
31.50000	171.00000
21.50000	140.00000

Figure 3 - Typical Test Problem
(Sheet 1)

No. of Times Cutting Pattern Used	No. of Lengths to be Cut	Ordered Lengths
2382.50000	1	81.00000
	2	68.25000
302.55552	2	81.00000
	1	56.00000
36.00000	1	81.00000
	2	51.75000
	1	33.50000
1194.38860	1	81.00000
	2	67.50000
94.99999	1	67.50000
	1	60.00000
	1	52.50000
	1	38.00000
141.00000	1	67.50000
	1	63.75000
	1	51.75000
	1	35.00000
86.40738	1	67.50000
	1	66.00000
	1	63.00000
	1	21.50000
77.48144	1	60.00000
	3	52.50000
26.40740	2	67.50000
	1	51.00000
	1	31.50000
529.00000	1	67.50000
	2	51.75000
	1	46.50000
133.00000	1	67.50000
	1	66.00000
	1	49.50000
	1	35.00000

Figure 3 - Typical Test Problem
(Sheet 2)

No. of Times Cutting Pattern Used	No. of Lengths to be Cut	Ordered Lengths
19.59259	1	63.00000
	2	51.75000
	1	51.00000
56.00000	2	67.50000
	1	51.00000
	1	32.00000
144.24070	2	56.25000
	2	52.50000
145.25922	1	67.50000
	2	51.75000
	1	45.50000
5.50000	1	67.50000
	2	52.00000
	1	45.50000
272.99999	2	67.50000
	1	50.00000
	1	33.00000
94.00000	1	70.00000
	2	51.75000
	1	44.50000
53.59262	3	51.75000
	1	41.25000
	1	21.50000
34.24078	1	67.50000
	2	52.50000
	1	45.50000
170.51850	1	60.00000
	1	56.25000
	1	51.75000
	1	50.00000

Figure 3 - Typical Test Problem
(Sheet 3)

No. of Times Cutting Pattern Used	No. of Lengths to be Cut	Ordered Lengths
169.70368	2	67.50000
	2	41.25000
109.99999	2	66.00000
	1	51.00000
	1	35.00000
27.00001	1	67.50000
	1	63.00000
	1	52.50000
	1	35.00000
90.00000	1	67.50000
	1	66.75000
	1	51.75000
	1	31.50000
13.48148	3	56.00000
	1	50.00000
263.00000	1	64.00000
	2	51.75000
	1	50.00000
54.59259	1	67.50000
	1	66.00000
	1	52.50000
	1	31.50000
197.00000	1	81.00000
	1	70.00000
	1	66.00000
47.00001	1	67.50000
	1	60.00000
	1	52.00000
	1	38.50000

Figure 3 - Typical Test Problem
(Sheet 4)

Data	Decrease in No. of Pivots	Increase in % Waste	Decrease in Time
A1-1	15	0.0470	0.25
2	155	0.3424	4.08
3	110	0.4340	6.00
4	43	0.3165	0.32
5	43	0.1493	1.02
A2-1	19	0.1673	0.18
2	3	0.0181	0.05
3	0	0	0
4	0	0	0
5	0	0	0
A3-1	3	0.0077	0.08
2	74	0.1723	2.92
3	22	0.1022	0.19
4	25	0.0961	0.17
5	0	0	0
A4-1	72	0.1877	2.12
2	83	0.4581	3.86
3	42	0.1276	0.53
4	32	0.0725	0.45
5	25	0.0622	0.70
A5-1	158	0.6356	13.30

Figure 4 - Cutoff Device

(Stop when improvement over 10 pivots is less than 1/10%.)

Figure 5 - Variation of Waste with Stock Length
(The problem used was that given in Figure 3.)

