ALGORITHMS FOR TWO - DIMENSIONAL CUTTING PROBLEMS

Eleni A. Hadjiconstantinou
BSc. in Statistics and Computer Science

Thesis submitted for the degree of
Doctor of Philosophy
of the University of London
and for the
Diploma of Imperial College

July 1987

Imperial College of Science and Technology
(University of London)
Department of Management Science

To my parents

ABSTRACT

The constrained two-dimensional cutting problem is the problem of cutting a number of small rectangular pieces from a single stock rectangle. Each piece has a given size, value and an upper bound on the number of that type of piece that is required. The objective is to maximise the value of the pieces cut (or minimise the waste).

In this thesis a literature survey of various cutting stock problems is given. Two versions of these problems are discussed, namely guillotine and general cutting problems.

(i) Guillotine cutting problems: This type of problem restricts the cuts to be made from one edge of the rectangle to the opposite edge, parallel to the two remaining edges. A solution method is described based on a tree-search algorithm with the search limited by the use of an upper bound obtained from the state-space relaxation of a dynamic programming recursion for the original problem. An interactive system using computer graphics and capable of solving guillotine cutting problems manually is also described. Computational results are given for the exact algorithm and for the manual experiments enabling comparisons to be made for a number of medium - size problems.

(ii) General cutting problems: This is the problem of finding optimal cutting patterns of rectangles which are not restricted to be of the " guillotine " type. Bounds based on linear programming relaxations of some mixed integer and zero - one integer programming formulations of the problem are given. An exact tree-search algorithm is described that uses a bound derived from a Lagrangean relaxation of the problem with the Lagrangean problem being solved by a subgradient optimisation procedure. The algorithm is shown to be capable of optimally solving small to medium - size problems and extensive computational results are presented.

ACKNOWLEDGEMENTS

I have been fortunate in having received the assistance of a number of people during the preparation of this thesis and it gives me pleasure to acknowledge this.

My greatest indebtedness is to my supervisor, Professor Nicos Christofides who provided me with sound advice, valuable criticism and guidance and maintained a constant interest throughout the course of this research.

I am obliged to Professor Eilon, for the opportunity to carry out research in his department.

I welcome the opportunity to record my thanks to the Science Research Council for the financial support provided for this work.

The contribution of my colleagues of the Management Science Research Unit is also greatly appreciated. Their friendship, helpful discussions and encouragement made my task a lot easier.

Last but not least, I would like to express my sincere gratitude to my family for the endless support and encouragement they have given me. It is no exaggeration to say that without their help this thesis would not have been written.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION
- A SURVEY OF CUTTING STOCK PROBLEMS (CSP's) -

1.1 Introduction

Many materials used in industry and construction come in the form of whole units (sheets of glass, tin-plate, wood, paper, roofing and sheet iron, logs, boards, beams, reinforcing rods, forms, etc.). In using them directly or for making semi-finished products, it is necessary to divide these units into parts of the required dimensions. In doing this, scrap is usually formed and the materials actually utilised constitute only a certain per cent of the whole quantity - the rest going into scrap.

The above problem can be stated in a more general form as the problem of cutting a set S_0 of one (or more) dimensional stock objects into a set S_p of smaller, one (or more) dimensional pieces of specified dimensions, in such a way as to minimise wastage or maximise the value of the pieces cut. This has become

known as the trim loss or cutting stock problem.

The Cutting Stock Problem (CSP) firstly appeared in the Operational Research Literature in 1957, when Eisemann considered the one-dimensional problem of minimising trim loss when slitting rolls of material for the supplying of customer orders. Much of Eisemann's material had however already been anticipated by *Kantorovich* [1960] in his paper presented in Russian at Leningrad State University in 1939. To our knowledge, Kantorovich was the first to formulate and propose a solution to the CSP; he considered a number of problems concerned with the minimisation of scrap and formulated them mathematically in a form later to be known as integer linear programs.

Since 1957, there has been a growing interest in cutting stock problems. This interest is motivated by the fact that the problem can be used to model a variety of problems in the real world. Specific applications are related to :

- steel bars (*Eilon* [1960], *Tilanus and Gerhardt* [1976], *Stainton* [1977])
- paper industry (*Pierce* [1964], *Marconi* [1971])
- glass industry (*Dyson and Gregory* [1974], *Chambers and Dyson* [1976])
- garment making (*Art* [1966])
- time tabling
- multi-program (batch) scheduling (*Short* [1973])
-scheduling commercial television advertising spots (*Brown* [1971]).

This interest is also a consequence of the appearance of the high speed digital computer and the development of such computer oriented optimisation techniques as linear and dynamic programming. This is especially true for those applications involving only a limited class of shape types, such as rectangles. In the

area under consideration, there are relatively few algorithmic methods available for providing exact solutions (i.e.methods that are guaranteed to produce optimal results); they basically involve dynamic programming (e.g. *Gilmore and Gomory* [1966], *Beasley* [1985]) and tree search techniques (e.g. *Christofides and Whitlock* [1977], *Greenberg and Hegerich* [1970], *Herz* [1972]). An algorithm may not be available, or the computational cost of using the best available algorithm may be prohibitive. In general, a heuristic method is highly "domain dependent", that is, it uses information about the particular problem for which it is developed in order to find good solutions. This is reflected in a variety of papers (e.g. *Pierce* [1964], *Paull and Walter* [1955], *Metzger* [1958], *Paull* [1956]) dealing with practical problems. A survey of approaches to the problem can be found in *Hinxman* [1980] and *Golden* [1976].

In this chapter, we first describe the nature of the CSP in terms of computational complexity. We then go on to describe and classify the various types of the problem. Some of the methods available for their solution are briefly surveyed. The two-dimensional CSP is examined in greater detail ; new exact solution methods for this problem are developed and described in the following chapters of the thesis.

