The trim prolem

by R. E. Gomory

The Trim Problem has many forms, and most of this paper will be devoted to a discussion of the simplest one, the so-called one-dimensional Trim Problem. Some other—and more difficult forms—will be mentioned at the end, but even the simplest case has both applications significance and technical difficulties that make it interesting. To gain some idea of the problem, consider this case.

A paper company owns paper machines which produce wide rolls of paper. Sometimes, in practice, these rolls may be twelve feet in width. The paper company's customers, however, cannot use rolls of this form but need, let us say, 50-inch-wide rolls. The paper company, then, is faced with customer demand for rolls of various width and in various quantities of each width. The demand is to be met by cutting up the wide rolls into narrow rolls. In this process, due to the fact that the smaller rolls do not add up perfectly to make a large one, a certain amount of paper at the end of the rolls goes to waste. A typical Trim Problem, then, is to cut up a given supply of large rolls into smaller ones in such a manner that the demands for each smaller width are met and so that the paper loss is minimized.

Notes

That one approach to this problem is by way of linear programming has been known for some time. In particular, one can cite an article by A. E. Paull in the *Pulp and Paper Magazine of Canada*, January, 1956; an abstract in *Econometrica* in July, 1955, by A. E. Paull and John R. Walter; and an article in *Management Science* in April, 1957, by Kurt Eisemann. To formulate the Trim Problem as a system of linear inequalities, one can proceed as follows.

Each way of cutting up a roll of paper is associated with a variable. The j^{th} pattern of cutting is associated with the variable x_j (a pattern of cutting would be, for example, to cut a twelve-foot roll into three 32-inch rolls, one 30-inch roll and one 15-inch roll). We can then write down the system of inequalities $\sum a_{ij}\, x_j \leq d_i$. If d_i is the amount the customers demand of the i^{th} width, a_{ij} is the number of pieces of that width provided by the j^{th} cutting pattern, and x_j is the number of rolls that are cut up using the j^{th} pattern, then this system of inequalities merely expresses the fact that all customer orders are met.

However, we not only want to meet the orders, but we want to do so and minimize paper wastage. To do this, we must add one more consideration; that is, to the j^{th} pattern, we associate the number l_j which is the loss of waste paper incurred using that pattern. The total paper lost then is $\sum l_j x_j$ and the full problem is to choose x's that satisfy the set of inequalities above and minimize the linear expression $\sum l_j x_j$. To minimize a linear expression subject to linear inequalities is, of course, a linear programming problem.

Although we are still discussing only the simplest form of the one-dimensional Trim Problem, we are already confronted with two technical difficulties.

1. In solving our linear programming problem, since the x_j are to be interpreted as the number of rolls to be cut up according to the j^{th} pattern, the x_j must be whole numbers. Unfortunately, ordinary linear programming does not generally yield whole numbers so that the solution obtained by ordinary methods will have to be changed or rounded in some way before it can be used.
2. If the number of different widths demanded by the customers is large and the rolls to be cut up are large in width, the number of possible patterns to be considered can become astronomical. If 20 widths are demanded, the demanded widths have a reasonable distribution and the basic roll size is 300 inches, the number of combinations exceeds 100,000. It is, therefore, possible for the number of combinations to defy writing down and to be certainly too large for incorporation into a linear programming computation.

In spite of this, people have employed linear programming effectively to solve these problems. The reasons why it has been possible in some instances to ignore these difficulties are as follows.

Difficulty 1. In some industries, notably the paper industry, it is often understood that the customer will accept a number of rolls that is slightly different from the number he has asked for. Therefore, one can take a non-integer solution, round the numbers up, and the extra rolls obtained from this rounding up process will be accepted by the customer and do not constitute extra waste. Another factor, which again operates in some paper applications, is that the resulting x_j values are quite large. Thus, the waste introduced by any method of rounding to a nearest integer is considered to be negligible. However, in other problems capable of the same formulation, such as the problem of cutting up steel beams into shorter ones, neither of these mitigating factors apply, and this is one reason why linear programming has not been used in that application.

Difficulty 2. This can be overcome in small problems by simply enumerating all possible patterns. As the size of the problem grows and the situation becomes worse, some of the patterns are dropped from consideration. For example, one might consider only those patterns whose trim loss is four per cent or less. On large problems, however, even this approach is not easy, for one must still generate an enormous number of columns even with these exclusions, and it sometimes happens (see, for example, the paper by P. F. Reith entitled "The Trim Problem" published by IBM, Amsterdam) that a pattern which is bad in itself (wastes a lot of paper) will figure in the solution because it has the effect of allowing a great many of other effective patterns to be used. There is also the great practical nuisance of having to generate a new collection of patterns whenever a new problem involving any different widths has to be solved.