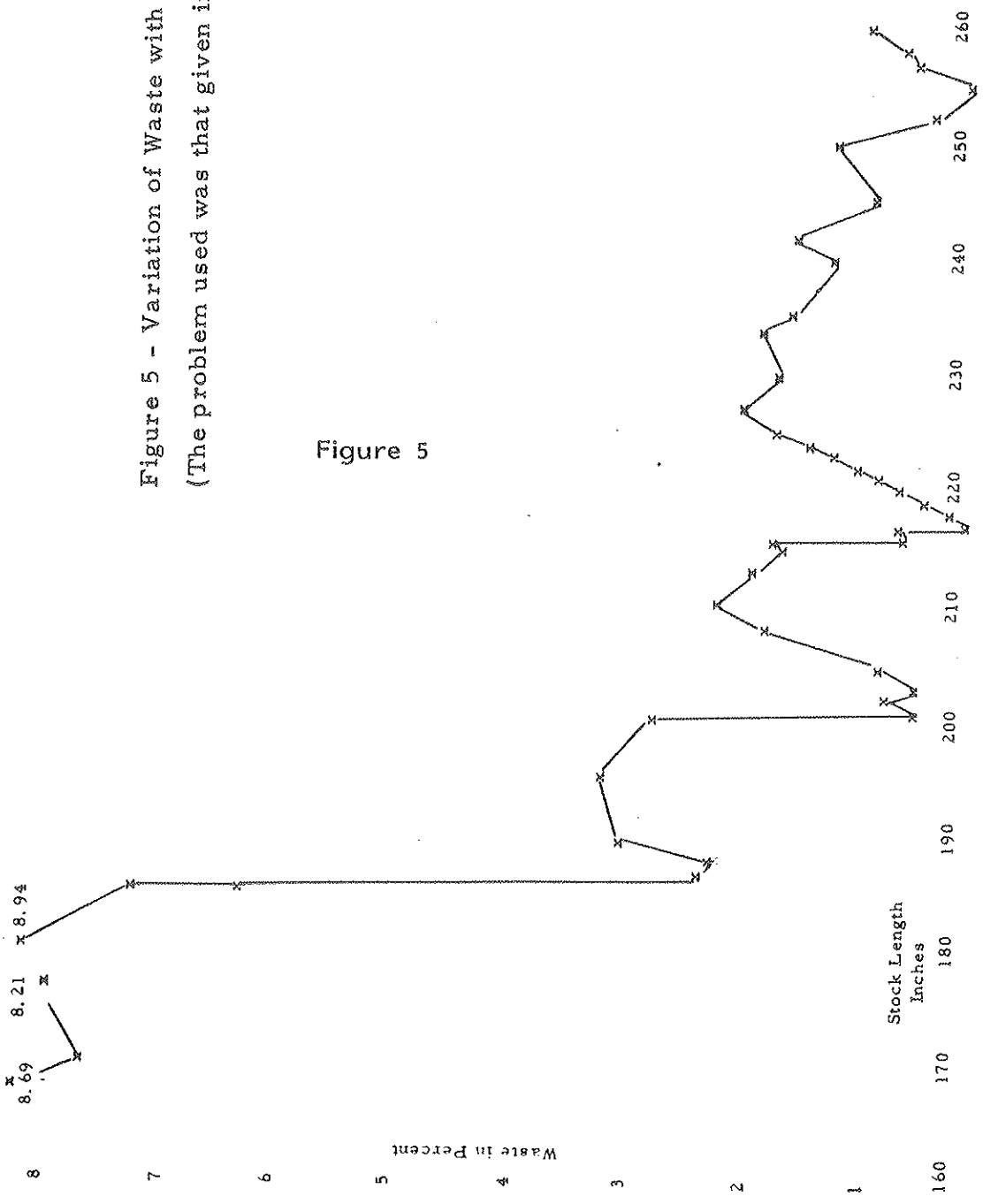


Figure 5

Data Used	No. of Ordered Lengths	Stock Lengths Used	Completed at Pivot No.	Time (Mins.)	% Waste
A2-1	40	168	110	0.52	4.7232
		168, 145	130	0.68	1.5655
		168, 145, 124	132	0.76	1.0650
		168, 145, 140, 124	118	0.75	0.7857
A2-2	35	168	90	0.31	7.8219
		168, 145	93	0.40	3.6826
		168, 145, 124	88	0.45	3.0969
		168, 145, 140, 124	91	0.52	2.5841
A2-3	30	168	87	0.24	8.6921
		168, 145	94	0.37	3.1580
		168, 145, 124	85	0.40	2.7676
		168, 145, 140, 124	80	0.44	1.8374
A2-4	25	168	47	0.14	9.6407
		168, 145	60	0.19	2.8297
		168, 145, 124	59	0.22	2.5589
		168, 145, 140, 124	67	0.29	1.2289
