IBM

Application Program

1620 One-Dimensional Trim Program (1620-MT-01X)

Program Reference Manual

This program solves the one-dimensional trim problem encountered in the manufacture of paper and other industries.

This manual contains (1) a description of the application, including the methods used, the input and output formats, and sample problems, and (2) operating instructions for the program.

Copies of this and other IBM publications can be obtained through IBM branch offices. Address comments concerning the contents of this publication to IBM, Technical Publications Department, 112 East Post Road, White Plains, N. Y. 10601

CONTENTS

Program Abstract 1
General Description 1
Purpose 1
Advantages of the Program 2
Methods 2
Timing 10
Restrictions and Range 11
Precision 11
Program Systems Chart 12
Description of Systems Approach 13
Input Format 15
General Instructions 15
Identification Card 15
Stock Cards 15
Order Cards 16
Blank Card 17
Data Format Summary 17
Identification Card 17
Stock Cards 17
Order Cards 17
Output Description 18
Sample Problems 20
Program Modifications 29
1443 Printer Output 29
Rounded Solution 30
Maximum Number of Stock and Order Widths 31
Field Length 31
Operating Instructions 31
Program Section 1 31
Program Section 2 32
Program Section 3 33
Data Deck Order 34
Program Setup Sheet 35
Halt List 37
Storage Map 38
Bibliography 39

PROGRAM ABSTRACT

This program provides a linear programming solution to the one-dimensional trim problem. The one-dimensional trim problem can be described briefly as follows: We have a supply of material (either stocked or which we can produce) and a set of orders to be filled using this material. The material is stocked (or produced) in one or more fixed widths, each having a fixed cost associated with it. The orders each specify a width and the number of units of that width required, and they are filled by cutting up units of the stocked widths into the ordered widths. The cheapest way of cutting up the supply to fill the orders must be determined.

The program will solve any one-dimensional trim problem having up to 16 stock sizes and 35 order sizes. It is adapted particularly to the problem of cutting paper stock and allows the user to specify certain restrictions characteristic of the paper industry.

The program is coded in SPS, Symbolic Programming System for the 1620, and requires a 1620 with 20,000 characters of core storage, card input/output, automatic divide, indirect addressing, and additional instructions (TNS, TNF, MF).

GENERAL DESCRIPTION

Purpose

The One-Dimensional Trim Program provides a minimum-cost solution to the trim problem, which is to fill a set of orders for a material that varies in size in one-dimension (width) from stock whose size is fixed in that dimension. The orders each specify a width and the number of units of that width which are desired, and they are filled by cutting up units of stock into the ordered widths. The stock may be available in only one or in several widths, each width having a certain cost per unit associated with it. Given a list of the orders and a description of the stock available, the problem then is to determine how to cut up the units of stock into the ordered widths. The output of the trim program provides this information in the form of cutting patterns. Each pattern indicates the width of stock to be used, the manner in which each unit of this width should be cut into ordered widths, and the number of units of this stock width to be cut in this way.

The program will solve any one-dimensional trim problem which falls within the size limitations of the program. (These are described in the "Restrictions and Range" section of this manual.) The program is adapted particularly to the problem of cutting paper stock and allows the user to specify certain restrictions characteristic of the paper industry. In this application the order unit is a roll of paper, the stock widths are the widths of the machines that produce the paper, and the cost per unit of stock is simply the cost of manufacturing one roll of paper on the machine which produces it. As the paper is manufactured, knives on the machine slit the machine-width roll into rolls of the ordered sizes. Each pattern from the program output tells which machine size to use, where

to set the knives, and how much paper to produce with that setting.

The user is permitted to limit the number of pieces into which a machine-width roll may be slit, in order to prevent the use of patterns requiring more knives than are available on the machine. He may also limit the number of rolls to be produced on each machine, to avoid overuse of some machines. This last program feature provides the ability to limit the supply available in each stock size.

Advantages of the Program

The chief advantage of the One-Dimensional Trim Program is that it provides a least-cost solution. Obtaining even a reasonably good set of patterns by hand, as it has traditionally been done, is a difficult and time-consuming task, and there is no way of judging whether the solution so obtained is nearly optimal or far from it. Scheduling an order on more than one stock width by hand is even more difficult and is rarely attempted. Herein lies a second advantage of the trim program: it can handle a number of different stock sizes at once, each with a different cost and supply. Since total cost rather than trim loss is minimized, the cost associated with a stock width may be used to reflect the efficiency of the machine producing the width. The supply limits prevent overuse of any machine or use of inventory beyond available supply. Trim loss may be minimized as a special case by assigning to each stock width a cost proportional to the width. Finally, large problems, such as long-range planning problems, can now be solved. For example, the program can assist in determining a good stock width for a class of orders, a problem encountered in paper manufacture when a company orders a new paper machine.

Methods

The one-dimensional trim problem is a problem in linear programming and may be described in the following way. If the stock material is available in widths W_1, $W_2, \ldots,$ W_ℓ, and the orders together require q_1 units of width w_1, q_2 units of width $w_2, \ldots,$ and q_m units of width w_m, then the problem becomes:

$$\text{Minimize} \quad \sum_j C_{p_j} x_j$$

$$\text{Subject to} \quad \sum_j a_{ij} x_j = q_i \qquad i = 1, \ldots, m$$

where x_j is the number of units of stock to be cut up according to the jth cutting pattern, p_j is the index of the stock from which the jth pattern is cut, C_{p_j} is the cost of one unit of the stock width W_{p_j}, and a_{ij} is the number of units of width w_i produced each time one unit of stock is cut according to the jth pattern.

The solution to the problem is a list. Each entry in this list consists of a cutting pattern, the number of units of stock which are to be cut into the pattern, and which stock is to be used. In the notation above, the jth such pattern is a vector $(a_{1j}, a_{2j}, \ldots, a_{mj})$, indicating that the jth pattern consists of a_{1j} pieces of width w_1, a_{2j} pieces of width $w_2, \ldots,$ and a_{mj} pieces of width w_m; x_j units of the stock width W_{p_j} are to be cut in this fashion.

In some one-dimensional trim applications, some or all of the stock widths may be available only in limited supply. To express this restriction, for each stock width W_p for which the inventory is insufficient to cut the entire order, the following constraint is added to the formulation of the problem:

$$\sum_{j \in p} x_j \leq S_p$$

where S_p is the number of units of width W_p available and $j \in p$ means that the jth pattern was cut from stock width W_p.

The trouble with solving the problem with the techniques ordinarily applied to linear programming problems is that the number of variables (that is, possible cutting patterns) may be prohibitively great. This results in a matrix for the simplex method which has a very large number of columns. Apart from the physical difficulty of storing such a large matrix in a computer memory, there is also the difficulty of searching through all the columns at each pivot step, which requires a great deal of time.

One technique for overcoming this problem is to generate and store some reasonable number of good cutting patterns and obtain a standard linear programming solution using only these variables. A "good" cutting pattern is one having little trim waste (the part of the stock width left after the ordered widths have been cut from it), and the number of patterns which would be reasonable would depend on the speed and storage facilities of the computer on which the problem was solved. The resulting linear programming problem is still very large, and there is no way of knowing whether the solution is optimal or not, since only a fraction of the possible cutting patterns have been permitted in the solution. This approach is called the "library" method because of the use of a library of patterns.

Another approach to the problem is to abandon linear programming entirely and treat the problem by heuristic methods. Very simply stated, one such method is to generate a good pattern and use it as much as possible, that is, until the demand for one width is filled entirely by the use of that pattern. Then a good pattern using the remaining widths is generated and used as much as possible, until another demand is filled. This procedure continues until all demands are filled. This idea has

been exploited in a program now in use in some paper mills. From the nature of the method it is clearly impossible to guess how close to optimal this solution is. Nonetheless, it has one advantage over linear programming: the amount of each pattern to be cut can be an integer. This is generally not true of the standard linear programming solution, and this results in the rounding problem, which is explained below.

In some applications, it does not make sense to cut a fractional number of units of stock into a pattern. Consequently, the nonintegral amounts of the patterns to be cut in the linear programming solution may have to be rounded up or down to the next whole number to produce an integer solution. The quantities of the various ordered widths which are produced by this integer solution may differ slightly from the quantities which were ordered; there is no way to guarantee a solution which, rounded, will produce exactly the right quantities of the ordered amounts. However, the integer solution will be an optimal solution to the slightly revised problem if the unrounded solution was an optimal solution to the original problem.