1.2 The Computational Complexity of the CSP

Recent results in complexity theory (*Karp* [1972], *Cook* [1971]) indicate that many combinatorial optimisation problems are indeed very difficult to solve in the sense that a prohibitive amount of computation is required to construct optimal solutions for all but very small cases. If there exists an algorithm for a given

problem whose rate of computational increase with problem size is bounded above by some polynomial function of problem size, then this particular problem belongs to a class of problems denoted by P (for polynomial). Another class of problems, called the NP (for non-deterministic polynomial) class, includes all the problems in P but in addition contains problems for which the only known optimal algorithms (tree-search procedures) exhibit a rate of increase with problem size that is exponential. *Cook* [1971] showed that there exists a subclass of problems in the NP class which have the property that all the problems in the NP class can be transformed by an efficient (polynomially bounded) algorithm into any of the problems in this subclass. Problems in this subclass are called NP-complete problems.

The class NP includes an enormous number of practical problems that occur in business and industry (*Garey and Johnson* [1979]). It has been proved theoretically (*Garey and Johnson* [1979]) that the CSP is NP-complete. A proof that an NP problem is NP-complete is a proof that the problem is not in P (does not have a deterministic polynomial time algorithm) unless every NP problem is in P. Cook showed that if any member of the NP-complete class can be solved in time bounded by a polynomial in the size of the input, they all have such a polynomial time solution.

With the current state of knowledge about NP-complete problems the only optimal algorithms encountered in the literature are of the tree-search type with attention focused upon the problem of generating efficient bounds that are of good quality. In this thesis, we will concentrate on the design of such algorithms in an attempt to develop exact solution methods for the CSP, capable of dealing computationally with practical sized problems.

1.3 Terminology

In this section, we describe the terminology to be used in this chapter. CSP's have been categorised by dimension. Thus a one-dimensional problem is one in which only one dimension of the stock and order pieces is significant to the solution whilst a two-dimensional problem is one in which the stock is held as rectangular sheets and the customer requirement is for rectangles of smaller dimensions.

In a one-dimensional problem, there is a set S_0 of n stock objects in total which are referred to as "lengths", and their dimensions are L_i, i = 1, ..., n. The term " lengths " also refers to the items produced by cutting up the stock into a given set S_p of required " pieces ". The total number of pieces in S_p is m and their dimensions are l_j, j = 1, ..., m. The value of each piece to be cut is v_j, j = 1, ..., m (these values may or may not be proportional to the dimensions of the pieces). There may be an upper bound on the number of lengths to be cut denoted by Q_j, j = 1, ..., m.

The two-dimensional case involves a total number of n stock objects (set S_0), referred to as "rectangles" of width W_i and length L_i, i = 1, ..., n. An order is received for a set S_p of m smaller rectangular pieces of width w_j and length l_j, j = 1, ..., m. The value of each piece is v_j, j = 1, ..., m and the upper bound associated with each piece is Q_j, j = 1, ..., m.

A "cutting pattern" represents a set of instructions for dividing up a stock object. It is generated in the following way : If a_j is the number of piece j included in a particular cutting arrangement for a stock item, then a "combination" is defined

as any set of a_j, j = 1, ..., m such that the set of pieces represented by the a_j can be fitted within the stock item. The arrangement of a combination within the stock object is termed a "pattern" and is denoted by a =[a_1, a_2, ..., a_J] (J ≤ m). For most combinations there will be a number of patterns as illustrated in Figure 1.1. Assuming that not all pieces of S_p can be cut from S_0 the value of a pattern a =[a_1, a_2, ..., a_J] is given by:

$$\sum_{j=1}^{J} v_j a_j.$$

The terminology used when discussing CSP's must be extended when cutting restrictions are added to the standard problem. Considering the two-dimensional problem, it may be the case that the cutting process consists of a number of distinct steps in each of which a number of parallel cuts are made in rectangles, resulting from the previous step, at right angles to the cuts in that previous step. This is called "staged" two-dimensional cutting. Figure 1.2 illustrates four-stage cutting.

If there is no restriction on the number of cuts that can be made to obtain the required pieces, but only a restriction that any cut should be parallel to a side of the stock object, the problem is referred to as " orthogonal " two-dimensional. With materials such as glass there is a restriction that any cut made must be a " guillotine " cut, that is, it must extend the full width of the rectangle produced by previous cuts. Guillotine and non-guillotine orthogonal cutting are illustrated in Figure 1.3.

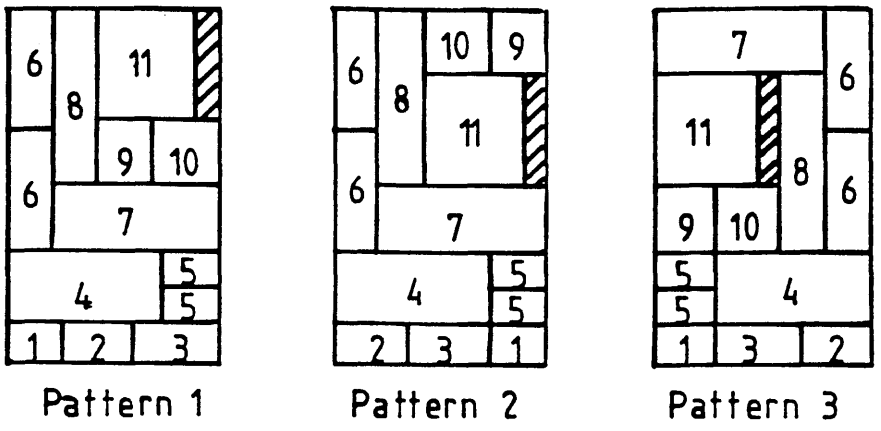


Figure 1.1 Three Different Cutting Patterns for a Combination of Eleven Pieces.

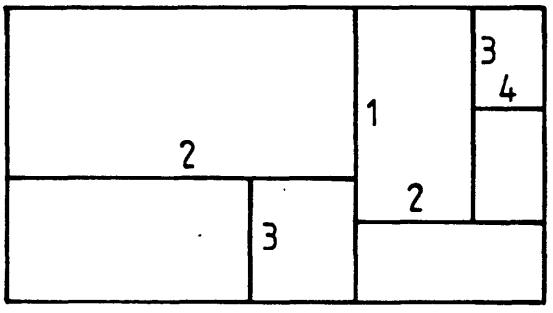


Figure 1.2 Four - stage cutting.