A2-5	20	168	21	0.09	5.1748
		168, 145	40	0.15	1.2257
		168, 145, 124	38	0.22	1.2257
		168, 145, 140, 124	42	0.19	0.9130

Figure 6 - Variation of Waste with Number of Stock Lengths

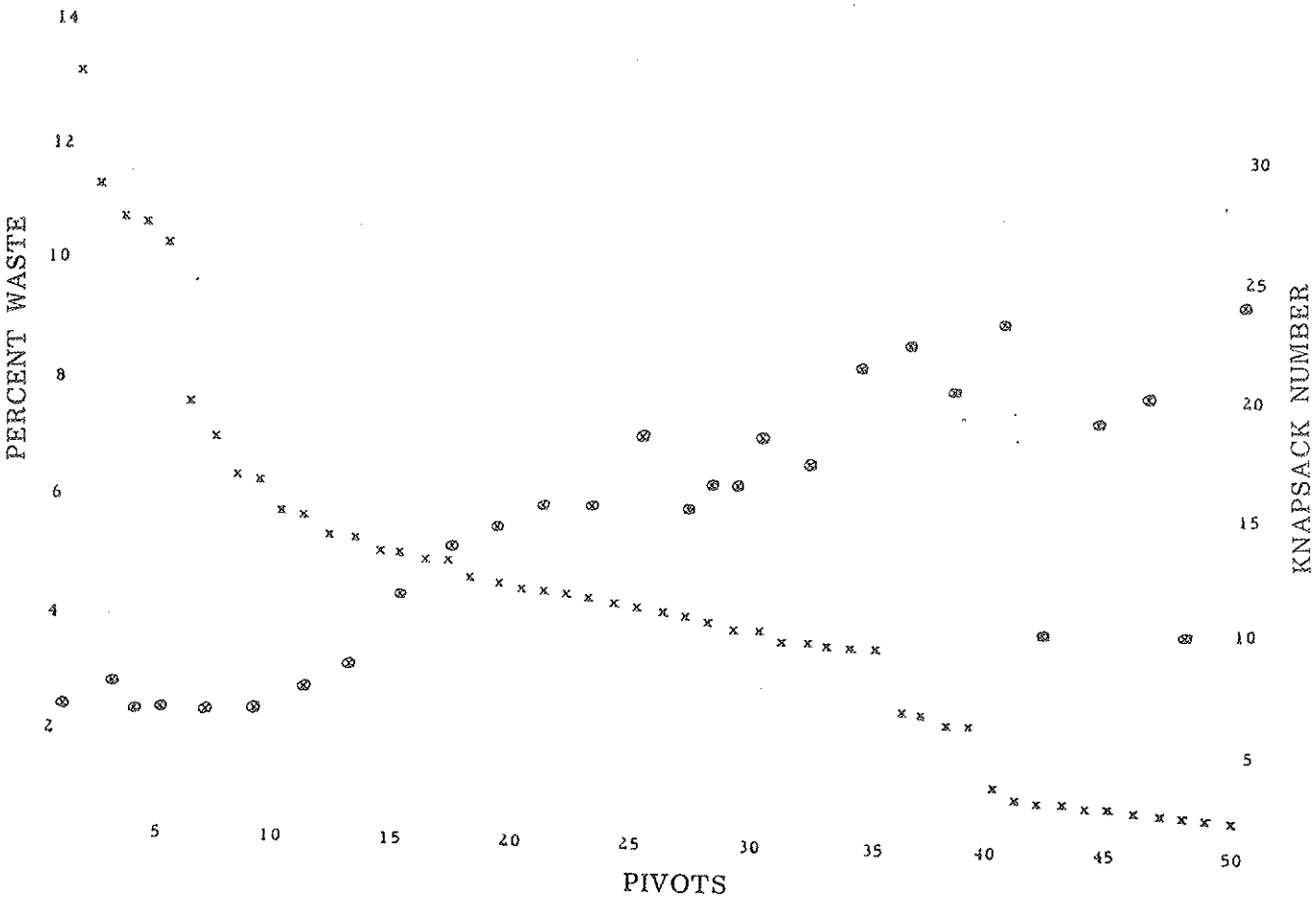
Data	No.	%	Time
A1-1	160	3.0474	3.32
-2	213	0.0184	5.03
-3	184	0.0616	6.88
-4	75	0.6227	0.30
-5	80	0.0539	1.50
A2-1	110	4.7232	0.44
-2	84	7.8219	0.26
-3	83	8.6921	0.22
-4	47	9.6407	0.10
-5	21	5.1748	0.04
A3-1	36	5.0845	0.20
-2	108	0.4825	2.48
-3	74	0.2114	0.35
-4	62	0.1484	0.17
-5	46	0.0865	0.21
A4-1	134	0.3869	3.98
-2	122	0.4326	1.43
-3	71	0.4494	0.54
-4	114	0.1415	0.89
-5	79	0.5490	0.40
A5-1	297	0.4862	9.62