One could, of course, hope to dispose of Difficulty 1 by means of integer programming; i.e., the systematic methods that have been developed to solve linear programming problems whose variables are construed to be integers. However, the present state

of development of these techniques does not permit the solution of problems with a large number of variables, and it seems that at the present time it is fair to say that integer programming cannot make a direct contribution to solving the first difficulty.

The second difficulty is much less fundamental and can be overcome by the methods described by Gilmore and Gomory in IBM Research Report No. 408. The essence of the idea is this. One starts with a small arbitrary collection of patterns—just enough patterns so that the customer's demands can be met in some fashion by using just these patterns and solving the associated small linear programming problem. If there are n different widths demanded, one need only use n patterns in this part of the problem. Once this problem has been solved, one uses the prices obtained from the linear programming solution to generate a new cutting pattern that will lead to an improved solution.

I will first describe this process mathematically and then give a short intuitive explanation.

The original problem is to solve a large linear programming problem $Ax \geq d$ involving a large matrix A.

$$\begin{bmatrix} 1 & a_{0j} & & & & \\ & \cdot & -1 & & & \\ & \cdot & & -1 & & \\ & a_{ij} & & & -1 & \\ & \cdot & & & & \\ & \cdot & & & & \\ & a_{nj} & & & & -1 \end{bmatrix} \begin{bmatrix} z \\ x_1 \\ \cdot \\ x_i \\ \cdot \\ \cdot \\ x_j \\ \cdot \\ \cdot \\ s_i \\ \cdot \\ \cdot \\ s_n \end{bmatrix} = \begin{bmatrix} 0 \\ d_1 \\ \cdot \\ d_i \\ \cdot \\ \cdot \\ d_n \end{bmatrix}$$

Each possible pattern is represented by a column. The entry in the top row of the j^{th} column a_{0j} is the width of roll or the cost C associated with a standard roll. (Minimizing the total paper width used and minimizing the waste are, of course, equivalent objectives.) Solving a restricted linear programming problem, say one involving only the first $n + 1$ columns and the slacks (i.e., considering only these patterns) means finding a $(N + 1 \times N + 1)$ matrix $P = (p_{ij})$ such that Pd has only non-negative entries and such that the top row Π of P (which will automatically be of the form $(1, -\Pi_1 \cdots -\Pi_n)$ with all $\Pi_i \geq 0$) has a non-negative scalar product with each allowed column. Suppose we find such a Π. We now want to know whether or not an improvement can

be made by including some omitted column; i.e., if we multiply the big matrix A by P, will there be any negative entries in the top row? Or, alternatively, does there exist a column whose scalar product with Π is negative? Any such column is a string of non-negative integers a_i such that $\sum a_i w_i \leq W$ when w_i is the i^{th} width and W the standard width, so the question is, do there exist non-negative integers a_i such that $\sum a_i w_i \leq W$ and $\sum a_i \Pi_i > C$. If we maximize $\sum a_i \Pi_i$ subject to $\sum a_i w_i \leq W$ we can easily test afterward to see if the maximum value exceeds C, so the essential problem that must be solved is

$$\max \sum_{a_i} \Pi_i$$
$$\sum_{a_i w_i} \leq W$$

where the a_i are non-negative integers. It is fortunate that this maximizing problem, which seeks a new pattern for us, is of a recognizable type, for it is the so-called "knapsack problem" which has been the subject of several articles, especially that of Dantzig, "Discrete Variable Extremum Problems," Operations Research Volume 5, No. 2.

The "knapsack problem" gets its name from the following intuitive interpretation—the single inequality appearing is taken as the capacity of the soldier's knapsack; that is, it represents the most weight which he can reasonably carry. The w_i are the weights of the individual articles which he might put into the knapsack and the Π_i represent the values to him of each article. His problem is to select how much of each article to include in the knapsack so that the weight limit is not exceeded and he carries the most valuable articles possible. This is the problem of selecting the a_i.