This program treats the problem by linear programming (and hence inherits the rounding problem), but uses a variation of the usual simplex method which overcomes the storage and search difficulties mentioned above. The technique is described in detail in reference 1 (see Bibliography). Briefly, however, this approach is made possible by recognizing that the problem of finding an improving column may be treated as an auxiliary problem, and indeed is a simple integer programming problem called the knapsack problem. Hence all possible patterns are implicitly considered, making the solution optimal, but only the cutting patterns in the basis are recorded explicitly at any one time so that the storage requirement is that of a small linear programming problem. Specifically, there are $m + q$ basis matrix columns, where m is the number of ordered widths, and q is the number of stock widths with limited supply. Since only these columns are stored when the "pricing out" stage in the simplex algorithm is reached, the auxiliary knapsack problem must be solved; that is, all of the nonbasic columns must be generated and evaluated using the current set of linear programming prices, and the best must be chosen as pivot column. Once the pivot column has been determined, it is updated by multiplying it by the inverse of the basis matrix. The remainder of the simplex method, namely, choosing a pivot row and the Gaussian elimination step, is unchanged. If no column can be created which will improve the current solution, this solution is optimal just as in the usual simplex method.

Since the primal simplex method requires a feasible starting solution, or a separate phase to determine one, the following obvious solution is used as a starting point. There are m patterns in the starting solution; the jth of these consists of as many pieces n_j of width w_j as can be cut from the first stock width for which the supply is sufficient to fill the whole demand q_j for that width by use of the one pattern. The amount required is q_j / n_j units. If there is no stock with a supply of this size remaining, the pattern is cut from an imaginary stock. This imaginary stock is available in unlimited supply and is as long as the last stock,

but has a very great cost per unit. Patterns cut from this imaginary stock are gradually dropped from the solution by the linear programming process, provided the total supply of real stock widths is sufficient to produce the order.

A number of methods for solving the auxiliary knapsack problem have been developed; they are discussed in references 1 and 2 (see Bibliography). The scheme used in the One-Dimensional Trim Program is described in the section which follows. Creating the best new variable at any step in the linear programming process is equivalent to the following problem:

$$\text{(1)}\quad \text{Maximize} \quad \sum_{i=1}^{m} \pi_i a_i - C_p \qquad a_i \geq 0 \text{ and integral}$$

$$\text{(2)}\quad \text{Subject to} \quad \sum_{i=1}^{m} w_i a_i \leq W_p \qquad \text{for some } p,\ 1 \leq p \leq \ell$$

The column which maximizes (1) subject to (2) is a pattern consisting of a_1 pieces of width w_1, a_2 of width w_2, and so on, cut from stock width W_p of cost C_p. The current linear programming price for the width w_i is π_i, $i = 1, \ldots, m$.

Strictly speaking, any column which satisfied (2) and for which (1) was positive could be used as pivot column to improve the current linear programming solution. However, experiments indicate that if just any improving column is chosen as pivot column, a very large number of linear programming steps will be required to reach an optimum solution*, whereas if the best column is used — "best" in the sense of maximizing (1) — the number of steps is greatly reduced.

Since the time spent on solving the knapsack problem is the major part of the time required for the whole problem, and since the time depends partly on the number of variables in the knapsack (number of different ordered widths to be combined into a pattern), it is desirable to have this number of variables as small as possible. To this end, on every other iteration of the linear programming problem an attempt is made to form a pattern from only half of the order widths, namely, those for which the order quantity is above the median order quantity. This device not only reduces the number of variables in the knapsack on alternate iterations but has another even more important effect. Since a pattern which contains a width with a small demand will generally be cut a small number of times (at most, enough to satisfy the demand for that width completely with the one pattern), the entry of such a column into the linear programming problem cannot very greatly alter the objective

*See (2) for a discussion of this phenomenon.

function, which is the cost of cutting the whole order. On the other hand, a pattern containing only widths with large ordered quantities may be cut a large number of times and hence affect the cost very strongly. If at any point no pattern so formed will improve the current linear programming solution, the program attempts to form one using all the widths.

The number of widths in the knapsack is further reduced in the following way. Of the widths being considered for the knapsack, if there is any width w_i whose price is not greater than those of all shorter widths in the knapsack, then it can be omitted, for any pattern using w_i will have the same or greater value if the offending shorter width is substituted for w_i, and the pattern will fit into the same stock width.

After these preliminaries, the next step is to reorder the widths in descending order of the density of price per unit width; that is:

$$\frac{\pi_1}{w_1} \geq \frac{\pi_2}{w_2} \geq \cdots \geq \frac{\pi_{\bar{m}}}{w_{\bar{m}}}$$

where $\bar{m}$ is the (reduced) number of widths in the knapsack.

Once the variables in the knapsack have been reduced in number and reordered by density, the following algorithm is used. It is applied separately for each stock width W_p.

Step 1: In the algorithm the possible vectors $(a_1, a_2, \ldots, a_{\bar{m}})$ which satisfy

$$\sum_{i=1}^{\bar{m}} a_i w_i \leq W_p \tag{3}$$

are generated in lexicographically decreasing order*. A test in Step 4 permits many of the possible vectors to be skipped. The process is started by generating the lexicographically greatest vector which satisfies condition (3).

* A vector $(a_1, a_2, \ldots, a_{\bar{m}})$ is lexicographically greater than another vector $(b_1, b_2, \ldots, b_{\bar{m}})$ if the first nonzero element of their term-by-term difference $(a_1-b_1, a_2-b_2, \ldots, a_{\bar{m}}-b_{\bar{m}})$ is positive.

$$a_1 = \left[\frac{W_p}{w_1}\right], \qquad a_2 = \left[\frac{W_p - a_1 w_1}{w_2}\right], \ldots,$$

$$a_{\bar{m}} = \left[\frac{W_p - \sum_{i=1}^{\bar{m}-1} a_i w_i}{w_{\bar{m}}}\right]^{**}$$

Step 2: If any vector has already been generated which would improve the current linear programming solution, the current solution vector (a) is tested to determine whether its value exceeds the cost of the stock on which the pattern is to be cut by more than the best previous pattern exceeded the cost of its stock. If the best previous pattern is the vector (b), cut on stock W_q with cost C_q, then the amount by which the pattern value exceeded the cost is:

$$\delta = \sum_{i=1}^{\bar{m}} b_i \pi_i - C_q$$

Then if

$$\sum_{i=1}^{\bar{m}} a_i \pi_i \geq C_p + \delta \tag{4}$$

the current solution (a) replaces (b) as the best solution, and δ is redefined as

$$\delta = \sum_{i=1}^{\bar{m}} a_i \pi_i - C_p.$$

If no vector has yet been generated which would improve the solution, δ is defined as zero, and test (4) is applied as before.

** $[x]$ means the greatest integer contained in x.

Step 3: k is defined as the highest index such that

$$a_i = 0 \qquad \text{for all } i = k+1, \ldots, \bar{m}-1$$

If k does not exist, that is, if

$$a_i = 0 \qquad \text{for all } i = 1, \ldots, \bar{m}-1$$

then all the possible vectors have been considered and the current best solution (b) is the best pattern for the current linear programming prices. If (b) does not exist for any stock width W_p, that is, no vector (b) satisfies

$$\sum_{i=1}^{\bar{m}} b_i \pi_i > C_p$$

$$\sum_{i=1}^{\bar{m}} b_i w_i \leq W_p$$

then an optimal solution has been reached for the linear programming problem.

Step 4: A test is performed to determine whether any pattern with the first k elements $a_1, a_2, \ldots, a_k$ could have a value greater than that of the current best variable. This is accomplished as follows: the amount of stock unused by the first k widths is filled out with the $(k+1)^{st}$ width with the integer restriction dropped. Since the $(k+1)^{st}$ width has the greatest price density (when the first k widths are excluded), clearly the pattern could have no greater value than that obtained if it could be filled out perfectly with this width. In other words, if

$$\sum_{i=1}^{k} a_i \pi_i + \left(\left(W_p - \sum_{i=1}^{k} a_i w_i \right) / w_{k+1} \right) \cdot \pi_{k+1} \leq C_p + \delta \tag{5}$$

no pattern $(a_1, a_2, \ldots, a_{\bar{m}})$ whose first k elements are $a_1, a_2, \ldots, a_k$ will have greater value than the current best column (b), and all such patterns may be ignored. Indeed, no downward adjustment in a_k, which would be the next step in lexicographical order, will help either, since this would increase the amount of stock to be filled out with the $(k+1)^{st}$ width, known to have poorer price density than the kth width. Hence the present a_k can be set to zero, and returning to the beginning of Step 4, a new smaller k determined and test (5) repeated.

Step 5: If, however, the cutoff test fails, that is:

$$\sum_{i=1}^{k} a_i \pi_i + \frac{\left(W_p - \sum_{i=1}^{k} a_i w_i \right)}{w_{k+1}} \cdot \pi_{k+1} > C_p + \delta$$

then patterns with first elements $a_1, \ldots, a_k$ must be considered. Hence the next pattern in lexicographical order is generated.

$$a'_i = a_i \qquad i = 1, \ldots, k-1$$

$$a'_k = a_k - 1$$

$$a'_i = \left[\frac{W_p - \sum_{j=1}^{i-1} a'_j w_j}{w_i} \right] \qquad i = k+1, \ldots, \bar{m}$$

The new pattern (a') becomes the current pattern, denoted (a), and the procedure is repeated by returning to Step 2.

