AN ALGORITHM FOR SOLVING THE ONE DIMENSIONAL CUTTING STOCK PROBLEM

by

Andrew L. Crouter

A thesis submitted to the Faculty and the Board of Trustees of the Colorado School of Mines in partial fulfillment of the requirements for the degree of Master of Science (Mathematical and Computer Sciences).

Golden, Colorado

Date 6-3-96

Signed: Andrew L. Crouter

Andrew L. Crouter

Approved: Ruth A. Maurer

Dr. Ruth A. Maurer
Thesis Advisor

Golden, Colorado

Date 6-4-96

Graeme Fairweather

Graeme Fairweather
Professor and Head
Department of Mathematical
and Computer Sciences

ABSTRACT

The cutting stock problem - the problem of cutting material from stock in the most efficient manner - has been studied in the field of operations research for many years. Instead of rounding up a non-integer valued answer to the problem generated by a linear programming model to get an exact solution, a method of rounding down the non-integer valued answer is developed here. This rounding down heuristic proved to be effective on problems where the optimal answer was known. The amount of wasted material resulting from the solution generated was used as an indicator of effectiveness on problems where the optimal answer was unknown.

TABLE OF CONTENTS

LIST OF TABLES AND FIGURES

ACKNOWLEDGMENTS

I would like to first thank Dr. Maurer for serving as my advisor. Her commitment to teaching classical operations research, as well as trying to expose students to current methods and tools used, has fostered my interest in an ever changing environment. After spending considerable time with Dr. Maurer, she is indeed the dual of Dr. Woolsey to whom I would also like to express my thanks. Dr. Woolsey's mix of showmanship and knowledge in the field of operations research will no doubt be an inspiration to me and to all those who choose to study under him. I want to acknowledge Dr. Woolsey for keeping my confidence level high in writing this thesis. I shall also acknowledge Dr. Nelson for serving on my thesis committee, providing useful insights which helped me think clearly in times of confusion and constantly reminding me of practical issues that warranted consideration.

Chapter 1

INTRODUCTION

1.1 The Cutting Stock Problem

Consider the task of filling an order for a requested number of various specified lengths of a material such as aluminum, steel, or wood. In most cases these specified lengths must be cut from longer pieces of stocked material. The process of cutting to fill an order usually results in some amount of wasted, or lost, material. Now if there are different lengths of material requested, there is often a choice that can be made as to how to cut these various lengths from the chosen stock. The choice of a cutting pattern will then determine the amount of the lost material from the stock used. Since the stock material generally has some monetary value, a cost can be associated with any unused or wasted material (although some of this cost may be recouped via salvage). Therefore it should be clear that one would want to minimize this cost by trying to make intelligent decisions on how to cut from the stock in such a way that unused material is at a minimum, while still fulfilling the requested orders. When the cuts made are of a single dimension and the stock material used has a single standard length, the problem just described is known as the standard one-dimensional cutting stock problem.

Cutting stock problems have been dealt with in industry for many years. The first published appearance of this type of problem in literature known to this author was by

Eisemann (1957). He described the problem of suppressing trim losses in cutting rolls of various materials such as foil, paper, cellophane, and textiles. The main idea was that this problem could be formulated as a linear programming model which was solvable by existing methods.

What makes the cutting stock problem so difficult is that the number of possible combinations of cutting patterns that must be enumerated to reach an optimal answer can grow to the tens of thousands for a typical real world model. Even if all the possible cutting patterns can be formulated into the linear programming model, the fractional answers that commonly result have little practical meaning. Integer programming models were just as difficult to implement, again because of the impracticality of formulating every possible cutting pattern. Hence in the case of the cutting stock problem, fractional answers would be rounded up to integer ones, and it was accepted that the cost of overproducing some orders was negligible.

Probably the greatest advancement in solving cutting stock problems occurred in the early sixties, due to the work of P.C. Gilmore and R. E. Gomory (1961). They developed a method of solving the cutting stock problem by generating improved cutting patterns rather than searching all possible patterns for those that were most efficient. The procedure overcame the difficulty of including all possible cutting patterns into the formulation of the problem, which in turn reduced the size of the model to something that could be solved in a reasonable amount of time. While the ability to reduce very large problems that were unable to be solved in a reasonable amount of time was a great achievement, the formulation did not include the restriction of an all integer solution.

If the linear answer is used as a starting point, the possibilities of reaching an integer solution are rounding up, rounding down, or a combination of both. The topic of recent research on the cutting stock problem is finding the best way of reaching an integer solution, possibly starting from the linear solution obtained by the method developed by Gilmore and Gomory. Most of the methods described or referenced in journals of mathematics or operations research are heuristics, each claiming to perform better than the others. Scheithauer and Terno (1995) have recently developed a heuristic that they claim works well on problems that observe certain mathematical characteristics; the method used to reach integer answers is rounding up. One of the few articles that include real world problems for testing a heuristic was by Stadtler (1990), who also used a method of rounding up linear solutions to achieve integer answers. It seems apparent from the literature that rounding up linear solutions may preserve the most efficient cutting patterns that could be used (Gau 1995; Stadtler 1990; Scheithauer 1995), but this author has not found any mathematical justification of this conjecture.

The rounding procedure that must be used to get exact answers is the area of study in this thesis. A heuristic will be developed whose method will be to round down a linear solution, and then "fill out" the remaining demands. The rest of this chapter gives two traditional formulations of a small sized cutting stock problem, which should give the reader an indication of how a model can grow to sizes that are impractical to solve with traditional methods.

1.2 Example Problem

Consider a problem where the requested order is

Number of pieces needed	Length of each piece
10	4
12	5
6	10

and the stock that will be used has length 15. Given that there are as many stock pieces as needed, how should these stock pieces be cut so that the material left over is minimized? A cutting pattern is a way of cutting some integer number of pieces with length l_i from a piece of stock material with length L. The possible cutting patterns for this example are (where $L = 15$):

cut pattern no.	cut length	cut length	cut length	waste
1	4	4	4	$w_1 = 3$
2	5	5	5	$w_2 = 0$
3	4	4	5	$w_3 = 2$
4	5	5	4	$w_4 = 1$
5	4	4	0	$w_5 = 7$
6	4	5	0	$w_6 = 6$
7	5	5	0	$w_7 = 5$
8	10	4	0	$w_8 = 1$
9	10	5	0	$w_9 = 0$
10	4	0	0	$w_{10} = 11$
11	5	0	0	$w_{11} = 10$
12	10	0	0	$w_{12} = 5$

Representation of cutting patterns in vector notation can be done as follows. Each component corresponds to a number of times a different cut length (l_i) appears in that pattern. For example, the vector representation of the above cutting patterns, using $(l_1, l_2, l_3) = (4, 5, 10)$ is

cut pattern no.	vector representation
1	(3, 0, 0)
2	(0, 3, 0)
3	(2, 1, 0)
4	(1, 2, 0)
5	(2, 0, 0)
6	(1, 1, 0)
7	(0, 2, 0)
8	(1, 0, 1)
9	(0, 1, 1)
10	(1, 0, 0)
11	(0, 1, 0)
12	(0, 0, 1)

Two different ways to formulate this problem are given next.

1.3 Linear Program Formulation

The following formulation of the one dimensional cutting stock problem given above was suggested by Maurer (1996). Solutions are represented by the number of times a cutting pattern is to be used on a specific number of stock pieces. Suppose there is a request for m orders having demand d_i for order i. Let any cutting pattern j be formed by the vector

$(a_{1j}, a_{2j}, \ldots, a_{mj})$

that satisfies

$$\sum_{i=1}^{m} l_i \cdot a_{ij} \leq L, j = 1, \ldots, n \tag{1.1}$$

$a_{ij} \geq 0$ and integer,

where a_{ij} represents the number of times an order length l_i occurs in cutting pattern j and L is the stock length to cut from. If

w_o is the total waste,
w_j is the waste from cutting pattern j,
x_j is the number of times to perform cutting pattern j,
n is the total number of patterns that can be cut from L,

then the following model can be used:

$$\min w_o = \sum_{j=1}^{n} w_j \cdot x_j \tag{1.2}$$

$$\text{s.t.} \sum_{j=1}^{n} a_{ij} \cdot x_j = d_i, i = 1, \ldots, m \tag{1.3}$$

$x_j \geq 0, j = 1, \ldots, n.$

More specifically, for the example problem we obtain:

$$\begin{aligned}
\min \ w_o &= 3x_1 + 2x_3 + 1x_4 + 7x_5 + 6x_6 + 5x_7 + 1x_8 + 11x_{10} + 10x_{11} + 5x_{12} \\
\text{s.t.} \quad & 3x_1 + 2x_3 + 1x_4 + 2x_5 + 1x_6 + 1x_8 + 1x_{10} = 10, \\
& 3x_2 + 1x_3 + 2x_4 + 1x_6 + 2x_7 + 1x_9 + 1x_{11} = 12, \\
& 1x_8 + 1x_9 + 1x_{12} = 6, \\
x_j &\geq 0.
\end{aligned}$$

The answer to this problem, as calculated by the optimization package STORM, is:

$w_o = 10,$
$x_1 = 1.33,$
$x_2 = 4,$
$x_8 = 6,$

all else 0.

Note, however, that x_1 is not integer-valued in this solution.