As these articles point out, the "knapsack problem" can be resolved by the techniques of dynamic programming (Bellman, *Dynamic Programming*, Princeton University Press, 1957). In our case, this means that we introduce the function $c_m(x)$, defined as being the solution to a problem in which use of the first m articles only is allowed and the weight limit, instead of being W is x. The function $c_m(x)$ can be computed recursively by means of the formula

$$c_m(x) = \max_{a_m} \{a_m \Pi_m + c_{m-1}(x - a_m w_m)\}$$
$$0 \leq a_m w_m \leq x$$

which says, essentially, that the m article, x capacity, knapsack problem can be reduced to the $m - 1$ problem by considering each possible amount of the m^{th} article and combining it with the best possible use of the remaining available weight in the knapsack. By means of this recursion formula, c_m can be computed from a knowledge of c_{m-1} and this process iterated until c_n is at hand where n is the number of articles. Then, $c_n(w)$ gives the value of the solution to the maximizing problem, and the a_i that actually attain that value are easily obtained by the standard methods of dynamic programming. Although the dynamic programming approach to the knapsack problem is the easiest one

to describe, other approaches are possible and sometimes better; one of these, which is rather difficult to describe briefly, is, in fact, being used in our latest program. (See P. C. Gilmore, "An Algorithm for the Generalized Knapsack Problem," IBM Research Note NC-15.)

Reverting to a more intuitive description, we can say this. One obtains from the linear programming solution a price Π_i for each width demanded. What these Π_i measure is how much it costs to produce rolls of that width using present cutting patterns. With present patterns, some widths may be easy or cheap to obtain and others expensive. If then a special algorithm is used to generate a new pattern to provide widths which are currently difficult or costly to obtain, it will produce an improvement over the present methods. This is the function of the knapsack sub-routine. Once the new pattern is discovered, it is added to the problem (another pattern automatically drops out), and the present solution is improved. This leads to the solution of a new sub-problem, a new vector Π, etc. This process is iterated until a solution is reached which no further pattern can improve, and this is, of course, optimal.

Using this technique, it has been possible to solve on the IBM® 7090 problems involving very large numbers of patterns and to obtain an answer that uses the best possible combination. This approach, in addition to considering all possibilities, has the following practical advantage—it is only necessary to load the machine with the available roll widths of the customers' demands. The machine itself generates the patterns as they are required. It is not necessary to create, maintain or change a library of useful patterns. W. E. Winans of IBM, Green Bay, Wisconsin, is currently writing an IBM 1620 program based on this technique.

Of course, the problem we have described is the very simplest one and in the very simplest form. I will cite a few complications.

1. Not all patterns of cutting are allowable because the cutting machines have only a certain number of knives.
2. The customers don't want rolls of paper but want sheets whose length as well as the width is specified.

The first of these complications can be dealt with quite easily. By certain modifications in the sub-routine, patterns whose cutting would require too many knives can be eliminated from consideration. The second is much more difficult, and I do not know of any fully satisfactory method of approach. For a fuller description of the second difficulty, an excellent reference is the paper by P. F. Reith cited earlier.

The second complication leads naturally to consideration of the so-called two-dimensional Trim Problem. This problem is encountered in metal industries where, instead of cutting widths from a roll, rectangles of different sizes are to be cut from large rectangular sheets. Linear programming can again be applied and the two technical difficulties occur again, with the number of possibilities occurring under the second difficulty being even greater than before. However, the application of the remedy

described above does not appear to be easy. Still further complications are being investigated by Helmut Maak, IBM Muhlheim, Germany.

Still other variants, both of problem and technique, are possible. Reith has studied the very interesting inverse problem; i.e., given a demand for paper, what width paper machine; i.e., what standard roll is best? Also, in Europe, M. Genuys of IBM Paris has done work on a column-generating technique that uses integer programming at the point where dynamic programming was used in the method described above. Here in the United States, C. E. Berry of IBM Portland has a 1620 program which gives an approximate solution to the problem and proceeds by quite different methods—no linear programming is employed. A technical description of this work will be available soon.

It seems likely that various forms of the Trim Problem will be with us for some time to come.

On modifying the 1620 ADD table

by G. Gerson

The fact that arithmetic is performed on the IBM® 1620 Data Processing System by means of TABLE LOOKUP provides an opportunity for introducing special operations which may be executed by suitable table modification.

It is the purpose of this note to suggest, by means of an example, that many practical problems may be resolved by this technique. The problem faced was as follows.

A 40-digit field was to be tested. Each of the digits of this field arose as the result of a physical test, and was independent of each of the other digits in the field. The physical tests were associated with a production process, and the accompanying computations had to be performed in real time. The field was to be compared with two other 40-digit fields called the *upper limit field* and the *lower limit field.* The requirement was that each digit of the original field had to be less than or equal to the corresponding digit of the upper limit field and greater than or equal to the corresponding digit of the lower limit field. Furthermore, there were 96 sets of these upper and lower limit fields. In the worst possible case, the original field had to be compared to all 96 sets, and the real time requirements were such that all comparisons had to be performed on the 1620 in one second. A careful timing estimate using the COMPARE operation (which required that the digits be stripped away and tested one by one) indicated