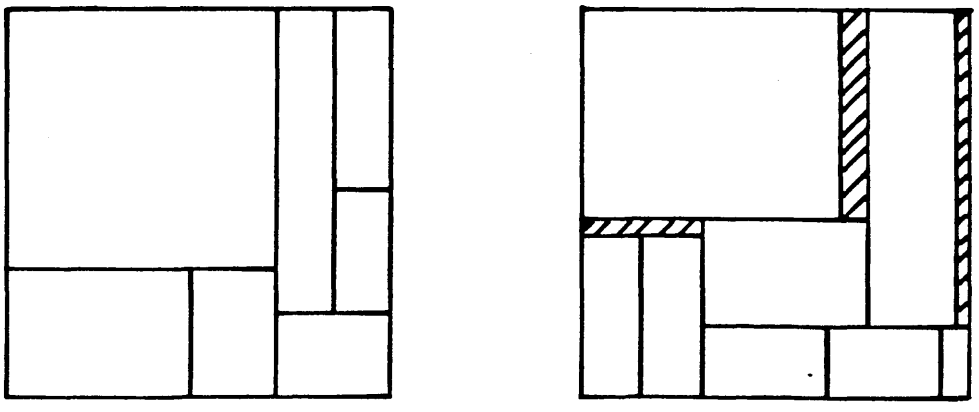


Figure 1.3 Guillotine and General Cutting Patterns.

1.4 A Classification of CSP's

CSP's essentially consist of finding the best way of cutting a set S_0 of stock objects into a set S_p of smaller pieces of specified dimensions (order sizes). Such problems can be classified in terms of a set of parameters such that each set of states for these parameters uniquely describes a particular CSP.

1.4.1 The Dimension Parameter

We can differentiate
- one-dimensional and
- two-dimensional CSP's.

Typical examples for one-dimensional problems are to be found where stocks of bars or rolls have to be cut into smaller pieces of the same cross section, there only the length of the material is relevant to the solution of the CSP.

Two-dimensional problems exist in situations where flat material (e.g. metal sheets, chipboard, panes of glass, textiles) have to be divided into smaller pieces.

One and two - dimensional problems have been considered in the literature. However, the greater the dimension the more complex the problem. Very little work has been done on two - dimensional problems (*Gilmore and Gomory* [1965]).

1.4.2 The Stock Size Parameter

In normal CSP's the size of the stock to be cut is known. However, in some cases, only a subset of object sizes are stocked because of storage or manufacturing limitations, economies of scale in production or storage and because of the costs associated with holding different sizes in stock. The objective then is to select the optimum number and size of the stock objects from some restricted set so as to minimise production (stockholding and wastage) costs. This is known in the literature as the " assortment problem ".

Furthermore, there is a two-dimensional case where stock comes in the form of a number of continuous lengths of material of known width. The objective then is to minimise the length of material required to meet customer demands. This problem belongs to the class of problems known in the literature as " bin packing " problems.

1.4.3 The Stock Constraint Parameter

In certain types of CSP's, the various stock sizes are considered to be available in unlimited supply and it is desired to find the optimal cutting pattern that will at least satisfy demand. These are known in the literature as " trim-loss problems ".

Another type of CSP is the " knapsack problem ". In this case, not all order pieces can be cut from a stock object (the stock to be cut is considered to be a limited resource). Hence the objective is to maximise the total value of pieces produced in a way that at most satisfies demand - this is known as the constrained

knapsack problem. It is also possible to formulate problems of the knapsack type in which there is no demand and it is simply required to find the optimum cutting pattern for a stock object (unconstrained knapsack problem).

1.5 Literature Survey

Using the classification parameters that have been described in the previous sections, we now define some of the CSP's that exist in the literature. For each type of problem, we will provide a comprehensive survey of the available solution approaches ; these can either be exact or heuristic. Very often, the methods developed for the CSP, rely upon the technological characteristics of the situation being modelled.

The survey will describe in greater detail the existing approaches to two-dimensional problems. Bin packing problems, which are considered to be special cases of CSP's, will be dealt with separately, as a result of the particular type of analysis applied to the performance of the algorithms developed for these problems.

1.5.1 The One-Dimensional Knapsack Problem

This is the simplest form of CSP. In such a problem each of the order lengths l_j, $j = 1, ..., m$ is given a value v_j and the objective is to maximise the total value of items cut from one stock length L. The problem can be formulated by defining a_j as the number of lengths of size j cut from L as:

$$\text{maximise } z = \sum_{j=1}^{m} v_j a_j$$

subject to the condition that $[a_1, a_2, ..., a_m]$ corresponds to a feasible cutting pattern.

The above problem has been studied by a great many authors and it is of considerable importance to mathematical programming. In particular, it is closely related to trim-loss problems.

Gilmore and Gomory [1966] develop the idea of the one-dimensional knapsack function from lengths $l_1, ..., l_m$ of given values $v_1, ..., v_m$ by the equation

$$F(x) = \max \{ v_1 x_1 + ... + v_m x_m ; x_j \text{ non-negative integers and}$$

$$l_1 x_1 + ... + l_m x_m \le x ; x \text{ non-negative integer } \}.$$

Clearly, if a function $F_0(x)$ is defined such that

$$F_0(x) = \max \{ 0, v_j \mid l_j \le x \}$$

then $F(x)$ satisfies

$$F(x) = \max \{ F_0(x), F(x_1) + F(x_2) \mid x \ge x_1 + x_2,\ 0 < x_1 \le x_2 \}.$$

From this definition of F(x), Gilmore and Gomory have developed a dynamic programming formulation for solving the one-dimensional knapsack problem. Their

method suffers from the main disadvantage of dynamic programming i.e that for large scale programs, a large amount of storage is required. Thus, much better tree-search algorithms have been developed since, but will not be discussed here since they are not generalisable to two or more dimensions. A comprehensive survey on the problem is given by *Salkin and Dekluyver* [1975] and *Martello and Toth* [1979].