1.4 Integer Program Formulation

The following integer programming formulation of the one dimensional cutting stock problem was presented by Gau and Wascher (1995). Solutions are comprised of vectors representing cutting patterns and a corresponding frequency for each pattern necessary to satisfy the order. Suppose there is a request for m orders having demand d_i for order I. Let any cutting pattern j be formed by the vector $(a_{1j}, a_{2j}, \ldots, a_{mj})$ that satisfies

$$\sum_{i=1}^{m} l_i \cdot a_{ij} \leq L, \tag{1.4}$$

$a_{ij} \geq 0$ and integer,

where a_{ij} represents the number of times an order length l_i occurs in cutting pattern j and L is the stock length to be cut from. If

x_o is the total number of stock lengths used,
x_j is the number of times to perform cutting pattern j,
n is the total number of cutting patterns that can be cut from L,

then an integer programming model could be written as

$$\min \ x_o = \sum_{j=1}^{n} x_j \tag{1.5}$$

$$\text{s.t.} \sum_{j=1}^{n} a_{ij} \cdot x_j = d_i \tag{1.6}$$

$x_i \geq 0$ and integer, $i = 1, \ldots, m$

Using this model for the example problem with the same cutting patterns as in the previous formulation yields

$$\begin{aligned}
\min\ x_o &= 1x_1 + 1x_2 + 1x_3 + 1x_4 + 1x_5 + 1x_6 + 1x_7 + 1x_8 + 1x_9 + 1x_{10} + 1x_{11} + 1x_{12} \\
\text{s.t.}\quad & 3x_1 + 2x_3 + 1x_4 + 2x_5 + 1x_6 + 1x_8 + 1x_{10} = 10, \\
& 3x_2 + 1x_3 + 2x_4 + 1x_6 + 2x_7 + 1x_9 + 1x_{11} = 12, \\
& 1x_8 + 1x_9 + 1x_{12} = 6, \\
& x_j \geq 0 \text{ and integer.}
\end{aligned}$$

The answer to this problem, as calculated by the optimization package STORM, is

$x_o = 12$,
$x_1 = 2$,
$x_2 = 4$,
$x_8 = 4$,
$x_{12} = 2$,
all else 0.

It is of interest to note that the objective of minimizing wasted material is equivalent to the objective of minimizing the number of stock pieces used. The following proof of this is a generalization of an example taken from Winston (1995).

Proof 1.4.1:

Assume m orders exist, each having demand d_i for some length l_i. Let x_j be the number of times a piece of stock material with length L is cut using cutting pattern j. Let w_j be the length (amount) of waste associated with cutting pattern j. Then the total demand, D, is

$$D = \sum_{i=1}^{m} d_i. \tag{1.7}$$

The total length (amount) of stock cut is

$$L \cdot \sum_{j=1}^{n} x_j. \tag{1.8}$$

The total waste resulting from the above cuts is

$$W_o = L \cdot \sum_{j=1}^{n} x_j - D. \tag{1.9}$$

If the objective is to minimize waste, then the objective function is

$$\min Z = L \cdot \sum_{j=1}^{n} x_j - D. \tag{1.10}$$

Since D is a constant, (1.10) is equivalent to

$$\min Z = L \cdot \sum_{j=1}^{n} x_j. \tag{1.11}$$

Since L is a constant, (1.11) is equivalent to

$$\min Z = \sum_{j=1}^{n} x_j. \tag{1.12}$$

Since (1.12) is the minimization of the number of stock pieces cut, it has been shown that the minimization of waste is equivalent to the minimization of the number of stock pieces used.

1.5 Summary

The one dimensional cutting stock problem arises in various types of industry. It can be formulated as either a linear or integer program, but an integer program formulation is often impractical. The work of Gilmore and Gomory made it possible to generate linear answers to the cutting stock problem in a reasonable amount of time. Much of the research done in this area has been on heuristics that round up the linear answer obtained by using the methods of Gilmore and Gomory to achieve an integer solution. Chapter two explains the Round Down Heuristic, a method of rounding down the linear programming solution, and then generating the integer solution necessary to solve the problem.

Chapter 2

GENERAL ALGORITHM

2.1 Problem statement

For the one dimensional cutting stock problem, cuts will be of the square, or guillotine variety, and the stocked material shall be limited to a single length. An order consists of a request for some integer number of lengths l, which will be cut from an unlimited supply of standard stock material having length L, as long as $l < L$. If a cost is assigned to the stock length L, then the cost of an order is the total amount of stock material cut to fill that order. The objective is to complete the order at minimum cost or, equivalently, to use the least amount of stock.

Recall the formulation presented in chapter 1.3:

$$\min w_o = \sum_{j=1}^{n} w_j \cdot x_j \tag{2.1}$$

$$\text{s.t.} \sum_{j=1}^{n} a_{ij} \cdot x_j = d_i \,, \; i = 1, \ldots, m \tag{2.2}$$

$$x_j \geq 0, \; j = 1, \ldots, n.$$

If the w_j are replaced with c_i, the cost of the stock length cut using the ith cutting pattern, and c_0 replaces w_0, representing total cost, then the cutting stock problem is formulated given the circumstances listed in the preceding paragraph. This is essentially the formulation presented by Gilmore and Gomory.

2.2 Gilmore and Gomory's method: The Basics

The technique published by Gilmore and Gomory in the early sixties is still the foundation for many algorithms used in solving the one dimensional cutting stock problem (Haessler 1992). Methods used up to that point in time were traditional linear programming models, whose solvability can become impractical when the number of variables increase to the size needed to reflect real world applications. What Gilmore and Gomory did was create a method that would generate an improved solution (i.e. cutting pattern) to the problem instead of searching over all the possible improvements for the best one, using a condensed simplex tableaux to keep track of ensuring feasibility, which turns out to be manageable computationally even when the number of decision variables is large. Unfortunately, along with overcoming the problem of having a large number of decision variables, the drawback is that it must be done under the assumption that those variables can be non-integer.

To start the method, an initial basic feasible solution must be created, as in any simplex method. Next, instead of determining which variables (if any) should enter the basis, a problem of the knapsack variety must be solved. The solution to the knapsack problem is an improved cutting pattern, one that maximizes the amount of stock used while maintaining feasibility (the cost of the stock length in Gilmore and Gomory's formulation). These improved cutting patterns are recorded and make up part of the solution to the cutting stock problem.

2.3 The Round Down Heuristic

As was noted earlier, the cutting pattern generating technique of Gilmore and Gomory does not have the restriction of an all integer solution. While this relaxation often gives non-integer solutions, the computation time for larger problems is considerably less when compared to the time used to solve the type of formulations presented in chapter 1. There are three choices one can make to arrive at an all integer answer, given a linear one has been found: either round up, round down, or a combination thereof. This has been the primary area of concentration of research lately, as can be seen from the recent journal articles focusing on the cutting stock problem. There are pros and cons for whichever method is used. For example, rounding up may preserve the most efficient cutting patterns (Stadtler 1990), but then the potential problem of having more cut lengths than required must be dealt with. For the purposes of this thesis, the method used in achieving an exact answer will be to round down and then "fill out" the order in some systematic fashion. In this section, a procedure called the Round Down Heuristic (RDH) is developed to form an exact answer from the solution generated by Gilmore and Gomory to the one dimensional cutting stock problem.

The Round Down Heuristic is a systematic way of determining which cut lengths to use in a cutting pattern based on a utilization ratio. The utilization ratio is computed by dividing the number of pieces of a given cut length that are needed by the cut length. The larger the ratio, the more pieces that can be cut from the given stock. Once these utilization ratios are computed for all cut lengths, order them from largest to smallest. Starting with the largest, a cutting pattern is generated by adding more individual cuts until there is not enough stock material to cut from. The process is continued until all of the cut lengths needed are accounted for. While the objective of traditional cutting stock problem formulations is to minimize the amount of unused stock material or the number of stock pieces used, the objective when using the RDH is the maximization of the number of pieces that can be cut from a given piece of stock material.

2.4 Applying the Round Down Heuristic: Example One

To illustrate this process, recall the example presented in chapter one. The requested orders for three different lengths of material are:

Number of pieces needed	Length of each piece
10	4
12	5
6	10

and the stock to be cut from has length 15. Solving this problem as a linear program yields the following solution:

$1.33\cdot(3,0,0)\ +$
$4.00\cdot(0,3,0)\ +$
$6.00\cdot(1,0,1)$,

where each vector component represents the frequency of length 4, 5, and 10 respectively in that cutting pattern. So the last cutting pattern is to be done six times and is made up of one piece of length 4 and one piece of length 10. Rounding down the non-integer valued frequencies gives the solution

1.00·(3,0,0) +
4.00·(0,3,0) +
6.00·(1,0,1).

The above cutting patterns and their frequencies result in twelve pieces of length 5, six pieces of length 10, but only nine pieces of length 4 - one piece short of the original order. Therefor it is necessary to cut from another piece of stock material one piece of length 4. The complete answer is given below:

1.00·(3,0,0) +
4.00·(0,3,0) +
6.00·(1,0,1) +
1.00·(1,0,0).

This solution uses 12 bars and results in 20 units of waste, which is the same as the optimal solution presented in chapter one. A more complex example is given next.

2.5 Applying the Round Down Heuristic: Example Two

Consider the following cutting stock problem. The requested orders for seven different lengths of material are:

Number of pieces needed	Length of each piece
37	102
2	88
18	60
61	58
14	57
12	48
70	39

and the length of the stock that will be used is 723.