One restriction enters into the generation of possible patterns from the knapsack. When the program is used for paper trim, patterns with more pieces in them than there are slitters on the machine must be excluded from the solution. Since one slitter is required for each piece in the pattern, the user may specify for each machine a maximum number of slitters available, and the knapsack routine will exclude any pattern violating this restriction. This maximum is taken into account in steps (1) and (5) as the patterns are formed. If at most K_p pieces may be cut from stock width W_p, then

$$a_i = \min \left(K_p - \sum_{j=1}^{i-1} a_j, \left[\left(W_p - \sum_{j=1}^{i-1} a_j w_j \right) / w_i \right] \right)$$

rather than simply

$$a_i = \left[\left(W_p - \sum_{j=1}^{i-1} a_j w_j \right) / w_i \right]$$

Timing

The amount of time required to solve a particular trim problem is very difficult to predict. The time increases rapidly with the number of order widths and, to a lesser extent, with the number of stock widths. It decreases with the percentage of trim waste in the optimal solution. That is, a problem for which the optimal solution contains very high waste may take only a fraction of the time required for a problem with the same number of stock and order widths which has very low inherent waste. Since the waste is not known until the solution is determined, it is impossible to consider this factor in estimating how long a particular problem may run.

In the low waste problems, generally more linear programming steps are required than in other problems. Much more important, however, is the fact that in any problem the time per step increases markedly as the solution nears the optimum. In fact, in nearly all problems for which the solution time is long, a large percentage of the total processing time may be spent in proceeding from a very good solution to an optimum one. Hence it is often practical to interrupt the program after the trim loss has dropped to an acceptable level or has become nearly constant, and use the current solution instead of the optimum. The program contains provisions for printing the trim when the user wishes to see it and also for interrupting the solution (see "Operating Instructions").

Below is a table of computing times for some typical problems which were run on a Model II 1620. The time figures do not include the time required for printing the solution, which varies from 5 to 10 minutes, depending on the amount of typing.

Order widths	Stock widths	Percent trim	Iterations	Minutes
4	1	2.50	5	1
5	1	1.45	10	2
7	3	.00	10	2
10	2	.17	29	4
14	1	2.37	22	4
16	1	.12	28	4
16	1	.12	48	5
17	3	3.49	33	3
18	1	.56	58	10
20	1	.05	82	27
20	1	5.17	21	3
20	1	2.73	25	4
20	1	.55	74	14
20	1	.05	82	24
20	15	1.27	108	43
20	16	.61	46	14
21	1	2.01	37	5
22	1	1.60	54	6
22	3	.56	90	15
25	1	.62	88	15

Order widths	Stock widths	Percent trim	Iterations	Minutes
25	1	9.64	51	6
25	1	.15	71	12
25	1	3.67	45	6
30	1	8.69	76	12
30	1	.21	93	20
30	1	.45	95	25
33	1	.00	77	20

Restrictions and Range

Because of the limitations of core storage, there may be at most 16 stock widths and 35 distinct order widths. In addition, if there are stock widths with limited supply, the number of these plus the number of order widths must not exceed 35. There must be at least two distinct order widths specified. There may be as many as 125 order cards.

The widths and costs supplied as input to the program may be anywhere in the range from .1 to 999.99999. The best range for maximum accuracy is in the upper part of this range, say above 10. The order quantities may be any integer between 1 and 99,999,999, and the supply quantities may be as large as 999,999,999. If the supply of a stock is unlimited, at least for the purposes of a particular problem, it should, for increased accuracy, be marked as such rather than given a very large supply.

Precision

The calculations in the One-Dimensional Trim Program are carried out in fixed-point arithmetic. Most of the numbers are ten digits long, stored in the format

XX. XXXXXXXX

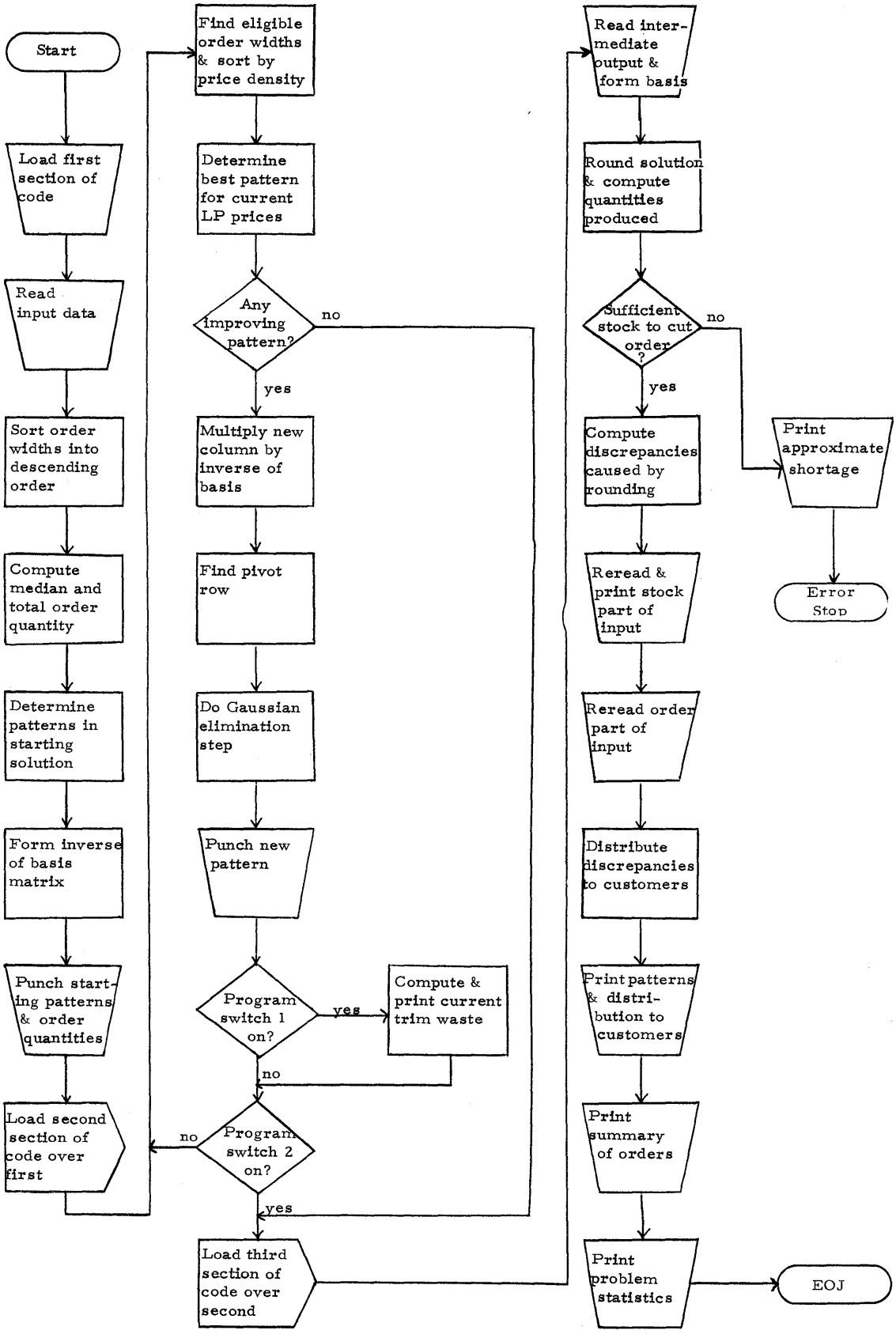
although the widths and costs are stored

XXXXX. XXXXX

and numbers which are integers are stored with the decimal point to the right of the rightmost digit. Integers whose values will never exceed 99 are stored as two-digit rather than ten-digit numbers.

The program results are accurate from five to eight significant figures.

PROGRAM SYSTEMS CHART



DESCRIPTION OF SYSTEMS APPROACH

The One-Dimensional Trim Program is divided into three sections of code which overlie one another to conserve storage. The first section sets up the matrix of the linear programming problem. After being loaded into core, this part of the code first reads the entire data deck, converting and storing the information required for determining the cutting patterns by linear programming. All of the stock information is used. Only the different widths and the total quantity desired of each are recorded from the order cards. The order widths are then sorted into descending order. The median order quantity is determined and the total quantity of material ordered is computed. Next the starting solution is generated, in the manner described in the "Methods" section of this manual. The basis matrix, which is determined by the starting patterns and stock supply constraints but which is not stored as such, is inverted and the inverse is stored. Finally, the starting patterns and the order quantities are punched for intermediate storage.

Having set up the linear programming problem, the first section of code is overlaid by the second, which finds an optimum solution. As explained in "Methods", a modification of the simplex method is used, and each iteration of this procedure begins with the solution of an auxiliary problem — that of generating the best pattern for the current set of linear programming prices. This pattern corresponds to one of the columns in the nonbasis, which are not stored explicitly. The column generation procedure consists of eliminating some order widths from consideration, sorting the remaining ones in descending order of the density of price per unit width, and finally applying the algorithm described under "Methods". If no column can be created which will improve the current linear programming solution, cutting patterns which will minimize the cost of filling the order have been determined, and the final section of code is brought in to print the program output.