A variation on the knapsack problem has been referred to in the literature as the multidimensional 0-1 knapsack problem. This term has been used to describe a knapsack problem in which there is more than one set of constraints (see, for example, *Weingartner and Ness* [1967]). To avoid confusion, knapsack problems concerned with the design of cutting patterns in two-dimensional Euclidean space will be referred to, in this thesis, as two-dimensional knapsack problems ; problems where the dimensionality refers to the vector of parameters for each piece, will be referred to as multiparameter knapsack problems. The main method that has been used to solve this problem is branch and bound (*Weingartner and Ness* [1967], *Ghare and Walters* [1968]).

1.5.2 The Constrained One-Dimensional knapsack Problem

This problem is formulated in a way similar to the unconstrained version with the addition of specified demands on the maximum number of each type of piece that is to be produced. That is, it may be required that $a_j \leq Q_j$, $j = 1, ..., m$ for some set of integers $\{Q_1, ..., Q_m\}$. A variety of approaches - similar to the unconstrained case - exist for this problem (*Gilmore and Gomory* [1966], *Martello and Toth* [1979]).

1.5.3 The One-Dimensional Trim Problem

Kantorovich [1960] produced a mathematical formulation for this problem in 1939 but this was not published in English for another twenty-one years. Meanwhile, mostly approximate methods had been developed, including linear programming together with rounding up or down to provide an integer solution and various other heuristics.

The simple form of the basic formulation (see *Eilon* [1960], *Metzger* [1958], *Paull* [1956], *Vajda* [1958]) is given below:

An unlimited number of standard lengths L_i, i = 1, ..., n of some material is held in stock from which lengths are to be cut to fill orders. An order consists of a request for some number Q_j of pieces of length l_j, j = 1, ..., m. For each stock length there will be a large number of combinations - say N - (Section 1.3) for cutting the required lengths. Define a_{jk} as the number of pieces of length l_j to be cut by the kth combination or cutting pattern and x_k as the number of times the kth combination is used. Also associate with each combination k a cost c_k representing the cost of the stock length that the combination uses. Then the general one-dimensional trim problem is given by

$$\text{minimise } z = \sum_{k=1}^{N} c_k x_k$$

subject to

$$\sum_{k=1}^{N} a_{jk} x_k \geq Q_j, \qquad j = 1, ..., m$$

$$x_k \geq 0, \qquad x_k \text{ integer}, \quad k = 1, ..., N.$$

It is noted that there are two factors contributing to make this integer linear program impractical for real size problems. Firstly, the number of variables (cutting patterns) is enormous so that it is impossible to obtain an integer solution and secondly the practicality of using the normal simplex method, for solving the corresponding large linear program (LP) in order to obtain bounds for tree-search methods of solution is limited.

A significant contribution to this problem was due to *Gilmore and Gomory* [1961,1963]. In their method, an initial basic feasible solution is found by an ad hoc method, then the LP is set up using as variables only the cutting patterns that occur in the initial solution. In order to find a new pattern that will improve the solution, a useful column is generated by solving an auxiliary problem at each pivot of the simplex method. The auxiliary problem to be solved is the unconstrained one-dimensional knapsack problem, described in section 1.5.1. Gilmore and Gomory also observed that towards the end of computation considerable time was required to achieve small improvements. For this reason, a heuristic method was introduced by terminating the LP if a certain number of consecutive pivots do not produce at least a certain percentage improvement in the optimal solution.

The algorithm of Gilmore and Gomory has been applied successfully to a broad class of cutting problems. However, it is only appropriate in cases in which minimising the trim loss is the only objective. There are many practical problems which are too large to be solved by this method or have a special structure due to

additional conditions with respect to production. These additional restrictions may involve using only a few distinct cutting patterns (*Haessler* [1971], *Coverdale and Wharton* [1976]), limiting production from a cutting machine, using only a fixed number of cutting knives etc. As a result, a number of authors have developed heuristic methods to the problem.

Eisemann [1957] considered the one-dimensional CSP for the case in which two cutting machines are in use. *Pierce* [1964,1966], studying problems in the paper industry, was the first author to decide that heuristic methods must be adopted; he considered the economic balance between exact procedures (which tend to be computationally expensive) and heuristic ones (which are usually relatively cheap). *Haessler* [1971,1975], also examining problems associated with the paper industry developed a similar heuristic approach; his solution procedure deals with the more commonly occuring case in which there is a fixed charge in changing the slitting patterns, thus reducing the total number of cutting patterns used. *Coverdale and Wharton* [1976] presented an improved heuristic pattern enumeration technique for solving Haessler 's trim problem. The problem involving the cutting of steel reinforcement bars was investigated by *Stainton* [1977]; in this case a heuristic solution was developed considering the possible utilisation of wastage. Finally, heuristics which use the cutting pattern that is best in terms of an evaluation (this evaluation procedure assigns heuristically determined penalties to patterns that can only be used a small number of times) are employed by *Marconi* [1971] in paper cutting problems. Value heuristics have also been used by *Tilanus and Gerhardt* [1976] in a problem arising when steel slabs have to be cut as they are manufactured.

1.5.4 The Two-Dimensional Knapsack Problem

The unconstrained problem in which the value of the rectangular pieces cut from a single plane rectangular stock object is maximised, without limits being placed on the number of pieces of each type used, is termed by *Gilmore and Gomory* [1965, 1966] as the "two-dimensional knapsack problem". The related problem of minimising the amount of waste produced by the cutting can be converted into this problem by making the value of all pieces equal to their areas.

Various methods of solution for this problem have been proposed in the literature, each with different assumptions on the allowable cutting patterns. In almost all the methods used, it has been assumed that the cuts made on the stock rectangle can only lie in the two orthogonal directions parallel to its edges, resulting in an "orthogonal" pattern. Otherwise, a cutting pattern is called "non-orthogonal". It is interesting to remark that the restriction to orthogonal cutting patterns can prevent the optimal arrangement of the required rectangles in the stock object from being obtained, as it was pointed out by *Erdös and Graham* [1975] and by *DeCani* [1978]. However, there do not seem to be any computational studies on the use of non-orthogonal cutting. An example of non-orthogonal cutting is shown in Figure 1.4.