Suppose after running the linear programming portion of the model the solution obtained is

$$
\begin{aligned}
5.91&\cdot(5, 0, 1, 0, 1, 2, 0) + \\
.33&\cdot(0, 6, 0, 0, 0, 0, 5) + \\
.13&\cdot(5, 0, 1, 0, 2, 0, 1) + \\
6.78&\cdot(1, 0, 1, 9, 0, 0, 1) + \\
.64&\cdot(0, 0, 0, 0, 12, 0, 1) + \\
.18&\cdot(2, 0, 0, 0, 0, 1, 0) + \\
3.55&\cdot(0, 0, 1, 0, 0, 0, 17),
\end{aligned}
$$

where each vector represents a cutting pattern giving the frequency of the ith length l, for $i = 1,2,\ldots,7$. So, for example, the first cutting pattern listed is to be done 5.91 times, and is comprised of five cuts of length 37, one cut of length 18, one cut of length 14, and two cuts of length 12. Using the notation presented in section 1.2,

$$
\begin{aligned}
x_1 &= 5.91, \\
x_2 &= .33, \\
x_3 &= .13, \\
x_4 &= 6.78, \\
x_5 &= .64, \\
x_6 &= .18, \\
x_7 &= 3.55.
\end{aligned}
$$

The first step in using the RDH is to round down all non-integer valued variables. Doing this to the above linear solution yields the following integer valued solution:

$$
\begin{aligned}
x_1 &= 5, \\
x_2 &= 0, \\
x_3 &= 0,
\end{aligned}
$$

$x_4 = 6,$
$x_5 = 0,$
$x_6 = 0,$
$x_7 = 3.$

The table below lists how much of each order would be completed with this solution.

Number of pieces cut	Length of each piece
31	102
0	88
14	60
54	58
5	57
10	48
57	39

At this point a statement can be made about how much of the original orders have yet to be filled. Using the cutting patterns of the above linear solution along with the rounded down integer frequencies, the remaining orders to be filled are:

Number of pieces still needed	Length of each piece
6	102
2	88
4	60
7	58
9	57
2	48
13	39

Now form a Utilization Ratio Table, as shown below.

Order Number	A Number of pieces still needed	B Length of each piece	A/B Utilization Ratio
1	6	102	$r_1 = 0.0588$
2	2	88	$r_2 = 0.0227$
3	4	60	$r_3 = 0.0667$
4	7	58	$r_4 = 0.1207$
5	9	57	$r_5 = 0.1579$
6	2	48	$r_6 = 0.0417$
7	**13**	**39**	$\mathbf{r_7 = 0.3333}$

Look for the largest ratio; in this case it is r_7. Next, determine how many pieces with length 39 can be cut from the stock of length 723, which is $\text{int}(723 \div 39) = 18$. Since the demand for pieces with length 39 is only thirteen, update the Utilization Ratio Table by setting the number of pieces still needed for order number seven equal to zero. Calculate the amount of stock that would be left over if thirteen cuts of length 39 were made, which is $723 - (13 \cdot 39) = 216$. The table now looks like:

Order Number	A Number of pieces still needed	B Length of each piece	A/B Utilization Ratio
1	6	102	$r_1 = 0.0588$
2	2	88	$r_2 = 0.0227$
3	4	60	$r_3 = 0.0667$
4	7	58	$r_4 = 0.1207$
5	9	57	$r_5 = 0.1579$
6	2	48	$r_6 = 0.0417$
7	**0**	**39**	**used**

Since 216 is greater than all of the remaining cut lengths, find the next largest utilization ratio, which is r_5. Now calculate how many pieces of length 57 can be cut from the remaining stock of length 216. This calculation reveals that three pieces of length 57 can be used, reducing the stock to length $216 - (3 \cdot 57) = 45$. Now reduce the number of pieces still needed for order five to $9 - 3 = 6$. The updated table is shown below.

Order Number	A Number of pieces still needed	B Length of each piece	A/B Utilization Ratio
1	6	102	$r_1 = 0.0588$
2	2	88	$r_2 = 0.0227$
3	4	60	$r_3 = 0.0667$
4	7	58	$r_4 = 0.1207$
5	**6**	**57**	**used**
6	2	48	$r_6 = 0.0417$
7	**0**	**39**	**used**

Since all the remaining cut lengths are greater than 45, the length of the remaining stock, the iteration is complete. The cutting pattern that has been generated can be expressed in vector form as:

$1 \cdot (0, 0, 0, 0, 3, 0, 13)$.

Since all the orders have not been filled, repeat the process by building a new Utilization Ratio Table and start with another piece of stock with length $L = 723$. The final table resulting from this iteration is given below:

Order Number	A Number of pieces still needed	B Length of each piece	A/B Utilization Ratio
1	6	102	$r_1 = 0.0588$
2	2	88	$r_2 = 0.0227$
3	4	60	$r_3 = 0.0667$
4	**0**	**58**	**used**
5	**1**	**57**	**used**
6	2	48	$r_6 = 0.0417$
7	0	39	$r_7 = 0.0$

The cutting pattern generated is:

1·(0, 0, 0, 7, 5, 0, 0).

All orders are still not completely filled, so repeat the process. The final Utilization Ratio Table resulting from this iteration is:

Order Number	A Number of pieces still needed	B Length of each piece	A/B Utilization Ratio
1	**2**	**102**	**used**
2	2	88	$r_2 = 0.0227$
3	**0**	**60**	**used**
4	0	58	$r_4 = 0.0$
5	1	57	$r_5 = 0.0175$
6	**1**	**48**	**used**
7	0	39	$r_7 = 0.0$

The cutting pattern generated from this iteration is:

1·(4, 0, 4, 0, 0, 1, 0).

The orders are still not completely filled, so repeat the process. The final Utilization Ratio Table for this iteration is shown below:

Order Number	A Number of pieces still needed	B Length of each piece	A/B Utilization Ratio
1	**0**	**102**	**used**
2	**0**	**88**	**used**
3	0	60	$r_3 = 0.0$
4	0	58	$r_4 = 0.0$
5	**0**	**57**	**used**
6	**0**	**48**	**used**
7	0	39	$r_7 = 0.0$

The cutting pattern generated is:

1·(2, 2, 0, 0, 1, 1, 0).

Since all the orders have been completely filled, the process stops. Combining the cutting patterns generated by the Round Down Heuristic with the rounded down LP solution gives the final, exact answer shown below:

5·(5, 0, 1, 0, 1, 2, 0) +
6·(1, 0, 1, 9, 0, 0, 1) +
3·(0, 0, 1, 0, 0, 0, 17) +
1·(0, 0, 0, 0, 3, 0, 13) +
1·(0, 0, 0, 7, 5, 0, 0) +
1·(4, 0, 4, 0, 0, 1, 0) +
1·(2, 2, 0, 0, 1, 1, 0).

This solution uses eighteen bars of stock material resulting in 342 units of waste.

The following list summarizes the steps used in applying the Round Down Heuristic; a flowchart for the procedure is included as Figure 2.1 on the next page.

1) Obtain the optimal linear answer to the cutting stock problem. If the answer is all integer, stop; otherwise go to step 2.

2) Round down the non-integer valued frequencies.

3) Determine the number of stock pieces needed to complete the original order, using the updated solution. If none are needed, stop; otherwise go to step 4.

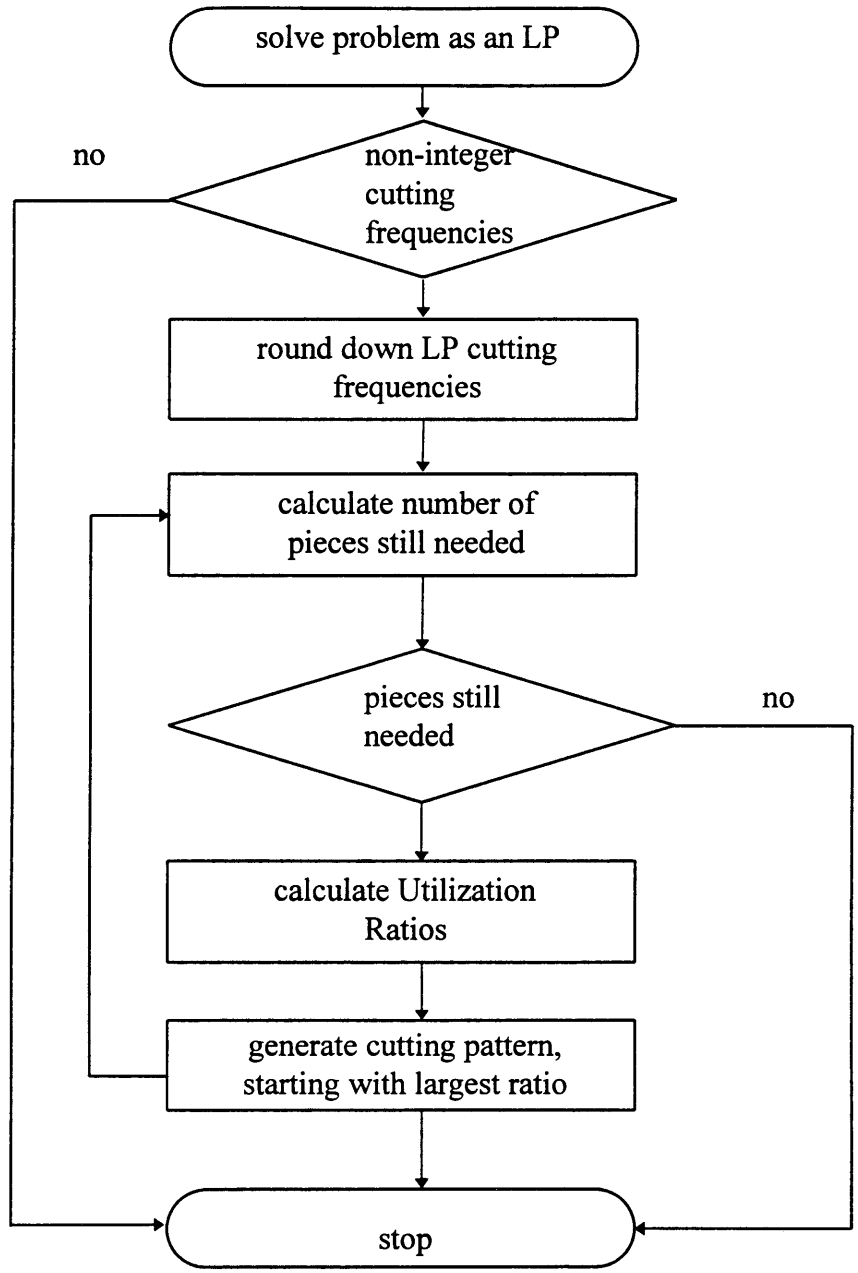
4) Form the Utilization Ratio Table. The utilization ratios are formed by dividing the number of each cut length still needed by the cut length. If the demand for a particular cut length has been met, set the corresponding utilization ratio equal to zero.