Otherwise, the new column formed above is updated to the current solution by multiplying it by the inverse of the basis matrix and becomes the pivot column for the ordinary simplex method. The pivot row is determined in the usual way, and a Gaussian elimination step is performed. The new column is punched for intermediate storage.

If sense switch 1 is on at the conclusion of the pivot step, the percentage of trim waste in the current solution is computed and printed so that the user may observe the progress of the solution. Next sense switch 2 is interrogated; if it is off, the program returns to the beginning of this section of the code to generate a column for the new linear programming prices and to perform another Gaussian elimination. This cycle continues until the solution is optimal (no improving column can be found), or until sense switch 2 is turned on, indicating that the user wishes to terminate the linear programming and use the current solution.

At this point the third and final section of code which prints the output, is loaded, overlying the second. The first step in this section is to read the intermediate output, forming a list of order quantities and a

matrix of the patterns in the final solution from the information read. After the intermediate output is read, the quantity of each pattern to be cut, not generally an integer, is rounded to the nearest integer and the effect of this rounding on the amount produced of each ordered width is calculated. At this point the solution is tested to determine whether there is sufficient stock to cut the order. If not, the approximate shortage is indicated and the program stops. Otherwise, the data cards are reread. First, the identification card is read and printed. Then the stock cards are read and printed, along with the amount of each stock used in the rounded solution. Finally, the order cards are reread, and for each ordered width the discrepancy between the total quantity ordered and the quantity produced in the rounded solution is distributed among the customers who ordered the width, in so far as is possible within the customers' tolerances. If all extra or short units of a width can be assigned, the amount received by any customer is proportional to his contribution to the maximum that could be assigned within the tolerances. Otherwise, the customer receives as much as he will accept. This done, the order widths are printed, along with the total quantity ordered of each width, the quantity produced in the rounded solution, and, of the difference, the number which could not be assigned to any customer.

Next the cutting patterns in the solution are printed, each indicating the size and quantity of stock to be used for the pattern, and the manner in which the ordered widths produced in the pattern are to be distributed to customers. Following the patterns the individual orders are listed, each entry indicating the width ordered, the customer, the quantity ordered, the number of extra or short units received within the tolerance, and any shortage received beyond the tolerance.

Finally, the program prints the following statistics about the run: the cost of the linear programming solution, the cost of the rounded solution, the trim waste in each solution, and the number of iterations performed in the linear programming solution.

INPUT FORMAT

Input to the One-Dimensional Trim Program consists of a description of the stock available for cutting the order and a description of the order itself. The deck order is:

1. Identification card
2. Stock cards
3. Order cards
4. Blank card

General Instructions

The "Restrictions and Range" section of this manual contains information on the permissible range of values for stock and order widths, quantities and costs. In all fields leading and trailing zeros may be left blank, but decimal points should not be omitted.

IDENTIFICATION CARD

The first card in the data deck is assumed to be an identification card, which simply identifies the run on the printed output. It may contain any alphameric information desired by the user in columns 1-80.

STOCK CARDS

Each stock width is described on a separate card. The first field on the card is the width of the stock. It is usually given in inches, but any unit is acceptable provided the same unit is used throughout the run, both on the stock and the order cards. This field is punched in columns 2-10, with a decimal point in column 5.

The second field on the card is the cost per unit for the stock width. For example, in paper manufacture this number would be the cost per roll of paper of the width being described. Here again, any cost unit may be used, but the cost of all stock widths must be stated in the same units. Cost is punched in columns 12-20, with the decimal point in column 15.

Since it is the total cost of cutting an order which is minimized rather than the total quantity of stock used, the cost associated with a stock may be used to reflect the expense of obtaining or producing or handling that particular width in comparison to other stock widths. Referring again to the example of cutting paper stock, suppose that an order is to be filled from stock produced on two machines, one 90 inches and the other 100 inches in width. Suppose further that a cost analysis shows that a roll of paper from the 90-inch machine costs $90.00 to produce, but that a roll of paper from the 100-inch machine costs only $95.00 because of the greater efficiency of the wider machine. Then these costs may be assigned directly to the stock widths to reflect this difference in efficiency. The program can also be used to minimize trim by simply setting the cost per unit of stock equal to the width of the stock. Under no circumstance should the cost be zero or omitted.

The third field on the card contains the maximum number of units of stock of this width which may be cut up to fill the order. The number should be punched as an integer, without a decimal point, right-adjusted in columns 22-30. If there is no limit on the supply available for this width, the word "unlimited" should be punched in columns 22-30.

The last field on the stock card is the maximum number of ordered widths that may be cut from the stock width. This information is required because frequently only a limited number of knives are available for cutting the stock widths into ordered widths, and consequently some cutting patterns must be eliminated from the solution. If this restriction does not apply, the field should contain any number greater than the number of times the shortest order width divides the stock width. It is an integer, punched in columns 34-35, right-adjusted and with no decimal point.

The columns on the stock card not specifically mentioned above must be left blank. The stock cards are usually placed in descending order of width, but they may appear in some other order if the user desires. There may be as many as 16 different stock widths. Stock cards follow the identification card and precede the order cards. They are distinguished by the presence of a decimal point in column 5.

ORDER CARDS

Each order card describes one width required for one order (that is, for one customer), of the batch which is to be filled. A customer may order a single width requiring one card, or he may order several, in which case there will be as many order cards for his order as there are widths. Likewise, several customers may order the same width, and one card will be punched for each customer for that width. If no customer information is associated with the orders to be filled, the order may be considered to have a single customer, and there will be as many order cards as there are ordered widths.

The first field on the card, columns 1-5, contains the customer name or order number or whatever alphameric information is convenient to identify the source of the order. Some nonblank information should be punched in this field, even if there is only one customer.

The second field contains the width which has been ordered. It is punched in columns 7-15, with the decimal point in column 10. The units in which the width is expressed must be the same as for the stock widths.

The number of units of this width which are required by this customer is punched in the next field. This order quantity should be an integer with no decimal point, right-adjusted in columns 18-25.

In some instances of the trim problem, the quantity required of a certain width may vary within certain limits. Typical orders for paper, for example, indicate not only a quantity desired but also the amount above and below this quantity which is acceptable to the customer. This information is not used in determining the linear programming solution,

but when the linear programming solution is rounded (a process which may produce slight discrepancies between the total number of rolls ordered and total produced of any width), the tolerances are used to distribute these discrepancies among the customers as well as possible. These tolerances may be expressed either as percentages of the order quantity or simply as numbers of units. If program switch 3 is off, the tolerances are taken to be percentages; if the switch is on, they are units. The percentage or number of units below the stated amount of his order that a customer will accept is punched in columns 26-30, without a decimal point, right-adjusted in the field. The percentage or number of units acceptable above the stated order quantity is the last field on the card, columns 31-35, punched in the same way as the percentage or amount acceptable below.

Order cards are identified by a decimal point in column 10. They follow the stock cards in the input deck and may be in any order whatsoever. The unused portion of the order cards, columns 36-80, may be used for any purpose the user wishes.

There may be as many as 125 order cards. However, the number of different order widths plus the number of stock widths for which the supply is not unlimited must not exceed 35. A typical input deck is in the "Sample Problems" section of this manual.

BLANK CARD

A blank card must follow the last order card to indicate the end of the data deck.

Data Format Summary

IDENTIFICATION CARD

Columns 1-80 Alphameric identification

STOCK CARDS

Column	1	Blank
	2-10	Stock width (decimal point in column 5)
	11	Blank
	12-20	Stock cost (decimal point in 15)
	21	Blank
	22-30	Stock supply or "unlimited"
	31-33	Blank
	34-35	Maximum number of usable pieces
	36-80	Blank

ORDER CARDS

1-5	Alphameric order identification
6	Blank
7-15	Order width (decimal point in 10)
16-17	Blank

18-25	Order quantity
26-30	Tolerance below order quantity
31-35	Tolerance above order quantity
36-80	Not used

OUTPUT DESCRIPTION

The output format has been designed specifically for use of the program in cutting paper stock. All of the information necessary for any application is printed, but the headings and formatting are oriented toward this one application.

The first section of the output is primarily a summary of the input data. The identification card is printed, followed by a list of all the stock widths. The stock information appears exactly as in the input deck, except that one column is added. This is the amount of each stock width which was used in the rounded solution. Next there is a list of the order widths, arranged in descending width order. Opposite each width is the total number of rolls of that width required by all of the orders, the number of rolls produced when the linear programming solution was rounded, and, of the difference, the number which could not be absorbed within the tolerance on some order for that width.

The next section of type contains the most important output of the program: the patterns which tell how the stock should be cut up. The patterns are grouped according to the stock width from which they are to be cut, and each group is headed by the statement "Run on machine width XXX. XXXXX". In the first column of pattern information is the number of units of stock to be cut in the pattern. The first number in the column (for a particular pattern) is the number to be cut for the rounded solution. In parentheses immediately below it is the number to be cut in the linear programming solution. The second column of information for a pattern is the number and size of the order widths which make up the pattern. If there is any waste in the pattern, the amount will appear, marked "TRIM", as the last entry in this column. Opposite each width in the pattern, in the third column, is a description of the way in which the amount of that width produced by that particular pattern is to be distributed among the orders. This description consists of a list of amounts and orders, running horizontally across the page.