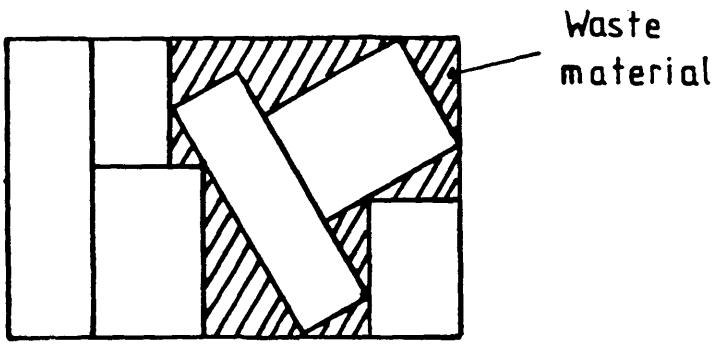


Figure 1.4 Example of a non-orthogonal cutting pattern.

A special case of the two-dimensional cutting problem restricts any cuts made to be of guillotine type (Section 1.3) - Gilmore and Gomory observe that this restriction occurs very often in practice, for example in the cutting of paper or glass. All of the CSP 's that yield to this treatment are ones in which the cutting is done in " stages " (Section 1.3); if the rectangle to be cut is being processed along a production line, there may be a number of cutting machines along the line each corresponding to a stage.

The generalised two-dimensional problem, which is related to cutting problems of non-guillotine type, has been considered by few authors in the literature. A number of heuristic procedures have been developed for the well known bin-packing (Section 1.5.7) and pallet loading (Section 1.5.5) problems which are special cases of the non-guillotine cutting problem.

In this section, the problem to be discussed will be the two-dimensional knapsack problem with guillotine cuts imposed.

There have been two optimal methods developed to solve this problem, using dynamic programming (*Gilmore and Gomory* [1966], *Beasley* [1985a]) and tree-search techniques (*Herz* [1972]).

In the paper by Gilmore and Gomory, two methods are described. Since the second method has been shown to be incorrect (*Hertz* [1972]) the first one is presented below:

Gilmore and Gomory introduce a knapsack function F (x, y) which is defined as follows:
Given m rectangular pieces (l_j, w_j), [with non-negative values v_j associated with

them], and which are required to be cut from a single stock rectangle (L, W), then

$$F(x, y) = \max \sum_{j=1}^{m} \xi_j \upsilon_j.$$

In this equation, $\xi_1, ..., \xi_m$ are non-negative integers such that there exists a way of cutting any rectangle (x, y) ($x \le L$ and $y \le W$) into ξ_j rectangles (l_j, w_j) for j = 1, ..., m using only guillotine cuts. The function F (x, y) has the following properties:

(i) $F(x, y) \ge 0$

(ii) $F(x_1 + x_2, y) \ge F(x_1, y) + F(x_2, y)$

$F(x, y_1 + y_2) \ge F(x, y_1) + F(x, y_2)$

(iii) $F(l_j, w_j) \ge \upsilon_j$, $j = 1, ..., m$

A dynamic program based on the above properties can be developed for calculating F (x, y). The recursive formula used is given below:

$$F(x, y) = \max \{ F_0(x, y), F(x_1, y) + F(x_2, y), F(x, y_1) + F(x, y_2);$$

$$x \ge x_1 + x_2, 0 < x_1 \le x_2, y \ge y_1 + y_2 \text{ and } 0 < y_1 \le y_2 \}$$

where

$$F_0(x, y) = \max_j \{ 0, \upsilon_j \mid l_j \le x \text{ and } w_j \le y, \ j = 1, ..., m \}.$$

Essentially, this formula defines the value of a rectangle (x, y) at any point by considering it as a whole, as two pieces created by making a vertical cut or as two pieces created by a horizontal cut. The result for the demanded rectangle is built up

iteratively from the value of smaller rectangles. The algorithm requires two functions l (x, y) and w (x, y) to record how the value F (x, y) is achieved (i.e. to give the associated cutting pattern). These functions are defined as follows:

$$l(x, y) = \min \{ x_1, x \mid 0 < x_1 \leq x - x_1, F(x, y) = F(x_1, y) + F(x - x_1, y) \}$$

$$w(x, y) = \min \{ y_1, y \mid 0 < y_1 \leq y - y_1, F(x, y) = F(x, y_1) + F(x, y - y_1) \}$$

Once the value of the best cutting pattern has been determined, the process backtracks to determine the actual cutting positions using a special binary tree structure.

Herz [1972] presents a recursive procedure for the same problem, but reduces the number of patterns which need to be considered. He achieves this by restricting potential cutting positions to those which are an integral combination of piece lengths or widths from a given edge. He refers to these as canonical dissections. Upper bounds on the values of each sub-rectangle are introduced when they can be calculated easily, as a means of speeding up the computation. If the value of each piece is equal to its area, this bound is simply the area of the sub-rectangle.

Herz compares his algorithm to that of Gilmore and Gomory's and concludes that the use of true recursion, canonical dissections and the upper bound result in a more efficient procedure. *Beasley* [1985a] describes a similar iterative procedure.

Gilmore and Gomory [1965] consider CSP 's in which there are two stages of cutting. This means the corresponding generalised knapsack problem is of the

form: maximise $\upsilon_1 a_1 + \upsilon_2 a_2 + ... + \upsilon_m a_m$ subject to the condition that [a_1, a_2, ..., a_m] corresponds to a two-stage guillotine pattern. They develop two methods for this problem, one using dynamic programming and the other using linear programming.

The first method is carried out in two stages:

(i) For all widths w_i calculate υ_i^*, the optimum value obtainable by fitting rectangles (l_j, w_j) where $w_j \leq w_i$, end to end into a strip of width w_i and length L. For each i this is a one-dimensional Knapsack problem.

(ii) The optimal value of the objective function $\upsilon_1 a_1 + \upsilon_2 a_2 + ... + \upsilon_m a_m$ is then obtained by solving one more knapsack problem:
maximise $\upsilon_1^* a_1' + \upsilon_2^* a_2' + ... + \upsilon_m^* a_m'$ subject to $W \geq w_1 a_1' + ... + w_m a_m'$ and subject to a_i', $i = 1, ..., m$ being nonnegative integers.