5) Starting with the largest utilization ratio, generate a cutting pattern by determining how many cuts with a length corresponding to that ratio can be made from one piece of stock.

6) Calculate how much stock would remain if the cuts created in step 5 are made. If this remaining length is longer than any of the still required cut lengths, look for the next largest utilization ratio and go to step 5. Repeat steps 5 and 6 until the remaining stock's length is less than any of the remaining cut lengths.

7) Go to step 3.

Figure 2.1 RDH Flowchart



2.6 Summary

The Round Down Heuristic is a method used to obtain an integer answer, starting from a linear answer generated by the column generation technique of Gilmore and Gomory, for the one dimensional cutting stock problem. The objective of the RDH is to maximize the number of cut lengths used in a cutting pattern. Chapter three describes a test of the RDH on some cutting stock problems. The objective of the test is to judge the effectiveness of using the RDH.

Chapter 3

ALGORITHM TEST

3.1 Test Problem Background

The problems used in the testing of the Round Down Heuristic have been compiled from more than one source. Problems one through six make up the data set for which RDH was created. They are real world problems taken from an aluminum services company. Problems seven through twelve are taken from Pierce (1964). Some of these problems originated from real world problems, but were altered so as to foster empirical study. Problem number thirteen is from Woolsey (1996). It was used for testing during the development of the RDH, and is the example problem given in chapter two.

Unfortunately, the only problems in which the optimal solution (expressed as the minimum number of stock pieces used) was known were those from Pierce. Therefore, these were the problems used to judge the effectiveness of the RDH.

The BASIC program used to solve the problems is capable of taking into account material used in the cutting process (kerf) and any unusable material on each piece of stock (endtrim). The time it took to solve the problems was not used in judging the effectiveness of the RDH for one reason: the two computer codes solving the same data set could not be run on the same computer. So the measure of effectiveness of the RDH was based on the number of stock lengths used in the final answer, compared with the optimal solution of the problem as given by Pierce.

3.2 Results

The selection of problems was not random, and therefore no inference about their representation of typical real world problems should be made. Although the RDH performed quite well on the problems from Pierce, there really is no guarantee that it will perform this well on others. Another criterion of effectiveness is the amount of waste created by the generated cutting patterns. This author's experience has shown most problems are solved with a waste percentage between two and fifteen percent. In the first four problems from Pierce, the RDH arrived at the optimal answer. Solutions to the fifth and sixth problems were within at least 99% of optimality. For the rest of the test problems, only the waste percentage is given as an indication of effectiveness. The following table summarizes the test problem results. Complete problem listings with solutions can be found in appendix A.

Table 3.1 RDH Test Results

Problem No. (Name)	No. of Orders	Optimal answer (no. of bars)	RDH answer (no. of bars)	Percent of Optimality
7 (7)	7	640	640	100 %
8 (7 RQ)	7	245	245	100 %
9 (7-10)	7	493	493	100 %
10 (7-10 RQ)	7	197	197	100 %
11 (10)	10	239	240	99.6 %
12 (20-20)	20	1000	1001	99.9 %
		waste percent**		
1	7	17 %	4	*
2	5	12.43 %	9	*
3	3	19.36 %	5	*
4	17	4.18 %	22	*
5	18	3.65 %	25	*
6	18	5.09 %	27	*
13 (Woolsey)	7	2.63 %	18	*

* Optimal answer not known

** Computed as total length of waste / total length of stock material used

3.3 A Different Approach

Since the optimal answers to problems one through six and problem thirteen were unknown, it seemed reasonable that the solutions obtained by applying the RDH may be improved by exploring other possible ways of solving those problems. Recall from chapter two the LP solution to example two:

$$
\begin{aligned}
&5.91\cdot(5, 0, 1, 0, 1, 2, 0) + \\
&.33\cdot(0, 6, 0, 0, 0, 0, 5) + \\
&.13\cdot(5, 0, 1, 0, 2, 0, 1) + \\
&6.78\cdot(1, 0, 1, 9, 0, 0, 1) + \\
&.64\cdot(0, 0, 0, 0, 12, 0, 1) + \\
&.18\cdot(2, 0, 0, 0, 0, 1, 0) + \\
&3.55\cdot(0, 0, 1, 0, 0, 0, 17).
\end{aligned}
$$

Notice that three of the above cutting pattern frequencies are greater than one. Another direction that might be researched is the following question. What would happen if the original demands were adjusted to reflect the remaining cuts needed after the rounded down solution is obtained, and then this problem was solved again using the method of Gilmore and Gomory? Is there a point where the LP solution will have all cutting pattern frequencies less than one? Another question that might be asked is whether every non-integer valued frequency be rounded down after each LP solution, given that the above procedure is used to solve the problem. The first two questions posed above were explored in an attempt to improve the solutions obtained for problems one through six and problem thirteen.

The following procedure was employed to try to improve solutions of the aforementioned problems. First the LP solution was obtained. If all the non-integer valued cutting frequencies were less than one, the RDH was applied and a final solution given. If all non-integer valued cutting frequencies were not less than one, each frequency was rounded down and the number of pieces cut resulting from this solution was subtracted from the original demands. The resulting problem was then solved using the column generating technique once again. This procedure was repeated until all resulting cutting frequencies given by the LP solution were less than one.

Doing the above procedure resulted in an improved solution to problem number two. The details of problem two are given below.

Number of pieces needed	Length of each piece
6	938
10	1238
6	1278
6	1388
8	1638

Stock length used: 6000

The vector representation of a solution obtained using the RDH is

$1.00 \cdot (1, 4, 0, 0, 0) +$
$4.00 \cdot (0, 0, 1, 1, 2) +$
$1.00 \cdot (5, 1, 0, 0, 0) +$
$1.00 \cdot (0, 4, 0, 0, 0) +$
$1.00 \cdot (0, 0, 2, 2, 0) +$
$1.00 \cdot (0, 1, 0, 0, 0),$

which uses a total of nine pieces of stock

The vector representation of a solution obtained by reapplying the LP to rounded down answers is

$1.00 \cdot (1, 4, 0, 0, 0) +$
$4.00 \cdot (0, 0, 1, 1, 2) +$
$1.00 \cdot (1, 3, 1, 0, 0) +$
$1.00 \cdot (2, 0, 1, 2, 0) +$
$1.00 \cdot (2, 3, 0, 0, 0),$

which uses a total of eight pieces of stock, a savings of one piece of stock. The third LP solution obtained in solving this problem yielded all cutting frequencies less than one. At first glance a reduction of one piece of stock material may not seem like much of an improvement, but as the price of stock increases so does the savings.

3.4 Exploring the RDH Solution

One might wonder if when using the RDH, solutions may have some type of predictable quality such as the number of cutting patterns generated, waste distribution, and so on. Another consideration is if there is a predictable pattern in the RDH solution, is this pattern somehow related to characteristics of the original problem. These questions are addressed and answered in this section.

Consider the number of different and exact (no waste) cutting patterns generated in applying the RDH to the following problems (a complete listing can be found in appendix A).

Problem No.	No. of Orders	No. of Different Cutting Patterns	No. of Exact Cutting Patterns Used (no waste)
1	7	4	0
2	5	6	0
3	3	3	0
4	17	14	0
5	18	19	0
6	18	18	0
7	7	11	0
8	7	9	0
9	7	10	2
10	7	7	0
11	10	17	3
12	20	29	17
13	7	6	3

It appears from the above table that the number of different cutting patterns generated by using the RDH is problem dependent. Recall that there may be a number of different optimal solutions to any one cutting stock problem, and each of these may generate a different number of cutting patterns. If changing cutting patterns is a costly procedure, some investigation into solutions with a minimum number of cutting patterns should be done. Does the RDH generate a minimum number of cutting patterns, a maximum, or somewhere in the middle? To be certain of a minimum or maximum number of cutting patterns, all solutions would have to be known, which is most likely an impractical task at the least. This author conjectures the RDH generates a number of cutting patterns greater than the minimum and less than the maximum in most instances. To illustrate the previous statement, six additional optimal solutions to problem thirteen were computed and the number of cutting patterns recorded, along with other statistics. Results of this experiment are shown below.