The third section of output lists the orders which the problem comprises. For each order, the width ordered, the customer identification, and the quantity ordered are copied from the card as the entries in columns 1, 2, and 3 respectively. In column 4 is the number of extra or short rolls assigned to the order within the specified tolerance, if any, and in the last column is the number short beyond the tolerance, if any. The assignments of rolls to customers within their tolerances are made as follows: Suppose there were p extra rolls of a certain width to be distributed, of which q have not yet been assigned to any order; a total of r can be absorbed by all orders. A particular customer has ordered s rolls and will take up to t extra. (Here t may have been specified exactly on the input card or may have been computed as a percentage of s, depending on the setting of program switch 3.) Then

this customer should receive

$$\text{Min } (s + q,\ s + t,\ s + \left\lceil \frac{p \cdot t + r - 1}{r} \right\rceil).$$

If there is a shortage rather than an excess of p rolls, and the customer will accept as many as t rolls too few, then he will receive

$$\text{Max } (s - q,\ s - t,\ s - \left\lceil \frac{p \cdot t + r - 1}{r} \right\rceil).$$

The final portion of the output consists of some statistics about the problem. The total cost of both the linear programming and rounded solutions is printed, along with the percentage of trim waste in each. Finally, the number of iterations performed in the linear programming solution is printed.

SAMPLE PROBLEMS

Both of the following examples describe problems in which orders for various sizes of rolls of paper are to be cut from rolls produced on machines of standard widths. In the first example, the manufacturer is filling the orders from a single machine, 222 inches wide. Since there is only one machine, minimizing the cost of filling the orders is equivalent to minimizing the amount of paper used, or again, to minimizing the amount of waste paper which must be trimmed off. The stock is available in unlimited supply, because it would be pointless to limit the supply of the only stock available. The collection of orders to be filled involves five customers whose various orders have been sorted into ascending order of ordered width, for some reason dictated by the manufacturer's procedures.

The input deck for this problem looks as follows:

```
EXAMPLE PROBLEM 1
 222.00000 222.00000 UNLIMITED    10
JONES   22.75000        396     0     1
JONES   23.5           3728     1     0
JONES   31.00000        135     1     1
JONES   31.50          5086     0     1
ABC     35.            2780     0     0
JONES   35.              14     0     1
SMITH   35.25000          5     1     1
XYZ     35.25          1052     5     5
ABC     35.25000         15     0     0
ABC     35.50            10     0     0
XYZ     35.5           1359     5     5
JONES   40.             180     0     1
ABC     44.             360     0     0
JONES   44.           00008     0     1
JONES   45.             213     0     1
JONES   46.00000       0665     0     1
JONES   47.25000      06590     0     1
SMITH   52.              10     1     1
ABC     52.               2     0     0
JONES   52.25            22     0     1
SMITH   52.25          1400     1     1
XYZ     52.25           100     0    10
ABC     52.25           100     0     0
SMALL   52.25           100     1     5
JONES   53.             498     0     1
JONES   58.75000        635     0     1
XYZ     58.75000        683     5     5
SMITH   58.75          1317     1     1
JONES   63.00000       1181     0     1
JONES   65.625          500     0     1
ABC     65.6250         480     0     0
XZY     65.6250          20     5     5
SMITH   65.62500       1560     1     1
JONES   67.75          3985     0     1
SMITH   68.62500       1032     1     1
```

The first two cards of the deck are the identification card and the stock width card. Next follows an order from Jones for 396 rolls of 22.75-inch paper. Program switch 3 is off, so that the tolerances on the order cards will be considered percentages of the order quantity. Jones will not settle for any number of rolls fewer than 396, but will accept up to one percent (or three rolls) more. The second order is also from Jones, this one for 3,728 rolls of 23.5-inch paper. On this order, he will accept up to one percent (37) fewer rolls than specified, but will take no extras. The last order is from Smith, who wishes 1,032 rolls of width 68.625. He will tolerate a one percent discrepancy in either direction from his stated figure.

Following is the output from the problem.

```
STEP 009, 02.31 PERCENT WASTE
STEP 029, 00.27 PERCENT WASTE
STEP 039, 00.20 PERCENT WASTE
STEP 049, 00.17 PERCENT WASTE
STEP 060, 00.11 PERCENT WASTE
STEP 066, 00.07 PERCENT WASTE
STEP 073, 00.06 PERCENT WASTE
STEP 075, 00.06 PERCENT WASTE
STEP 078, 00.05 PERCENT WASTE

EXAMPLE PROBLEM 1
```

MACHINE WIDTH	COST PER MACHINE ROLL	NUMBER OF ROLLS AVAILABLE	ROLLS USED ROUNDED SOLUTION	MAXIMUM NUMBER PIECES PER ROLL
222.00000	222.00000	UNLIMITED	7549	10

WIDTH ORDERED	NUMBER OF ROLLS ORDERED	ROLLS PRODUCED, ROUNDED SOLUTION	NUMBER OF UNASSIGNED ROLLS
68.62500	1032	1032	
67.75000	3985	3984	1 SHORT
65.62500	2560	2561	
63.00000	1181	1181	
58.75000	2635	2634	
53.00000	498	498	
52.25000	1722	1720	
52.00000	12	12	
47.25000	6590	6590	
46.00000	665	665	
45.00000	213	213	
44.00000	368	368	
40.00000	180	180	
35.50000	1369	1369	
35.25000	1072	1072	
35.00000	2794	2794	
31.50000	5086	5086	
31.00000	135	135	
23.50000	3728	3728	
22.75000	396	396	

RUN ON MACHINE WIDTH 222.00000

MACHINE ROLLS	PATTERN		DISTRIBUTION	
80	1 OF	65.62500	80 FOR JONES	
(79.75158)	2 OF	35.25000	5 FOR SMITH	155 FOR XYZ
	2 OF	31.50000	160 FOR JONES	
	1 OF	22.75000	80 FOR JONES	
	TRIM	0.12500		
498	1 OF	58.75000	498 FOR JONES	
(498.00006)	1 OF	53.00000	498 FOR JONES	
	1 OF	47.25000	498 FOR JONES	
	2 OF	31.50000	996 FOR JONES	
325	1 OF	58.75000	137 FOR JONES	188 FOR XYZ
(324.78935)	1 OF	47.25000	325 FOR JONES	
	1 OF	46.00000	325 FOR JONES	
	2 OF	35.00000	650 FOR ABC	
1181	2 OF	67.75000	2362 FOR JONES	
(1181.00003)	1 OF	63.00000	1181 FOR JONES	
	1 OF	23.50000	1181 FOR JONES	
180	1 OF	52.25000	22 FOR JONES	158 FOR SMITH
(179.99987)	2 OF	47.25000	360 FOR JONES	
	1 OF	40.00000	180 FOR JONES	
	1 OF	35.25000	180 FOR XYZ	
135	1 OF	65.62500	135 FOR JONES	
(135.00007)	1 OF	58.75000	135 FOR XYZ	
	1 OF	35.00000	135 FOR ABC	
	1 OF	31.50000	135 FOR JONES	
	1 OF	31.00000	135 FOR JONES	
	TRIM	0.12500		
164	2 OF	58.75000	328 FOR XYZ	
(164.48136)	2 OF	52.25000	328 FOR SMITH	
12	1 OF	52.00000	10 FOR SMITH	2 FOR ABC
(12.00000)	2 OF	47.25000	24 FOR JONES	
	1 OF	44.00000	12 FOR ABC	
	1 OF	31.50000	12 FOR JONES	
248	4 OF	47.25000	992 FOR JONES	
(247.84241)	1 OF	31.50000	248 FOR JONES	
	TRIM	1.50000		
17	1 OF	52.25000	17 FOR SMITH	
(17.15687)	3 OF	35.50000	10 FOR ABC	41 FOR XYZ
	2 OF	31.50000	34 FOR JONES	
	TRIM	0.25000		
316	1 OF	58.75000	31 FOR XYZ	285 FOR SMITH
(316.24797)	2 OF	47.25000	632 FOR JONES	
	1 OF	46.00000	316 FOR JONES	
	1 OF	22.75000	316 FOR JONES	