The knapsack problems of (i) above can all be solved together by a dynamic program similar in structure to a program for the one-dimensional problem of section 1.5.1. The same program can also be used to solve the knapsack problem of (ii).

The second method to the two-stage guillotine problem involves a two-stage linear programing formulation of the problem with the first stage corresponding to the process of slitting the stock rectangle into strips with widths corresponding to the widths of the demanded rectangles and with the second stage corresponding to the process of chopping the strips into the demanded lengths. Gilmore and Gomory's approach is a mixture of their LP approach they used for the one-dimensional CSP [1961] and the decomposition of *Dantzig and Wolfe* [1960].

Hahn [1968] shows how the above procedure can be extended to more than two stages. She considers three-stage problems where any cuts at the third stage produce pieces of identical dimensions and there are defects in the rectangle being cut. The values are of the form $\alpha A_p + \beta A_p^2$ where A_p is the area of a piece. The method was designed for the glass industry and a dynamic programming algorithm is used to produce a cutting pattern in which the sums of heuristic values are maximised.

Related to the two-dimensional Knapsack problem, is the so-called template layout problem, in which the pieces to be cut are not rectangular. Here there is a demand for an unlimited number of various two-dimensional pieces and the objective is to cut the most valuable combination of pieces from a single sheet of stock material. *Haims and Freeman's* [1970] approach to the solution of this problem is to enclose the irregular pieces into rectangular areas (modules) from which they are to be cut. A dynamic programming algorithm is then used to lay out the resulting modules in the rectangular stock sheet.

A number of heuristic or partially heuristic solution approaches involving the methods mentioned above for staged cutting have also been applied to some unconstrained cutting problems in the glass industry. Such problems have been discussed by *Dyson and Gregory* [1974], *Chambers and Dyson* [1976], *Madsen* [1979] and *Farley* [1983a, 1983b] (section 1.5.7).

1.5.5 The Constrained Two-Dimensional Knapsack Problem

In practice, cutting problems appear in a constrained form, the most usual constraint being the one that restricts the maximum number of pieces of each type to

be cut. The version of the problem stated in the previous section, when this restriction is added, is known as the constrained two-dimensional knapsack problem. There have been two optimal tree-search algorithms developed to solve this problem; one method deals with guillotine cuts and was developed by *Christofides and Whitlock* [1977], the other method deals with non-guillotine cuts and was developed by *Beasley* [1985b].

Christofides and Whitlock present a tree-search procedure to solve the constrained problem of cutting the most valuable combination of demanded rectangles from a stock rectangle. Their method generates all possible cutting patterns without duplication in the stock rectangle. All of these patterns can be represented in the form of a tree in which, each node is defined by a set of cut rectangles together with the next cutting position in each and branchings corresponding to guillotine cuts. Like Herz, they reduce the number of posssible positions through canonical dissections, called in this case"normal cuts". By using the *Gilmore and Gomory* [1966] unconstrained two-dimensional knapsack solution, obtained by dynamic programming and a transportation routine, an upper bound on the value of layouts derived from any node can be determined. This upper bound is incorporated into a branch-and-bound procedure in order to limit the amount of search necessary to obtain the optimal layout of demanded rectangles on the stock rectangle. It is reported that the method can be used to solve practical problems of medium size (twenty pieces in the order list).

The work of *Beasley* [1985b] provides an exact solution to non-guillotine problems in which the objective is to find a layout of the pieces in the stock rectangle that has the highest possible total value. He formulates the problem as an integer program as follows:

Let $a_{ipqrs} = 1$ if a piece of type i overlaps co-ordinates (r, s) when cut with its bottom-left hand corner at (p, q)

$= 0$ otherwise

and

$x_{ipq} = 1$ if a piece of type i is cut with its bottom-left hand corner at (p, q)

$= 0$ otherwise

Then the optimal layout of the pieces on the stock rectangle (L, W) is given by:

$$\text{Max} \sum_{i=1}^{m} \sum_{p=0}^{L-1} \sum_{q=0}^{W-1} v_i \, x_{ipq}$$

subject to

$$\sum_{i=1}^{m} \sum_{p=0}^{L-1} \sum_{q=0}^{W-1} a_{ipqrs} \, x_{ipq} \leq 1 \quad , \; r = 0, ..., L \text{ and } s = 0, ..., W$$

$$P_i \leq \sum_{p=0}^{L-1} \sum_{q=0}^{W-1} x_{ipq} \leq Q_i , \quad i = 1, ..., m$$

$$x_{ipq} \in \{ 0, 1 \}, \qquad i = 1, ..., m, \; p = 0, ..., L \text{ and } q = 0, ..., W$$

where P_i and Q_i represent the minimum and maximum number of pieces of type i that can be cut from (L, W), respectively.

The size of the program is then reduced by restricting the values of p and q to normal cutting positions. Lagrangean relaxation of the given formulation and

subgradient optimisation are used to obtain a good upper bound to the problem and a heuristic, based upon random placings, is used to obtain an initial lower bound. When these do not coincide, a tree search procedure is initiated and the upper bound is recalculated at each node. Results are reported for random problems which involve up to ten types of pieces (the number of pieces of each type being one, two or three) to be cut from stock rectangles of up to thirty units square.

The algorithms for the constrained two-dimensional knapsack problem mentioned above, are not generally suitable for solving problems of large size which are often met in applications. A number of heuristic or partially heuristic solution methods have been published, which appear to be particularly effective when the number of pieces to be packed or cut from the stock rectangle is large. For example, *Wang* [1983] approaches the problem in the following way: instead of enumerating all possible cuts that can be made on the stock rectangle (*Christofides and Whitlock* [1977]), he builds guillotine cutting patterns by successively adding sub-rectangles to each other. For any pair of current rectangles, a new rectangle is obtained by joining them in either a horizontal or vertical build. The number of possible combinations is reduced by placing an upper bound on the acceptable percentage of waste they create. The algorithm determines error bounds that measure the closeness of the best patterns to the optimal solution.