Problem 13 Solution Name	No. of Cutting Patterns	No. of Exact Cutting Patterns Used (no waste)	Solution's Average Length of Waste (as a % of stock)	Solution's Longest Piece of Waste (as a % of stock)
A	5	4	3 %	10 %
B	7	1	3 %	5 %
C	9	5	4 %	30 %
D	6	1	3 %	10 %
E	4	0	3 %	17 %
F	7	0	3 %	18 %
RDH	6	15	16 %	41 %

The smallest number of cutting patterns used to solve problem thirteen in the above solution set is four, from solution E. While not proven, experience compels the author to believe this is the minimum number of different cutting patterns possible. The largest number of patterns used in this solution set is nine, from solution C, while the RDH solution results in six different cutting patterns. Notice the RDH solution uses fifteen cutting patterns (there are three different patterns) that result in no waste. A characteristic of using the RDH is that the LP portion of the heuristic will always find the most efficient cutting patterns. Whether or not the LP says to do these patterns an integer number of times is problem dependent, however.

Consider the waste distribution of the solutions generated by using the RDH; more specifically look at the average length of a piece of waste. Could this length be expected to be larger or smaller based on the average order length or the standard deviation of the order lengths? The following table lists these statistics for the problems in this thesis. The solution's expected length of a piece of waste is calculated by the statistical definition of expected value and is expressed as a percentage of the length of stock used since this is different in most problems.

Problem No.	Solution's Expected Length of Waste (as a % of stock)	Standard Dev of Order Lengths	Average Order Length
1	18 %	163	659
2	13 %	226	1309
3	17 %	229	1833
4	5 %	165	709
5	5 %	175	519
6	5 %	373	1188
7	3 %	16	47
8	3 %	14	51
9	1 %	16	37
10	3 %	14	41
11	2 %	7	38
12	4 %	14	46
13	16 %	22	59

There does not appear to be any correlation between the expected length of a piece of waste and the standard deviation of the order lengths or the average order length, as suggested by the above table. This author conjectures that the relation between these statistics is problem dependent. One characteristic of the waste distribution of the RDH solutions that seems to be consistent is that there is one relatively long piece of waste generated, as suggested by the following table.

Problem No.	Solution's Expected Length of Waste (as a % of stock)	Solution's Longest Piece of Waste (as a % of stock)
1	18 %	60 %
2	13 %	80 %
3	17 %	48 %
4	5 %	46 %
5	5 %	34 %
6	5 %	56 %
7	3 %	13 %
8	3 %	18 %
9	1 %	52 %
10	3 %	52 %
11	2 %	63 %
12	4 %	53 %
13	16 %	41 %

A conclusion that can be drawn from taking a closer look at the RDH solutions is that their characteristics (as defined in this section) are problem dependent, except for the consistent generation of one relatively long piece of waste.

3.5 Conclusions

The test problems used cannot be said to be representative of the problems encountered in real world practice. The utilization ratios used in the RDH are a way of formalizing an intuitive approach to solving the problem. They allow clear decisions, without ambiguity, to be made in potentially hazy instances. Problems one through six are from the aluminum industry, and gave the incentive for developing the RDH. In one instance, the solution obtained by the RDH was improved by a process of reapplying the column generating technique until all cutting frequencies were less than one, and then applying the RDH. While the author was hopeful the solutions to these problems would contribute another measure of effectiveness of the RDH, a response has not yet been made by the provider of those problems.

An attempt was made to try to discern if there were any predictable qualities of the solutions generated when applying the RDH. Solution characteristics such as the number of different cutting patterns and the waste distribution could play an important role in the overall cost effectiveness of the final answer. The only consistent characteristic of the RDH solutions is that there is usually one relatively long piece of waste generated. This may or may not be a desirable quality - it depends on the environment in which the solution will be used.

3.6 Suggestions for Further Study

It was pointed out in chapter one that using a linear programming model to solve the one dimensional cutting stock problem often generates cutting patterns that are to be done a fractional number of times. To overcome this, non-integer valued frequencies need to be rounded to an integer in some fashion as to fulfill the requested orders. While many recent heuristics suggest rounding up, this thesis developed a method of rounding down. It seems reasonable to conjecture a more efficient answer may be obtained by using a combination of rounding up and down the non-integer valued frequencies.

The Round Down Heuristic was developed to work only for stock material of a single length. To be adaptable to more real world situations, the RDH will need to be capable of working with multiple stock lengths. Recall that the objective of the RDH is to use as many cut lengths as possible per piece of stock. If a choice of different stock lengths exists, this is effectively asking the heuristic to choose the length that uses the most cut lengths. If there should be a tie, the stock length used should result in the minimum waste.

Another area for study would be to perform a comparison test of new and existing cutting stock problem solving techniques (including the RDH) on a bench mark set of problems. The problem set could be generated using CUTGEN1 (Gau and Wascher), a problem generating procedure that uses various random number generators. For strict comparison purposes, this may be more ideal than using a compilation of test problems drawn from various sources.

Formulating the model to account for external information is another area of research. Note that the definition of optimal in this thesis is the minimum number of bars used to fulfill an order. Experience shows that in many cases different sets of cutting patterns will yield the optimal answer to the same problem. Different sets of cutting patterns result in the total amount of waste being distributed differently. This variable distribution of waste may affect the usefulness of the solution. For example, consider a solution that results in 100 units of waste distributed as ten bars with length 10 versus a solution that distributes the same 100 units of waste as two bars with length 50. If the two bars of length 50 could be re-cut and sold for profit but ten bars of length 10 could not be reused (maybe sold for scrap, at best), the obvious decision is to use the cutting patterns that yield waste in the form of two bars with length 50.

Taking the above example a step further suppose there exists more than one cutting stock problem that needs to be solved at any given time, each requiring a different length of stock to cut from. Suppose that different cut lengths have variable costs that are not linearly related as well. The most general formulation of Gilmore and Gomory's column generating technique is capable of considering these additional constraints, but the current RDH is not. An adaptation of the RDH to this knapsack type of problem could prove very effective in obtaining integer answers.

REFERENCES CITED

Eisemann, K. 1957. The trim problem. Management Science 3: 279-284.

Gau, T., and Wascher, G. 1995 CUTGEN1: A problem generator for the Standard one-dimensional cutting stock problem. European Journal of Operational Research 84: 572-579.

Gilmore, P.C., and Gomory, R.E. 1961. A linear programming approach to the cutting stock problem. Operations Research 9: 849-859.

Gilmore, P.C., and Gomory, R.E. 1964. Multistage cutting stock problems of two and more dimensions. Operations Research 7: 863-888.

Haessler, R.W. 1992. One-dimensional cutting stock problems and solution procedures. Mathematical and Computer Modeling 16 (1): 1-8.

Maurer, R., 1996. Private Communication. Colorado School of Mines, Golden, Colorado.

Pierce, Jr., J.F., 1964. Some large-scale production scheduling problems in the paper industry. Prentice-Hall, Englewood Cliffs, NJ.

Stadtler, H. 1990 A one-dimensional cutting stock problem in the aluminum industry and its solution. European Journal of Operational Research 44: 209-223.

Scheithauer, G., and Terno, J. 1995 The modified integer round-up property of the one-dimensional cutting stock problem. European Journal of Operational Research 84 562-571.

Winston, W. L. 1995 Introduction to Mathematical Programming Indiana University: Duxbury Press.

Woolsey, R.E.D., 1996. Private Communication. Colorado School of Mines, Golden, Colorado.

Appendix A

Test Problems

Problem 1

Stock length used: 6000

Cut length	Amount needed
463	10
613	4
638	2
713	2
818	4
823	6
888	2

Solution (Problem no. 1)

<u>What the numbers mean</u>: Each line gives the stock length to cut from, a cutting pattern, and how many times to do the pattern. For example, the first line says from a piece of stock with length 6000, cut two pieces of length 613, one piece of length 638, four pieces of length 818, one piece of length 823, and do this pattern one time.

6000 2 613 1 638 4 818 1 823 1 time(s)
6000 10 463 1 823
6000 2 613 2 713 4 823
6000 1 638 2 888

waste percentage: 17.00
number of bars used: 4

Problem 2

Stock length used: 6000

Cut length	Amount needed
938	6
1238	10
1278	6
1388	6
1638	8

Solution (Problem no. 2)*

6000	1 938 4 1238	1 time(s)
6000	1 1278 1 1388 2 1638	4 time(s)
6000	5 938 1 1238	
6000	4 1238	
6000	2 1278 2 1388	
6000	1 1238	

waste percentage: 12.43
number of bars used: 9
* See page 39 for solution format explanation

Problem 3

Stock length used: 5000

Cut length	Amount needed
1569	2
1669	4
2069	5

Solution (Problem no. 3)*

5000 1 1569 2 1669 2 time(s)
5000 2 2069 2 time(s)
5000 1 2069

waste percentage: 19.36
number of bars used: 5
* See page 39 for solution format explanation

Problem 4

Stock length to be used: 4750

Cut length	Amount needed
398	8
482	8
490	8
548	4
573	2
582	20
648	4
682	2
731	2
753	4
774	42
822	5
823	2
873	12
882	1
962	12
974	4

Solution (Problem no. 4)*

4750	5	482	1	548	2	873			1 time(s)
4750	8	582							2 time(s)
4750	1	398	2	648	4	753			1 time(s)
4750	6	774							6 time(s)
4750	4	398	1	490	3	873			1 time(s)
4750	1	822	4	962					3 time(s)
4750	1	822	4	974					1 time(s)
4750	1	398	7	490	1	873			
4750	6	774							
4750	5	873							
4750	3	482	1	548	4	582			
4750	2	398	2	548	2	573	2	648	
4750	2	682	2	731	2	823			
4750	1	822	1	873	1	888			

waste percentage: 4.18
number of bars used: 22
* See page 39 for solution format explanation