Figure 7 - Waste Without Cutting Knife Limitation

Data	No. of Ordered Lengths	% Waste	A Max. Knapsack Median Method Pivots	A Max. Knapsack Median Method Time (Min)	B Max. Knapsack No Median Method Pivots	B Max. Knapsack No Median Method Time (Min)	C No Max. On Knapsack Median Method Pivots	C No Max. On Knapsack Median Method Time (Min)	D No Max. On Knapsack No Median Metho Pivots	D No Max. On Knapsack No Median Metho Time (Min)
A1-1	40	3.0474	148	1.34	194	2.64	333	1.47	541	2.07
-2	35	0.0184	233	4.50	347	9.24	560	6.30	1104	11.03
-3	30	0.0616	161	6.32	171	6.75	418	7.19	586	7.84
-4	25	0.6227	101	0.47	135	0.93	177	0.62	545	1.51
-5	20	0.0539	90	1.32	111	2.23	214	2.25	370	3.08
A2-1	40	4.7232	110	0.52	167	0.62	122	0.50	316	1.06
-2	35	7.8219	90	0.31	76	0.28	93	0.30	126	0.42
-3	30	8.6921	87	0.25	71	0.22	87	0.24	111	0.25
-4	25	9.6407	47	0.14	73	0.15	64	0.14	76	0.17
-5	20	5.1748	21	0.09	19	0.06	28	0.07	28	0.08
A3-1	40	5.0845	57	0.33	73	0.32	66	0.33	80	0.52
-2	35	0.4825	141	3.22	157	2.34	348	3.35	450	3.83
-3	30	0.2114	83	0.59	124	0.78	371	1.97	518	1.69
-4	25	0.1484	90	0.54	95	0.58	259	0.81	380	0.92
-5	20	2.7297	24	0.12	27	0.07	173	0.30	476	0.93
A4-1	40	0.3869	138	5.53	210	41.58*	637	10.65*	727	15.87
-2	35	0.4326	125	4.20	208	7.79	648	4.58*	878	17.75
-3	30	0.4494	92	1.24	140	4.52	489	3.62	817	19.23
-4	25	0.1415	98	2.51	141	6.30	415	3.66	878	9.19
-5	20	0.5490	67	1.74	77	1.57	536	5.13	786	18.10
A5-1	50	0.4862	291	14.51	540	16.97*	980	12.51*	2123	19.87

VARIATION OF PERCENTAGE WASTE
AND KNAPSACK SIZE WITH
NUMBER OF PIVOTS EXECUTED

x = waste, ⊗ = size



VARIATION OF PERCENTAGE WASTE
AND KNAPSACK SIZE WITH
NUMBER OF PIVOTS EXECUTED

x = waste, ⊗ = size