A special case of the two-dimensional CSP, is the pallet loading problem with only one size of rectangular piece to be cut and with the objective of maximising the number of cut pieces. In cases where packaged material shipped in trucks, railcars aircraft and ships is packed on a pallet or in some other bulk container, the packing problem can be stated simply as trying to pack as many packages as possible into a container. Recently, a number of solution techniques have been developed to cater specifically for packing problems. *Steudel* [1979]

observes that in solving such problems, practical experience suggests that considerable advantage in terms of the utilisation of the pallet often can be gained by employing loading patterns which could not be obtained with guillotine type cuts (see Appendix B of Chapter 7). *Smith and DeCani* [1980], *Bischoff and Dowsland* [1982], *Dowsland* [1982], *Hodgson* [1982] and *Hodgson et al* [1983] have developed heuristic procedures for this problem of non-guillotine type.

1.5.6 The Two-Dimensional Trim Problem

The general two-dimensional trim problem, where there are n stock rectangles and a minimum demand Q_j on all the pieces (l_j, m_j), j = 1, ..., m to be cut, is too complicated to solve optimally and only certain subproblems with extra constraints have been considered in the literature by using heuristics. *Gilmore and Gomory* [1965] formulate this problem in a similar manner to the one-dimensional , with the complication that the cutting patterns are now for rectangular objects. It is noted that the difficulty in solving this problem as an ordinary LP problem is the immense number of columns in the constraint matrix. Applying a column generating technique of the type first presented in their paper of 1961 (see section 1.5.3) produces a generalised two-dimensional knapsack problem of the form:
maximise $v_1a_1 + v_2a_2 + ... + v_ma_m$ subject to the condition that $[a_1, a_2, ..., a_m]$ corresponds to an acceptable pattern. In their paper of 1966, they present a dynamic programming algorithm to solve this problem for the case in which all the cuts are restricted to be guillotine (Two-stage guillotine cutting patterns - see section 1.5.4). Alternatively, the technique of *Herłz* [1972] can be applied to the auxiliary knapsack problem. Essentially, the linear programming method is the most powerful algorithm available for finding an aproximate solution to the two-dimensional trim problem provided an efficient technique exists to solve the

auxiliary knapsack problem.

The Gilmore and Gomory LP approach using an example in the glass industry is shown to be a simplification of the real problem which involves the selection of the sequence in which the cutting patterns are to be processed. *Dyson and Gregory* [1974] identify this as the pattern allocation problem. They developed a two-stage heuristic approach to the solution of this problem; patterns are designed by applying the LP technique of Gilmore and Gomory and the sequencing of these patterns is treated as a problem of the travelling salesman type. An additional restriction on the problem arising in the glass industry, is that of imposing limitations on the positioning of cuts within the stock rectangle, relative to other cuts. *Farley* [1983a, b] considered this problem and presented a heuristic based on modifications to the *Gilmore and Gomory* [1965] algorithm. A glass cutting problem occuring in a small firm was examined by *Madsen* [1979]; his heuristic solution method resulted in approximately 50% reduction in waste compared to the solution normally used by the company.

Not all applications of stock-cutting deal with rectangular pieces and problems of packing non-rectangular shapes have been tackled in the garment and shipbuilding industries : In ship-building the problem is usually called a nesting problem. *Adamowicz and Albano* [1972] present a two-stage heuristic algorithm to solve this problem, in which it is required to cut out a specified set of irregular shapes from a number of rectangular stock sheets so as to minimise the amount of waste. The first stage consists of forming clusterings of the irregular pieces and then enclosing these clusterings into rectangular modules (*Adamowicz and Albano* [1976a]). The second stage concerns the layout of the rectangular modules onto the stock rectangle. *Adamowicz and Albano* [1976b] present a constrained dynamic programming algorithm to lay out groups of rectangles called strips which are then

packed within a rectangular enclosure. *Albano and Orsini* [1979] present an improved and extended version of this heuristic which provides greater flexibility in the use of strips.

1.5.7 The Two-Dimensional Bin Packing Problem

In this section, we consider one of the currently most active areas in the operations research literature : the problem of packing a set L of rectangles, each of given height h_j and width w_j (j = 1, ..., m), into two-dimensional bins. The goal is to pack them in a vertical strip of width C, so as to minimise the total height of the strip needed.

The problem described above, is a fundamental one for which a broad application in operations research is easily envisioned. The obvious interpretation of bin packing corresponds to problems of efficient use of time and/or space, especially problems in computer scheduling, where the items to be packed correspond to tasks. The height of an item represents the amount of processing time it requires and its width is the amount of contiguous memory it needs. The strip width C is then the total memory available and the strip length the amount of time needed to schedule all the items.

Further motivation for bin packing problems has been provided by the obvious industrial applications in stock cutting. In the simplest industrial setting, where the "raw" material involved comes in rolls, the objects required to be cut from the rolls, are viewed as being rectangles. The possible wastage is then minimised if we minimise the amount of roll (the strip length) used. The restriction of orthogonality applies once again, as in many applications, the cutting is done by

blades that must be either parallel or perpendicular to the strip.

It is readily verified that the bin-packing problem, or more precisely the decision problem "Given C, L and an integer bound K , can L be packed into K or fewer bins of capacity C? " (the classical one-dimensional bin-packing problem) is NP-complete (section 1.2). By the theory elaborated in *Garey and Johnson* [1979] and *Karp* [1972], this means that it is unlikely that efficient, (i.e. polynomial time) optimisation algorithms can be found for this problem.

Bin-packing algorithms usually consist of a specified ordering of pieces and a placement policy. In the simpler methods, the ordering is fixed at the start, while dynamic orderings allow the choice of the next piece to be made at each step. *Christofides* [1974] suggests that possibilities for fixed orderings include descending length, width, area, perimeter, or maximum dimension. Variable orderings are based on expected waste, which may be measured by the total waste to the left of each piece, by the waste below a piece, or by the evenness of the edge of the layout. *Short* [1973] has analysed the effects of some of these rules.

The evaluation procedure applied to the performance of the heuristic algorithms is usually in the form of worst case analysis in which the ratio of the actual solution to the optimal solution is examined. Two types of bounds are defined. Let A (L) be the height used by algorithm A , for packing a list L of rectangles and let OPT (L) be the height which would be used in an optimal packing. If for all lists L, $A(L) \leq \alpha\, OPT(L)$ then α is called the absolute bound. A second measure, the asymptotic bound is defined by: $A(L) \leq \alpha\, OPT(L) + \beta H$ for all lists, L, of rectangles with maximum height H, where β is the asymptotic bound.