Problem 5

Stock length used: 4750

Cut length	Amount needed
62	12
362	18
369	6
370	14
378	4
385	12
387	18
564	4
565	40
569	10
570	6
635	2
654	26
655	4
662	8
669	2
738	12
761	20

Solution (Problem no. 5)*

4750	7	362	1	370	1	378	1	669	1	738	1 time(s)
4750	12	387									1 time(s)
4750	1	387	1	635	5	738					1 time(s)
4750	8	565									2 time(s)
4750	8	569									1 time(s)
4750	3	62	12	370							1 time(s)
4750	7	654									3 time(s)
4750	7	662									1 time(s)
4750	2	62	8	565							2 time(s)
4750	6	761									3 time(s)
4750	5	62	11	385							
4750	11	362	1	369							
4750	8	565									
4750	5	369	5	387	1	570					
4750	1	378	5	570	2	738					
4750	2	564	5	654							
4750	1	378	4	655	2	738					
4750	1	370	1	378	2	564	2	569	2	738	
4750	1	385	1	635	1	662	1	669	2	761	

waste percentage: 3.65
number of bars used: 25
* See page 39 for solution format explanation

Problem 6

Stock length used: 4750

Cut length	Amount needed
762	8
770	6
777	6
794	4
871	2
919	6
954	2
977	6
1153	10
1154	6
1171	8
1178	4
1193	6
1394	4
1571	6
1653	4
1678	4
1962	10

Solution (Problem no. 6)*

4750 6 770	1 time(s)
4750 6 777	1 time(s)
4750 1 794 2 1962	4 time(s)
4750 5 919	1 time(s)
4750 1 762 4 977	1 time(s)
4750 4 1153	2 time(s)
4750 4 1154	1 time(s)
4750 4 1171	2 time(s)
4750 1 762 1 1178 2 1394	2 time(s)
4750 3 1571	2 time(s)
4750 1 1193 2 1653	2 time(s)
4750 1 1193 2 1678	2 time(s)
4750 1 762 2 1962	1 time(s)
4750 4 762 1 871	
4750 2 954 2 977	
4750 2 1153 2 1154	
4750 1 871 2 1178 1 1193	
4750 1 919 1 1193	

waste percentage 5.09
number of bars used: 27
* See page 39 for solution format explanation

Problem 7 (Pierce no. 7)

Stock length used: 215

Cut length	Amount needed
64	782
60	624
48	142
45	118
33	144
32	826
16	188

Solution (Pierce no. 7)*

215	2	64	1	48	1	32		9 time(s)
215	3	60	1	32				203 time(s)
215	1	48	5	33				28 time(s)
215	1	64	1	60	2	45		12 time(s)
215	2	64	1	45	1	32		93 time(s)
215	1	48	5	32				103 time(s)
215	3	64	1	16				188 time(s)
215	6	32						
215	1	60	4	33				
215	1	60	2	48	1	45		
215	2	64	1	60				

waste percentage: 2.76
number of bars used: 640
* See page 39 for solution format explanation

Problem 8 (Pierce no. 7 RQ)

Stock length used: 215

Cut length	Amount needed
64	12
60	624
48	142
45	8
33	144
32	6
16	88

Solution (Pierce no. 7-RQ)*

215	1	64	3	48			12 time(s)
215	1	60	3	48			29 time(s)
215	4	48	1	16			2 time(s)
215	2	60	1	48	1	45	8 time(s)
215	3	60	1	33			144 time(s)
215	3	60	1	32			6 time(s)
215	3	60	2	16			42 time(s)
215	3	48	2	16			
215	3	60					

waste percentage: 1.78
number of bars used: 245
* See page 39 for solution format explanation

Problem 9 (Pierce no. 7-10)

Stock length used: 215

Cut length	Amount needed
54	782
50	624
38	142
35	118
23	144
22	826
6	188

Solution (Pierce no. 7-10)*

215	3	54	1	50					40 time(s)
215	1	54	1	50	5	22			135 time(s)
215	1	54	2	50	1	38	1	23	141 time(s)
215	2	54	1	50	1	35	1	22	118 time(s)
215	3	54	2	23	1	6			1 time(s)
215	3	54	1	22	5	6			32 time(s)
215	2	54	2	50	1	6			23 time(s)
215	3	54	1	22	4	6			
215	3	50	1	38	1	23			
215	2	54							

waste percentage: 0.40
number of bars used: 493
* See page 39 for solution format explanation

Problem 10 (Pierce no. 7-10 RQ)

Stock length used: 215

Cut length	Amount needed
54	12
50	624
38	142
35	8
23	144
22	6
6	88

Solution (Pierce no. 7-10 RQ)*

215	3	54	1	50					4 time(s)
215	4	50							2 time(s)
215	3	50	1	35	1	23	1	6	8 time(s)
215	3	50	1	38	1	23			136 time(s)
215	3	50	1	38	1	22			6 time(s)
215	4	50	2	6					40 time(s)
215	2	50							

waste percentage: 2.03
number of bars used: 197
* See page 39 for solution format explanation

Problem 11 (Pierce no. 10)

Stock length used: 148

Cut length	Amount needed
55	60
49	30
47	70
43	80
42	50
40	50
39	180
35	60
32	80
30	260

Solution (Pierce no. 10)*

148	1	55	1	47	1	42				4 time(s)
148	2	42	2	30						3 time(s)
148	1	47	1	39	1	32	1	30		15 time(s)
148	1	55	1	49	1	43				29 time(s)
148	1	47	1	40	2	30				50 time(s)
148	1	55	1	32	2	30				25 time(s)
148	3	39	1	30						54 time(s)
148	1	43	3	35						20 time(s)
148	2	42	2	32						19 time(s)
148	2	43	2	30						14 time(s)
148	4	30								
148	1	39	3	30						
148	3	43								
148	2	39	2	32						
148	1	55	2	42						
148	1	49	1	47						
148	1	55								

waste percentage: 1.24
number of bars used: 240
* See page 39 for solution format explanation

Problem 12 (Pierce no. 20-20)

Stock length used: 150

Cut length	Amount needed
76	150
71	20
70	30
69	170
64	80
60	420
55	50
53	210
52	40
49	30
47	100
46	100
45	110
43	270
41	30
40	70
39	310
35	280
30	480
29	320

Solution (Pierce no. 20-20)*

150	1	76	1	39	1	35	150 time(s)
150	1	71	1	49	1	30	20 time(s)
150	1	70	1	45	1	35	27 time(s)
150	1	69	1	41	1	40	30 time(s)
150	1	64	2	43			70 time(s)
150	2	55	1	40			25 time(s)
150	1	43	2	39	1	29	4 time(s)
150	1	64	1	47	1	39	9 time(s)
150	1	69	2	40			4 time(s)
150	1	52	2	49			4 time(s)
150	2	53	1	43			34 time(s)
150	1	69	1	46	1	35	100 time(s)
150	1	60	2	45			40 time(s)
150	2	60	1	30			144 time(s)
150	1	69	1	52	1	29	35 time(s)
150	1	70	2	40			2 time(s)
150	1	60	1	47	1	43	90 time(s)
150	5	30					63 time(s)
150	1	53	1	39	2	291	40 time(s)
150	1	39	3	35			
150	3	40	1	29			
150	3	45					
150	1	43	2	39			
150	2	49	1	30			
150	2	53	1	43			
150	2	60					
150	1	52	1	47			
150	1	69	1	64			
150	1	70					

waste percentage: 0.19
number of bars used: 1001
* See page 39 for solution format explanation

Problem 13 (Woolsey)

Stock length used: 723

Cut length	Amount needed
102	37
88	2
60	18
58	61
57	14
48	12
39	70

Solution (Problem no. 13)*

723	5	102	1	60	1	57	2	48	5 time(s)
723	9	60	1	57	1	48	2	39	1 time(s)
723	1	102	6	58	7	39			9 time(s)
723	1	58	8	57	5	39			
723	1	102	4	60	6	58			
723	2	102	2	88	1	48			

waste percentage: 2.63
number of bars used: 18
* See page 39 for solution format explanation