24	1 OF	65.62500	24 FOR JONES		
(23.96233)	1 OF	47.25000	24 FOR JONES		
	1 OF	46.00000	24 FOR JONES		
	2 OF	31.50000	48 FOR JONES		
	TRIM	0.12500			
157	1 OF	52.25000	157 FOR SMITH		
(156.88614)	1 OF	47.25000	157 FOR JONES		
	1 OF	35.50000	157 FOR XYZ		
	2 OF	31.50000	314 FOR JONES		
	1 OF	23.50000	157 FOR JONES		
	TRIM	0.50000			
356	1 OF	67.75000	356 FOR JONES		
(355.99989)	1 OF	47.25000	356 FOR JONES		
	1 OF	44.00000	348 FOR ABC	8 FOR JONES	
	2 OF	31.50000	712 FOR JONES		
1032	1 OF	68.62500	1032 FOR SMITH		
(1031.99986)	1 OF	58.75000	1032 FOR SMITH		
	2 OF	47.25000	2064 FOR JONES		
	TRIM	0.12500			
298	5 OF	35.00000	1490 FOR ABC		
(297.98489)	2 OF	23.50000	596 FOR JONES		
213	3 OF	47.25000	639 FOR JONES		
(213.00007)	1 OF	45.00000	213 FOR JONES		
	1 OF	35.25000	213 FOR XYZ		
1161	2 OF	65.62500	262 FOR JONES	480 FOR ABC	20 FOR XZY
			1560 FOR SMITH		
(1160.64311)	1 OF	35.50000	1161 FOR XYZ		
	1 OF	31.50000	1161 FOR JONES		
	1 OF	23.50000	1161 FOR JONES		
	TRIM	0.25000			
633	2 OF	67.75000	1266 FOR JONES		
(633.49999)	2 OF	31.50000	1266 FOR JONES		
	1 OF	23.50000	633 FOR JONES		
519	2 OF	52.25000	738 FOR SMITH	100 FOR XYZ	100 FOR ABC
			100 FOR SMALL		
(519.49709)	1 OF	47.25000	519 FOR JONES		
	1 OF	35.25000	504 FOR XYZ	15 FOR ABC	
	1 OF	35.00000	505 FOR ABC	14 FOR JONES	

SUMMARY OF ORDERS

WIDTH	ORDER NUMBER	ROLLS ORDERED	DIFFERENCE FROM ORDER WITHIN TOLERANCE	DIFFERENCE FROM ORDER BEYOND TOLERANCE
22.75000	JONES	396		
23.50000	JONES	3728		
31.00000	JONES	135		
31.50000	JONES	5086		
35.00000	ABC	2780		
35.00000	JONES	14		
35.25000	SMITH	5		
35.25000	XYZ	1052		
35.25000	ABC	15		
35.50000	ABC	10		
35.50000	XYZ	1359		
40.00000	JONES	180		
44.00000	ABC	360		
44.00000	JONES	8		
45.00000	JONES	213		
46.00000	JONES	665		
47.25000	JONES	6590		
52.00000	SMITH	10		
52.00000	ABC	2		
52.25000	JONES	22		
52.25000	SMITH	1400	2 SHORT	
52.25000	XYZ	100		
52.25000	ABC	100		
52.25000	SMALL	100		
53.00000	JONES	498		
58.75000	JONES	635		
58.75000	XYZ	683	1 SHORT	
58.75000	SMITH	1317		
63.00000	JONES	1181		
65.62500	JONES	500	1 EXTRA	
65.62500	ABC	480		
65.62500	XZY	20		
65.62500	SMITH	1560		
67.75000	JONES	3985		1 SHORT
68.62500	SMITH	1032		

PROGRAM STATISTICS

COST OF OPTIMAL SOLUTION	1676042.93
COST OF ROUNDED SOLUTION	1675878.00
PERCENT TRIM WASTE (OPTIMAL)	0.053
PERCENT TRIM WASTE (ROUNDED)	0.053

ITERATIONS 82

The first nine lines are messages printed during the linear programming phase of the problem when the operator depressed program switch 1 to see how the solution was progressing (see "Operating Instructions"). The rest of the output is as described previously, under "Output Description".

The second example is a much smaller, but in some respects more illustrative, problem. In this problem, the manufacturer has three different machine widths available for cutting the ordered widths. Clearly the 110-inch machine is the most efficient of the three, since it has a cost of only $100 per roll. However, only 500 rolls from it can be used for the solution, whereas, 1,000 rolls are available on the somewhat less efficient 120-inch machine. Output from the third machine is available in unlimited supply, but it has the greatest cost per roll-inch of all.

There are four customers in the problem. A requires ten rolls each of 33- and 29-inch paper and 50 rolls of 20-inch paper. A permits a blanket five percent tolerance, above and below. Cutomer B wishes

70	rolls of 29-inch paper
950	rolls of 20-inch paper
3000	rolls of 18-inch paper
500	rolls of 17-inch paper

The only deviations he will accept are five percent above on the 29-inch order and five below on the 17-inch order. The last two customers' orders are:

Customer C:	60	rolls of 33-inch paper
	10	rolls of 29-inch paper
	1200	rolls of 18-inch paper
Customer D:	20	rolls of 47-inch paper
	80	rolls of 33-inch paper
	10	rolls of 29-inch paper
	800	rolls of 18-inch paper
	180	rolls of 17-inch paper
	1250	rolls of 15-inch paper

Like A, C and D will accept up to five percent above or below the order quantity. The input deck is as follows:

```
EXAMPLE PROBLEM 2
 120.00000 120.00000        1000     8
 110.00000 100.00000         500     8
 100.00000 110.00000   UNLIMITED     8
A       33.00000          10     5     5
A       29.00000          10     5     5
A       20.00000          50     5     5
B       29.00000          70     0     5
B       20.00000         950     0     0
B       18.00000        3000     0     0
B       17.00000         500     5     0
C       33.00000          60     5     5
C       29.00000          10     5     5
C       18.00000        1200     5     5
D       47.00000          20     5     5
D       33.00000          80     5     5
D       29.00000          10     5     5
D       18.00000         800     5     5
D       17.00000         180     5     5
D       15.00000        1250     5     5
```

The output is:

```
EXAMPLE PROBLEM 2

   MACHINE          COST PER    NUMBER OF ROLLS        ROLLS USED    MAXIMUM NUMBER
     WIDTH     MACHINE ROLL          AVAILABLE  ROUNDED SOLUTION   PIECES PER ROLL

 120.00000        120.00000               1000               784                 8
 110.00000        100.00000                500               500                 8
 100.00000        110.00000          UNLIMITED                 0                 8


    WIDTH          NUMBER OF   ROLLS PRODUCED,          NUMBER OF
  ORDERED      ROLLS ORDERED  ROUNDED SOLUTION  UNASSIGNED ROLLS

  47.00000                20                20
  33.00000               150               149
  29.00000               100                99          1 SHORT
  20.00000              1000              1003          1 EXTRA
  18.00000              5000              4999
  17.00000               680               680
  15.00000              1250              1250

RUN ON MACHINE WIDTH  120.00000

    MACHINE
      ROLLS          PATTERN          DISTRIBUTION

         20   1 OF  47.00000        20 FOR D
(  20.00000)  1 OF  20.00000        20 FOR A
              2 OF  18.00000        40 FOR B
              1 OF  17.00000        20 FOR B
```

58	2 OF	33.00000	10 FOR A	59 FOR C	47 FOR D
(58.33333)	3 OF	18.00000	174 FOR B		
33	1 OF	33.00000	33 FOR D		
(33.33333)	3 OF	29.00000	10 FOR A	70 FOR B	10 FOR C
			9 FOR D		
33	6 OF	20.00000	32 FOR A	166 FOR B	
(32.50002)					
625	5 OF	18.00000	2786 FOR B	339 FOR C	
(625.00000)	2 OF	15.00000	1250 FOR D		
15	1 OF	18.00000	15 FOR C		
(15.00000)	6 OF	17.00000	90 FOR B		

RUN ON MACHINE WIDTH 110.00000

MACHINE ROLLS	PATTERN		DISTRIBUTION		
285	2 OF	20.00000	570 FOR B		
(285.00000)	2 OF	18.00000	570 FOR C		
	2 OF	17.00000	390 FOR B	180 FOR D	
215	1 OF	20.00000	214 FOR B	1 UNORDERED	
(215.00000)	5 OF	18.00000	275 FOR C	800 FOR D	

SUMMARY OF ORDERS

WIDTH	ORDER NUMBER	ROLLS ORDERED	DIFFERENCE FROM ORDER WITHIN TOLERANCE	BEYOND TOLERANCE
33.00000	A	10		
29.00000	A	10		
20.00000	A	50	2 EXTRA	
29.00000	B	70		
20.00000	B	950		
18.00000	B	3000		
17.00000	B	500		
33.00000	C	60	1 SHORT	
29.00000	C	10		
18.00000	C	1200	1 SHORT	
47.00000	D	20		
33.00000	D	80		
29.00000	D	10		1 SHORT
18.00000	D	800		
17.00000	D	180		
15.00000	D	1250		

PROGRAM STATISTICS

COST OF OPTIMAL SOLUTION	144100.00
COST OF ROUNDED SOLUTION	144080.00
PERCENT TRIM WASTE (OPTIMAL)	0.000
PERCENT TRIM WASTE (ROUNDED)	0.000

ITERATIONS 10

As can be seen from the "Number of Rolls Ordered" and "Rolls Produced, Rounded Solution", exactly the right number of rolls of widths 47, 17, and 15 were produced in the rounded solution. Of the widths 33, 29, and 18, however, one fewer roll was produced than was required. In the case of both 33 and 18, the shortage was assignable (to customer C, as shown in the "Summary of Orders"), but the missing 29-inch roll could not be assigned to either B, who specified no deviation on that order, or A, C, or D, who each ordered only ten rolls, five percent of which, rounded, is zero. The shortage eventually fell, beyond the tolerance, on D, as is also shown in the "Summary of Orders". Three extra rolls were produced of the 20-inch paper; two of these were assigned to customer A (five percent of 50), but the other could not be assigned since the only other customer specified no extra rolls. It appears, in the distribution for the last pattern, as "unordered" and will presumably go into inventory.

In this last pattern, 215 rolls are to be cut from the 110-inch machine. Each roll so cut is to contain one 20-inch piece and five 18-inch pieces.

Note:

$$1\,(20) + 5\,(18) = 110$$

so that there is no trim waste at all in this particular pattern. This will produce 215 rolls of 20-inch and 5 x 215 = 1075 rolls of 18-inch paper. Of the 215 rolls of 20-inch paper, 214 are for B, who previously received 736 rolls, and as mentioned before, one simply was not ordered. The 1075 rolls are split as follows: 275 to C, who has already received the other 924 he will receive, and 800 to D, who has not previously received any 18-inch rolls.

PROGRAM MODIFICATIONS

1443 Printer Output

The program output has been written so that if a printer is available, the program can be modified fairly easily for this kind of output. The following changes should be made. Cards

```
07260       RCTY,                       ,,IS DOWN, TERMINATE
07270       WATYHOL.0,                  ,,SOLUTION. OTHERWISE
```

should be replaced by

```
07260       WA  HOL.0,900               ,,IS ON, STOP.
07270       NOP ,                       ,,OTHERWISE
```

in part 2 of the program. In part 3, cards

```
13140FILL.0WATY-FILL.0+1,               ,,WRITE PARTIAL LINE
13150       BB  ,                       ,,AND RETURN.
13160       DORG*-3                     ,,
13170PRNT.0WATY-PRNT.0+1,               ,,WRITE ALL OR REST OF LINE
13180       B   SKIP.0,                 ,,AND RESTORE AND RETURN.
13190SKIP.2RCTY,                        ,,SPACE TWO LINES
13200       NOP ,                       ,,AND RETURN.
13210SKIP.0RCTY,                        ,,SPACE ONE LINE
13220       BB  ,                       ,,AND RETURN.
13230       DORG*-9                     ,,
13240EMPT.0DS   ,SKIP.0                 ,,EMPTY WRITE BUFFER(DUMMY).
13250REST.0DS   ,SKIP.2                 ,,RESTORE TO NEW PAGE(DUMMY).
```

should be replaced by

```
13140PRNT.0SM   PRNT.0-1,01,10          ,,WRITE REST OR ALL
13144PRNT.1TR   WBUF,-PRNT.0+1          ,,OF LINE FROM WBUF.
13148       NOP ,                       ,,
13152EMPT.0WA   WBUF+1,900              ,,AFTER PRINTING RESTORE
13156       TFM PRNT.1+6,WBUF           ,,ADDRESS TO BEGINNING
13160       TR  WBUF,PBUF-1             ,,OF BUFFER.
13164       TR  WBUF+78,PBUF-1          ,,CLEAR BUFFER,
13168       BB  ,                       ,,AND RETURN.
13172       DORG*-3                     ,,
13176FILL.0SM   FILL.0-1,01,10          ,,PLACE PARTIAL LINE
13180       TR  -PRNT.1-6,-FILL.0+1     ,,IN BUFFER WBUF.
13184       AM  PRNT.1+6,01,10          ,,COMPUTE LOCATION OF
13188FILL.1AM   PRNT.1+6,02,10          ,,NEXT AVAILABLE SPACE
13192       BNR FILL.1,-PRNT.1-6        ,,IN BUFFER.
13196       SM  PRNT.1+6,01,10          ,,
13200       BB                          ,,RETURN.
```

```
13204       DORG*-3                          ,,
13208SKIP.2K    ,951                         ,,SPACE TWO LINES AND
13212       NOP ,                            ,,RETURN.
13216SKIP.0K    ,951                         ,,SPACE ONE LINE
13220       BB  ,                            ,,AND RETURN.
13224       DORG*-3                          ,,
13230REST.0K    ,971                         ,,RESTORE PAGE AND
13240       BB  ,                            ,,RETURN.
13250       DORG*-9                          ,,
```

Furthermore, if a printer is available, the user may wish to have the patterns for each stock size on a separate sheet. If this is desired,

```
10910       BTM SKIP.2,0                     ,,PRINT HEADING
```

should be replaced by:

```
10910       BTM REST.0,0                     ,,PRINT HEADING
```

and

```
11930       BTM SKIP.2,0                     ,,AFTER ALL PATTERNS PRINTED,
```

should be replaced by:

```
11930       BTM REST.0,0                     ,,AFTER ALL PATTERNS PRINTED,
```

ROUNDED SOLUTION

The way in which the quantities of the patterns to be produced in the linear programming solution are rounded to integers for the rounded solution is governed by a parameter in the program, THETA. If X units of a certain pattern are to be cut in the linear programming solution, then the number to be cut in the rounded solution will be greatest integer contained in $[X + THETA]$. The present value of THETA is .5, defined by the card

```
07540THETA DC  10,5000000000            ,,ROUND-UP LIMIT.
```

so that if the fractional part of the pattern quantity is one-half or greater, the quantity is rounded up. Otherwise it is rounded down.

This parameter can be changed to any number between 0 and 1.0 the user wishes. For instance, if the user wished always to round up, he would reassemble the program with

```
07540THETA DC  10,9999999999            ,,ROUND-UP LIMIT.
```

If he always wished to truncate, the card defining THETA would be changed to

```
07540THETA DC  10,0000000000            ,,ROUND-UP LIMIT.
```

MAXIMUM NUMBER OF STOCK AND ORDER WIDTHS

The limits on the number of stock widths, order widths, and order cards are set by parameters at the beginning of the program. Symbolically, the maximum number of stock widths is defined as LDIM, presently equated to 16. The maximum number of distinct order widths is MDIM, now set at 35; this number is also the limit on the sum of the number of order widths and the number of stock widths with limited supply. The maximum number of separate order cards is defined as CDIM, presently 125. These particular values were chosen to make the program as generally useful as possible within the limitations of 20K storage.

If more storage is available or if a different arrangement is desired, these limits may be changed as desired, with two restrictions: (1) because of the way the variables L through SCALE are attached to the vector KUT in the program, LDIM should always be at most MDIM-17, unless these variables are redefined elsewhere in DC statements; (2) if MDIM exceeds 39, the buffer PBUF must be doubled in length, and every statement which punches from or reads into PBUF must be replaced by two statements which write and read both halves of the buffer.

FIELD LENGTH

The field lengths defined symbolically early in the program at DS, DI, DB, DF, and DC must not be altered. The program logic depends materially on the present choice of values.

OPERATING INSTRUCTIONS

Operating procedure for the One-Dimensional Trim Program divides into three phases which correspond to the three overlay sections of the program. The steps which occur in the processing of a single problem are outlined by section below, with specific operating procedures numbered and indented.

Program Section 1

The first section of code performs various initialization steps, reads the data deck, sets up data arrays for section 2, and punches a few cards of intermediate output. To prepare the 1620 system for this program section, the operator should:

1. Set console check switches. The arithmetic check switch (O FLOW) should be set to PROGRAM. The other switches (I/O, PARITY, and, on Model II only, DISK) should be set to STOP.

2. Set program switches. PROGRAM SWITCH 1 may be ON or OFF (see "Program Section 2" below). SWITCH 2 must be OFF. Switch 3 should be OFF if the order quantity tolerances are expressed as percentages, ON if they are expressed as units, as explained in the "Input Format" section of this manual.

3. Set typewriter margins. The left margin should be as far left as possible and the right margin at 85 or beyond.

4. Clear memory and reset computer. On Model I the procedure is:

   Depress INSTANT STOP key
   Depress RESET key
   Depress INSERT key
   Type 260000800009 $\begin{matrix}R\\S\end{matrix}$ and wait 10 seconds
   Depress INSTANT STOP key
   Depress RESET key

   On Model II:

   Depress INSTANT STOP key
   Depress RESET key
   Depress MODIFY and CHECK RESET keys simultaneously
   Depress START key
   Depress START key
   When MANUAL light appears, depress RESET key

5. Ready the punch. Clear any cards presently in the punch by removing new cards from the punch input hopper and depressing the punch NON-PROC RUN OUT key for a few seconds. Discard any cards left in the stackers. Place a supply (200 or so) of blank cards in the punch hopper. These cards should be different in color both from the data deck and from the program deck. Depress PUNCH START key on the reader-punch.

6. Load program and data. Place the data deck, including the blank trailer card, between the cards numbered 065 and 066 in the program deck. The program deck cards are numbered sequentially in columns 78-80. Place the program deck with the imbedded data in the read hopper, and depress the LOAD key on the reader-punch.