Appendix B

Program Pseudocode

1) Input stock length data

2) Input cut length data

3) Set up initial tableaux for LP

4) Identify knapsack problem to be solved

5) Try to solve knapsack problem with ad-hoc method

    5.1) If a solution is found, record as an activity and go to step 4

    5.2) If no solution is found, try to solve with dynamic programming

        5.2.1) If a solution is found, record as an activity and go to step 4

        5.2.2) If no solution is found, current LP solution is optimal

6) Identify if the RDH needs to be applied

    6.1) If the RDH is not needed, problem is solved; go to step 8

7) Apply the RDH, generate a new cutting pattern and record it; go to step 6

8) Output the problem solution

Appendix C

Program Source Code

```
REM This program is a BASIC version of the method developed by Gilmore and
REM Gomory.  It is written for the standard one dimensional cutting stock problem.
REM An integer solution is generated from the lp solution using the Round Down
REM Heuristic.

REM Inputs are from data strings in this program.  Outputs are written to a file
REM designated by the user, before running.

REM Author:          Andy Crouter
REM                  Colorado School of Mines, Golden, CO
REM Programmed by: Andy Crouter, January 1996.

REM Variable List
REM a(m + 1)                    solution storage for ad-hoc method
REM act(m + 1, m + 50)          array of cutting pattern frequencies
REM addactivity(m + 1)          RDH iteration counter
REM b(m + 1, m + 1)             initial set-up array
REM bb(m + 1, 2)                array used in bang for buck calculation
REM Binv(m + 1, m + 1)          inverse of array b
REM cut(m + 1, 3)               cut lengths and required demand
REM k(m + 1, 2)                 Ad-hoc method computation array
REM lg(m)                       DP computation array
REM n(m + 1, 1)                 LP computation array
REM newcons1(m + 1)             RDH array
REM newcons2(m + 1)             RDH array
REM newobj(m + 1)               RDH array
REM Nprime(m + 1, 1)            LP solution array
REM p(m + 1, 1)                 LP computation array
REM pivotcolumn(m + 1, 1)       LP computation array
REM q(m)                        DP computation array
REM r(m + 1, 2)                 DP computation array
REM s(m, 475)                   DP computation array
REM stock(3, 3)                 stock length data array
REM temp1(m + 1)                temp array
```

```
REM temp2(m + 1)            temp array
REM u(m + 1)                DP computation array
REM x(m, 475)               DP computation array
REM etrim                   end trim on stock
REM kerf                    saw kerf
REM ufactor                 scaling factor for lengths


CLS
OPEN "a:e50243.txt" FOR OUTPUT AS #1
     m = 18
     kerf = 5
     ufactor = 10
     etrim = 0
     DIM a(m + 1)
     DIM act(m + 1, m + 50)
     DIM addactivity(m + 1)
     DIM b(m + 1, m + 1)
     DIM bb(m + 1, 2)
     DIM Binv(m + 1, m + 1)
     DIM cut(m + 1, 3)
     DIM k(m + 1, 2)
     DIM lg(m)
     DIM n(m + 1, 1)
     DIM newcons1(m + 1)
     DIM newcons2(m + 1)
     DIM newobj(m + 1)
     DIM Nprime(m + 1, 1)
     DIM p(m + 1, 1)
     DIM pivotcolumn(m + 1, 1)
     DIM q(m)
     DIM r(m + 1, 2)
     DIM s(m, 475)
     DIM stock(3, 3)
     DIM temp1(m + 1)
     DIM temp2(m + 1)
     DIM u(m + 1)
     DIM x(m, 475)
```

```
REM *** input stock data
    FOR i = 1 TO 1
        FOR j = 1 TO 3
            READ stock(i, j)
        NEXT j
    NEXT i
    DATA 1,475,475


REM *** sort stock by ascending length
REM *** scount = number of different stock lengths
scount = 1
100   flag = 0
    FOR i = 1 TO scount - 1
        IF stock(i, 2) < stock(i + 1, 2) THEN GOTO 101
        SWAP stock(i, 1), stock(i + 1, 1)
        SWAP stock(i, 2), stock(i + 1, 2)
        SWAP stock(i, 3), stock(i + 1, 3)
        flag = 1
101   NEXT i
    IF flag = 1 THEN 100

REM *** determine shortest stock length
REM *** shortest is a 1x2 matrix [index,length]
    shortest(1, 1) = 1
    shortest(1, 2) = 100000
    FOR i = 1 TO scount
        IF stock(i, 2) >= shortest(1, 2) THEN GOTO 400
        shortest(1, 2) = stock(i, 2)
        shortest(1, 1) = stock(i, 1)
400   NEXT i


REM *** input cut data
    FOR i = 1 TO m
        FOR j = 1 TO 3
            READ cut(i, j)
        NEXT j
    NEXT i
```

```
REM e5024
     DATA 1,6.7,12,2,36.7,18,3,37.4,6,4,37.5,14,5,38.3,4
     DATA 6,39.0,12,7,39.2,18,8,56.9,4,9,57.0,40,10,57.4,10,11,57.5,6,12,64.0,2
     DATA 13,65.9,26,14,66.0,4,15,66.7,8,16,67.4,2,17,74.3,12,18,76.6,20


REM *** determine longest cut length
REM *** longest is a 1x2 matrix [index,length]
REM *** m= number of different cut lengths

     longest(1, 1) = 0
     longest(1, 2) = 0
     FOR i = 1 TO m
          IF cut(i, 2) <= longest(1, 2) THEN GOTO 401
          longest(1, 2) = cut(i, 2)
          longest(1, 1) = cut(i, 1)
401    NEXT i


REM *** generate initial B, Binv, N, and Nprime matrices
REM *** assert: shortest stock length > all cut lengths

REM *** initialize B

     FOR i = 1 TO m + 1
          FOR j = 1 TO m + 1
               b(i, j) = 0
          NEXT j
     NEXT i

REM *** set B(1,1) = 1 and B(1,j) = -cost of shortest stock
     b(1, 1) = 1
     FOR i = 2 TO m + 1
          b(1, i) = -stock(shortest(1, 1), 3)
     NEXT i


REM *** set a[i,i]'s
IF shortest(1, 2) < longest(1, 2) THEN GOTO 50
     FOR i = 2 TO m + 1
```

```
            b(i, i) = INT(stock(shortest(1, 1), 2) / cut(i - 1, 2))
        NEXT i
        GOTO 51

50      STOP
        flag = 0
        FOR i = 2 TO m + 1
            FOR j = 1 TO scount
                IF stock(j, 2) <= cut(i, 2) THEN GOTO 10
                flag = 1
                b(i, i) = INT(stock(j, 2) / cut(i - 1, 2))
                GOTO 20
10          NEXT j
        IF flag <> 1 THEN STOP: GOTO 9990
20      NEXT i


REM *** set activity matrix first row with shortest stock length
REM *** assumption : shortest stock length > longest cut length

51      FOR i = 1 TO m
            act(1, i) = stock(1, 2)
        NEXT i
        FOR i = 2 TO m + 1
            FOR j = 1 TO m
                act(i, j) = b(i, j + 1)
            NEXT j
        NEXT i


REM *** initialize Binv

        FOR i = 1 TO m + 1
            FOR j = 1 TO m + 1
                Binv(i, j) = 0
            NEXT j
        NEXT i

REM *** set initial Binv
        Binv(1, 1) = 1
```

```
    FOR j = 2 TO m + 1
        Binv(1, j) = stock(shortest(1, 1), 2) / b(j, j)
    NEXT j
    FOR i = 2 TO m + 1
        Binv(i, i) = 1 / b(i, i)
    NEXT i


REM *** set Nprime

    Nprime(1, 1) = 0
    FOR i = 2 TO m + 1
        Nprime(i, 1) = cut(i - 1, 3)
    NEXT i


REM *** calculate N=Binv x Nprime

    FOR i = 1 TO m + 1
        n(i, 1) = 0
        FOR j = 1 TO m + 1
            n(i, 1) = n(i, 1) + Binv(i, j) * Nprime(j, 1)
        NEXT j
    NEXT i


REM *** Ad-hoc method for finding solutions to pairs of constraints
REM *** determine constraint ratios r=[index, Binv/cut]

1000  FOR i = 1 TO m
        r(i, 2) = 0
    NEXT i
    FOR i = 1 TO m
        r(i, 1) = i
        r(i, 2) = Binv(1, i + 1) / cut(i, 2)
    NEXT i


REM *** bubble sort ratios in descending order
200   flag = 0
```

```
    FOR i = 1 TO m - 1
        IF r(i, 2) >= r(i + 1, 2) THEN GOTO 201
        SWAP r(i, 1), r(i + 1, 1)
        SWAP r(i, 2), r(i + 1, 2)
        flag = 1
201   NEXT i
    IF flag = 1 THEN 200


REM *** determine subscripts (i's)

    FOR k = 1 TO m
        u(k) = r(k, 1)
    NEXT k


REM *** determine ad-hoc solution, if other than (0,0,...,0)
REM *** this version works for m cut lengths

    flag = 0
    a(u(1)) = INT(stock(1, 2) / cut(u(1), 2))
    IF a(u(1)) <> 0 THEN flag = 1
    temp = -a(u(1)) * cut(u(1), 2)
    FOR l = 2 TO m
        a(u(l)) = INT((stock(1, 2) + temp) / cut(u(l), 2))
        temp = temp - cut(u(l), 2) * a(u(l))
        IF a(u(l)) <> 0 THEN flag = 1
    NEXT l
    IF flag = 1 THEN GOTO 301

300   GOTO 8000

301   cost1 = 0