As soon as the LOAD key is depressed, section 1 of the program is loaded into core and begins execution. In rapid succession, it reads the data deck, punches a few cards of intermediate output, and initiates loading of Section 2.

Program Section 2

Program Section 2 performs the basic linear programming step iteratively until the solution is optimal or is interrupted. One step consists of generating a pattern and performing a Gaussian elimination step. During each such step, one card of intermediate output is punched. Execution of section 2 begins automatically after loading. Unless the problem is unusually small or easy, this section will take some time to execute — exactly how much time is impossible to estimate, although some idea may be obtained from the table under "Timing" in the "General Description" section of this manual. The operator has two options to exercise during this part of the program:

1. He may inspect the progress of the solution at any time by turning PROGRAM SWITCH 1 to ON. As long as the switch is ON at the end of each iteration of the linear programming solution, the percentage of trim waste in the current solution and the number of iterations completed are typed out.

2. He may terminate the solution at the end of the current linear programming step by putting PROGRAM SWITCH 2 ON (see remarks under "Timing").

Since one iteration may take several minutes under certain circumstances, the program should not be expected to respond immediately to either switch.

As soon as section 2 has reached an optimal solution (or been interrupted by PROGRAM SWITCH 2), it initiates loading of the third and final section of the code.

Program Section 3

The last section of the program reads the intermediate output which was punched by the first two sections, rereads the data deck, and types out all of the program output. Loading of this section proceeds until the last card of the program deck has fed part way into the reader. At this time, the computer will pause with the console light READ INTERLOCK on. The operator should:

1. Clear the punch. Again, remove blank cards from the input hopper, and depress the punch NON-PROC RUN OUT key for several seconds. The last card of punched intermediate output followed by two blank cards will fall into the leftmost stacker behind the other punched cards.

2. Re-enter intermediate output. Take the punched card deck from the leftmost stacker, remove the two blank cards at the end, and place the deck in the read hopper.

3. Re-enter data. Remove the data deck from the program deck, most of which is now in the rightmost stacker (the last two cards are still in the reader). Place the data in the reader, behind the punched output.

4. Depress READER START key on the reader-punch.

The last cards of the program are now loaded and the computer pauses with MANUAL lighted on the console. To initiate execution of the final section of code, the operator should:

5. Depress START key on the console.

This section begins by reading the punched output and the first part of the data deck. It then types the identification card and stock information and reads the rest of the data deck. As the last card of the data deck (the blank card) feeds into the reader, the computer will again pause

with READ INTERLOCK (READ NO FEED) lighted on the console. The operator should:

6. Depress READER START key on the reader punch. The program will type the rest of the output and stop at 19150.

Data Deck Order

- Identification card
- Stock cards
- Order cards
- Blank card

The cards are placed between cards 65 and 66 of the object deck.

PROGRAM SETUP SHEET

One-Dimensional Trim Program

Program Sections 1 and 2

1620

INPUT			OUTPUT		
Unit	Description	Source	Unit	Description	Disposition
1622 reader	Data	Keypunch	1622 punch	Intermediate output	Input to section 3
1622 punch	Blank cards				

Program Sw. *	1	2	3	4
On or Off	X		X	Not used
Off		X		Not used

Console Check Switches

O FLOW	PROGRAM
I/O	STOP
PARITY	STOP
DISK	STOP

Typewriter margins: Left 0 and right 85.

Setup

1. Clear memory and reset computer.
2. Press PUNCH START.
3. Place data deck, including the blank trailer card, between cards 065 and 066 of program deck, and then put program deck with imbedded data in reader.
4. Press LOAD.

* Note: Switches 1 and 2 may be used to inspect and terminate the solution respectively, as described under "Operating Instructions".

One-Dimension Trim Program

Program Section 3

1620

INPUT			OUTPUT		
Unit	Description	Source	Unit	Description	Disposition
1622 reader	Intermediate output	Sections 1 and 2	Type-writer	Program results	Management
1622 reader	Data	Keypunch			

Console check switches are the same as in sections 1 and 2. The only program switch which affects this section of the program is switch 3. It should be ON if order quantity tolerances are expressed as units, OFF if they are percentages.

Setup

1. Clear punch and place intermediate output in reader (after removing two blank cards at end).
2. Remove data from program deck and place in reader, after intermediate output.
3. Depress READER START.
4. Depress START.

HALT LIST

There are seven error halts in the program. Except for the last one, no messages are provided; the cause of the stop may be determined from the Memory Address Register and the table below. On a 1620, Model I, Memory Address Register contains the origin of the halt instruction +11; on Model II, it contains the origin +10.

Program section	Origin of halt instruction	Cause of stop	Correction procedure
1	16330	More than 16 stock width cards.	Reduce number of stock widths.
1	16666	More than 125 order cards.	Combine orders or otherwise reduce number of cards.
1	16848	More than 35 distinct order widths.	Reduce number of order widths.
1	17898	An ordered width exceeds all stock widths.	Supply a stock of adequate width or remove the offending order width.
1	18548	Sum of distinct order widths and stock widths with limited supply exceeds 35.	Reduce one or the other or both.
2	18568	Uncontrollable roundoff error.	See note.
3	15332	Supply of stock insufficient to cut entire order.	Increase supply by at least the amount specified in the typed message.

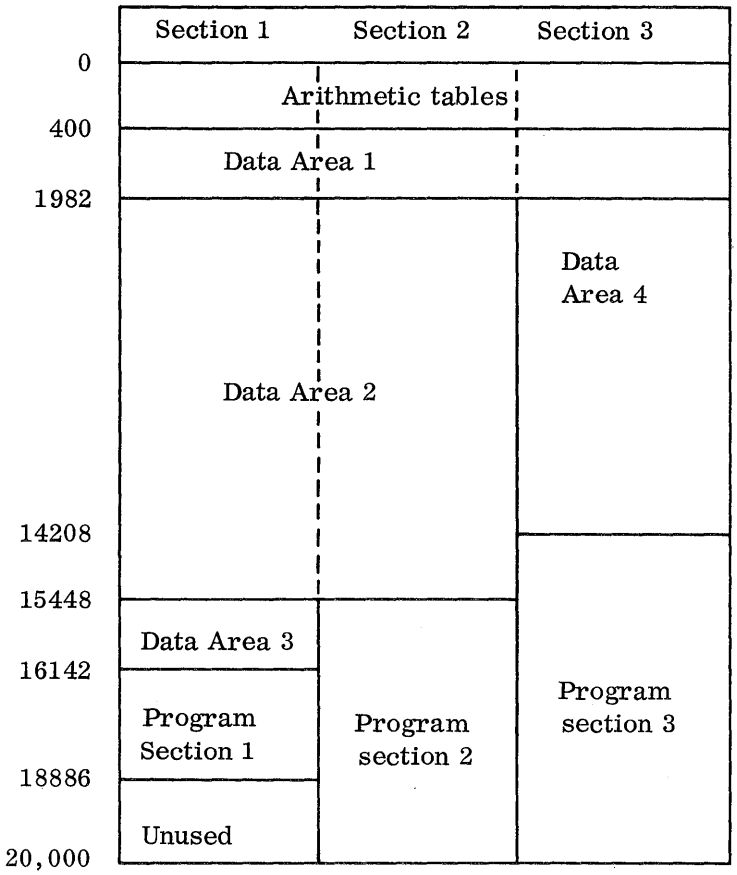
Memory must be completely cleared for proper execution of the program. If core is not properly cleared, a checkstop may occur, the location of the check depending on the part of core not cleared. If this occurs, clear memory and restart the program.

After any error stop, the data must be corrected and the entire problem rerun.

The normal end-of-program stop instruction origins at 19150.

Note: this stop should never occur. It is placed in the program to prevent erroneous results in the event that the data should produce a pivot column with no acceptable pivot row. A slight change in the order data should eliminate the problem.

STORAGE MAP


Section 1
Section 2
Section 3
0
Arithmetic tables
400
Data Area 1
1982
Data Area 2
Data Area 4
14208
15448
Data Area 3
16142
Program Section 1
Program section 2
Program section 3
18886
Unused
20,000

BIBLIOGRAPHY

1. Gilmore, P. C., and R. E. Gomory, "A Linear Programming Approach to the Cutting Stock Problem", Operations Research 9, 849-859, Number 6, November - December, 1961.

2. Gilmore, P. C., and R. E. Gomory, "A Linear Programming Approach to the Cutting Stock Problem, Part II", Operations Research 11, 863-888, Number 6, November - December, 1963.

3. Vajda, S., Mathematical Programming, Addison-Wesley Publishing Company, Reading, Massachusetts, 1961.

4. Wade, C.S., Cutting Stock 1 (PK CSS 1), SHARE Library program Number 1485, IBM Data Processing Division Program Distribution Service.

Printed in U. S. A. H20-0102-0

IBM®
International Business Machines
Data Processing Division
112 East Post Road, White Plains, New York 10601