    payoff1 = stock(1, 2)
    FOR k = 1 TO m
        cost1 = cost1 + cut(k, 2) * a(k)
    NEXT k
    IF payoff1 - cost1 < 0 THEN GOTO 300
    cost2 = 0
    payoff2 = stock(1, 3)
```

```
    FOR k = 1 TO m
        cost2 = cost2 + Binv(1, k + 1) * a(k)
    NEXT k
    IF cost2 - payoff2 <= 0 THEN GOTO 300
    IF ABS(cost2 - payoff) < .0001 THEN GOTO 300
    length = stock(1, 2)
REM *** form P matrix
305
    p(1, 1) = -stock(1, 3)
    FOR x = 2 TO m + 1
        p(x, 1) = a(x - 1)
    NEXT x
REM *** determine Binv*P, initailizing pivotcolumn first

    FOR x = 1 TO m + 1
        pivotcolumn(x, 1) = 0
    NEXT x
    FOR x = 1 TO m + 1
        FOR y = 1 TO m + 1
            pivotcolumn(x, 1) = pivotcolumn(x, 1) + Binv(x, y) * p(y, 1)
        NEXT y
        IF pivotcolumn(1, 1) <= 0 THEN GOTO 9998
    NEXT x
    FOR x = 2 TO m + 1
        IF pivotcolumn(x, 1) = 0 THEN 320
320  NEXT x


REM *** determine the pivotrow of pivotcolumn
    tcount = 1
    FOR t = 2 TO m + 1
        IF n(t, 1) <= 0 THEN 501
        IF pivotcolumn(t, 1) <= 0 THEN 501
        k(tcount, 1) = n(t, 1) / pivotcolumn(t, 1)
        k(tcount, 2) = t
        tcount = tcount + 1
501    NEXT t

502    flag = 0
    FOR t = 1 TO tcount - 2
```

```
    IF k(t, 1) <= k(t + 1, 1) THEN GOTO 503
        SWAP k(t, 1), k(t + 1, 1)
        SWAP k(t, 2), k(t + 1, 2)
        flag = 1
503   NEXT t
    IF flag = 1 THEN 502
    mink = k(1, 1)
    pivotrow = k(1, 2)
    IF mink = 0 THEN STOP


REM *** update activities matrix (B) using pivotrow as an index
    act(1, pivotrow - 1) = length
    FOR x = 2 TO m + 1
        act(x, pivotrow - 1) = a(x - 1)
    NEXT x


REM *** determine Gprime by performing gaussian elimination
REM *** normalize pivot row and element
    FOR y = 2 TO m + 1
        Binv(pivotrow, y) = Binv(pivotrow, y) / pivotcolumn(pivotrow, 1)
    NEXT y
    n(pivotrow, 1) = n(pivotrow, 1) / pivotcolumn(pivotrow, 1)
    pivotcolumn(pivotrow, 1) = 1


REM *** Binv
    FOR x = 1 TO m + 1
        FOR y = 2 TO m + 1
            IF x = pivotrow THEN GOTO 600
            Binv(x, y) = Binv(x, y) - pivotcolumn(x, 1) * Binv(pivotrow, y)
        NEXT y
600   NEXT x


REM *** N
    FOR x = 1 TO m + 1
        IF x = pivotrow THEN GOTO 601
        n(x, 1) = n(x, 1) - pivotcolumn(x, 1) * n(pivotrow, 1)
```

```
601    NEXT x


REM *** pivotcolumn (Binv x P)
     FOR x = 1 TO m + 1
          IF x = pivotrow THEN GOTO 602
          pivotcolumn(x, 1) = pivotcolumn(x, 1) - pivotcolumn(x, 1) *
pivotcolumn(pivotrow, 1)
602    NEXT x

GOTO 1000


REM *** output routine for activities
REM *** info is put in a file called
1100   FOR i = 1 TO m
     IF INT(n(i + 1, 1)) = 0 THEN 1160
     PRINT #1, USING "####"; etrim + ufactor * act(1, i); : PRINT #1, " ";

     FOR j = 2 TO m + 1
     IF act(j, i) = 0 THEN 1150
     PRINT #1, USING "###"; act(j, i); : PRINT #1, " ";
     PRINT #1, USING "####"; ufactor * cut(j - 1, 2) - kerf;
1150   NEXT j
     PRINT #1, USING "###"; INT(n(i + 1, 1)); : PRINT #1, " time(s)"
1160   NEXT i
     FOR i = m + 1 TO (m + 1) + (acount - 2)
     PRINT #1, USING "####"; ufactor * stock(1, 2) + etrim; : PRINT #1, " ";
     FOR j = 2 TO m + 1
     IF act(j, i) = 0 THEN 1180
     PRINT #1, USING "###"; act(j, i); : PRINT #1, " ";
     PRINT #1, USING "####"; ufactor * cut(j - 1, 2) - kerf;
1180   NEXT j
     PRINT #1,
     NEXT i


REM *** wastage computation routine
2     stockused = 0
     FOR i = 1 TO m
```

```
        stockused = stockused + INT(n(i + 1, 1)) * act(1, i)
    NEXT i
    stockused = stockused + (acount - 1) * stock(1, 3)
    PRINT #1, "amount used"; stockused
    amountcut = 0
    FOR i = 1 TO m
        amountcut = amountcut + cut(i, 2) * cut(i, 3)
    NEXT i
    PRINT #1, "amount cut"; amountcut
    PRINT #1, "waste percentage ";
    PRINT #1, USING "##.##"; 100 - (amountcut / stockused) * 100
    PRINT #1, "acount ="; acount - 1
    CLOSE #1


END
9990   PRINT "degeneracy has occurred"
END


9998
REM *** set newcons1 to the number of pieces still needed

    FOR i = 1 TO m
        newcons1(i) = cut(i, 3)
    NEXT i
    FOR l = 1 TO m
        FOR j = 1 TO m
            temp1(j) = INT(n(l + 1, 1)) * act(j + 1, l)
        NEXT j


        FOR w = 1 TO m
            temp2(w) = temp2(w) + temp1(w)
        NEXT w
    NEXT l
        FOR k = 1 TO m
            newcons1(k) = newcons1(k) - temp2(k)
        NEXT k
```

```
REM *** check to see if any more pieces are needed
    flag = 0
    FOR i = 1 TO m
        IF newcons1(i) <> 0 THEN flag = 1
    NEXT i
    IF flag = 0 THEN GOTO 1100
    FOR i = 1 TO m
        IF newcons1(i) = 0 THEN newobj(i) = 0 ELSE newobj(i) = 1
    NEXT i


REM *** set newcons2 coefficients to the different cut lengths
    FOR i = 1 TO m
        newcons2(i) = cut(i, 2)
    NEXT i



1190  mtemp = m
    remainder = stock(1, 2)
    GOSUB 8190
1200  IF mtemp = 0 THEN 1210
    largestbb(1, 1) = bb(mtemp, 1): largestbb(1, 2) = bb(mtemp, 2)
    GOSUB 8210
    mtemp = mtemp - 1
    GOTO 1200

1210  GOSUB 8220
    GOSUB 8230
    IF done$ = "false" THEN 1190

    GOTO 1100

REM Dynamic Programming Subroutine
REM Solves problems of the form: max := sum{c(j) * x(j)}
REM                   s.t.  sum{a(j) * x(j)} <= rhs
REM                   where 1<=j<=m


rhs = stock(1, 2)
```

```
     FOR i = 1 TO m
          q(i) = Binv(1, i + 1)
     NEXT i
     FOR i = 1 TO m
          lg(i) = cut(i, 2)
     NEXT i

     FOR z = 0 TO rhs
          s(1, z) = q(1) * INT(z / lg(1))
          x(1, z) = INT(z / lg(1))
     NEXT z
     FOR y = 2 TO m
     FOR z = 0 TO rhs
          big = 0
          FOR j = 0 TO INT(z / lg(y))
               whch = 0
               IF (z - lg(y) * j) > 0 THEN whch = z - lg(y) * j
               try = q(y) * j + s(y - 1, whch)
               IF try < big THEN GOTO 8005
               s(y, z) = try
               big = try
               x(y, z) = j
8005      NEXT j
     NEXT z
     NEXT y

     temp = 0
     a(m) = x(m, rhs)
     FOR y = m - 1 TO 1 STEP -1
          temp = temp + lg(y + 1) * a(y + 1)
          a(y) = x(y, rhs - temp)
          IF a(y) < 0 THEN STOP
     NEXT y

8010 cost1 = 0
     payoff1 = stock(1, 2)
     FOR k = 1 TO m
          cost1 = cost1 + cut(k, 2) * a(k)
     NEXT k
     IF payoff1 - cost1 < 0 THEN 9998
```

```
    cost2 = 0
    payoff2 = stock(1, 3)
    FOR k = 1 TO m
        cost2 = cost2 + Binv(1, k + 1) * a(k)
    NEXT k
    IF cost2 - payoff2 <= 0 THEN 9998
    IF ABS(cost2 - payoff) < .0001 THEN 9998
    length = stock(1, 2)
    GOTO 305


REM *** Bang for buck calculation subroutine
REM *** bb ratio's are sorted from largest to smallest
8190  FOR i = 1 TO m
        bb(i, 1) = i
        bb(i, 2) = newcons1(i) / newcons2(i)
    NEXT i

8195  flag = 0
    FOR i = 1 TO mtemp - 1
        IF bb(i, 2) <= bb(i + 1, 2) THEN GOTO 8196
        SWAP bb(i, 1), bb(i + 1, 1)
        SWAP bb(i, 2), bb(i + 1, 2)
        flag = 1
8196  NEXT i
    IF flag = 1 THEN 8195
    RETURN


REM *** Find largest amount of resource that can be used and still
REM *** be feasible subroutine.  If flag = 1, then an improvement has
REM *** been found
8210  flag = 0
    temp = newcons1(largestbb(1, 1))
8212  IF temp * newcons2(largestbb(1, 1)) <= remainder THEN 8214
        temp = temp - 1
        GOTO 8212
    flag = 1
8214  most = temp
```

```
    addactivity(largestbb(1, 1)) = most
    remainder = remainder - temp * newcons2(largestbb(1, 1))
    RETURN


REM *** update newcons1 and record new activity
8220   FOR i = 1 TO m
          newcons1(i) = newcons1(i) - addactivity(i)
    NEXT i
    FOR i = 1 TO m
          act(i + 1, m + acount) = addactivity(i)
    NEXT i
    FOR i = 1 TO m
          addactivity(i) = 0
    NEXT i
    act(1, m + acount) = stock(1, 2)
    acount = acount + 1
    RETURN

REM *** check to see if newcons1(i) = 0; if yes, then problem has been
REM *** solved!
8230   done$ = "true"
    FOR i = 1 TO m
          IF newcons1(i) <> 0 THEN done$ = "false"
    NEXT i
    RETURN
```