A number of authors have presented worst case evaluations of various bin-packing heuristics (*Garey, Graham and Ullman* [1973]). Johnson's doctoral thesis in 1973 has laid the theoretical groundwork for the worst-case analysis of the class of heuristic algorithms known as any-fit. Summary of this work appears in *Johnson* [1974]. A review of the literature for one-dimensional bin-packing algorithms is beyond the scope of this thesis - however, the interested reader is directed to reviews by *Johnson, Demers, Ullman, Garey and Graham* [1974], and *Garey and Johnson* [1981]. Garey and Johnson also have in 1984 put together a comprehensive paper incorporating both new and old results; as far as we are aware, this is the most recent survey in the literature of approximation algorithms for bin packing both in one and two-dimensions.

Baker, Coffman and Rivest [1980] consider a variety of strip packing algorithms based on a "bottom up-left justified" (BL algorithm) placement policy using a variety of orderings. The list of rectangles is packed such that each is placed in turn as low as possible in the "bin" and then left-justified. They show that the performance of such a placement procedure can be arbitrarily bad, but by ordering pieces by decreasing width an absolute bound of 3.0 OPT (L) applies. They also illustrate that with some problems, regardless of ordering applied, an optimal solution cannot be obtained using a BL placement policy. *Brown* [1980] provides an example where a BL packing requires at least 1.25 OPT (L), regardless of ordering.

The search for algorithms with better asymptotic worst case ratios was taken up by *Coffman, Garey, Johnson and Tarjan* [1980]. They propose three level-oriented algorithms in which the rectangles to be packed are preordered by non-increasing height and then placed at a series of "levels". The first level is simply the bottom of the bin. Subsequent levels are defined by a line drawn

horizontally through the top of the highest rectangle packed in the previous level. The pieces are placed left-justified at a level which is determined by one of two basic placement rules. The NFDH rule (next-fit decreasing height) uses the highest existing level and the FFDH (first-fit decreasing height), the lowest suitable level. Figure 1.5 illustrates an example of the application of these two rules to a set of 6 boxes labelled A-F. The analysis by Coffman et al shows that the asymptotic performance bounds of NFDH and FFDH are 2.0 and 1.7, respectively.

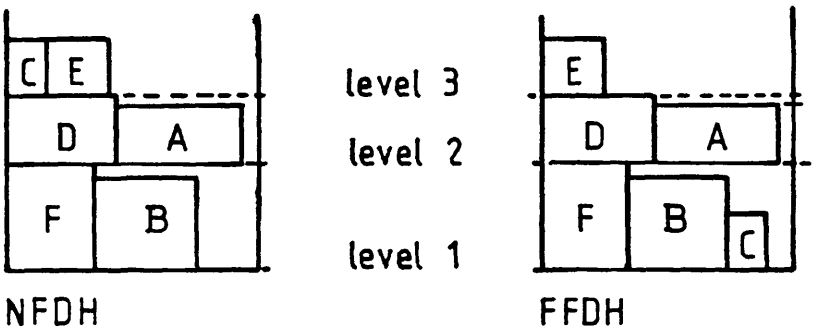


Figure 1.5 An example of a NFDH and a FFDH packing.

We should note that level-by level packings have a special significance stemming from their relation to guillotine cuts (*Gilmore and Gomory* [1965]). The 3-stage guillotine cuts corresponding to level-by-level packings such as Figure 1.5 involve first a set of horizontal guillotine cuts, then a set of vertical cuts and finally another set of horizontal "trim" cuts.

1.6 Thesis Outline

The remainder of this thesis is presented in seven chapters. In Chapter 2 the two-dimensional unconstrained guillotine cutting problem is solved using the method of dynamic programming. The solution method is illustrated by an example.

A tree-search algorithm for the two-dimensional constrained guillotine cutting problem is described in Chapter 3. A tight bound, derived from the state-space relaxation of a dynamic programming recursion given for the original problem, is used to limit the search. The bound is improved by state-space ascent methods. The computational performance of the algorithm is presented for a number of randomly generated problems with constraints of varying " tightness".

Chapter 4 presents an interactive system with graphical input-output for generating rectangular layouts manually for the problem of Chapter 3. The structure and the main features of an experimental version of the system are described and the results of manual experiments are discussed.

In Chapter 5 we discussed the cutting problems in which the optimal cutting patterns are not restricted to those with the guillotine property. Two mixed integer programming formulations of the problem are presented and a possible method of solution based on the use of cutting planes is investigated. Five 0-1 integer programming formulations of the same problem are also given and upper bounds are derived from the linear programming relaxations. The five bounds developed are evaluated and compared on a number of small randomly generated non-guillotine cutting problems.

In Chapter 6 the Lagrangean relaxation technique is used for a 0-1 integer programming formulation of the problem of Chapter 5. The resulting upper bound is improved by a subgradient optimisation method. Problem reduction tests derived from both the original problem and the Lagrangean relaxation are given. The final bound produced is evaluated on a number of test problems.

A tree-search procedure for solving the non-guillotine problem is the subject

of Chapter 7. The process of generating a finite number of orthogonal cutting patterns of rectangles is described. The algorithm incorporates into the tree-search the bound produced for the problem in Chapter 6. Computational results for a number of randomly generated problems are given.

Finally, Chapter 8 provides a summary of the main findings and achievements of this thesis.

1.7 Conclusions

The cutting stock problem is a large scale combinatorial problem encountered in a variety of industrial applications. In this chapter, the various types of this problem and solution methods have been presented, with emphasis being placed on the area of two-dimensional cutting. The combinatorial nature of the problems has often led to heuristic methods being adopted. However, it is clear that there is no generally applicable heuristic algorithm which can be applied to cutting problems in two-dimensions.

We have seen that much of the early work in this area is based on guillotine cuts. Christofides and Whitlock have solved optimally the constrained guillotine cutting problem. The class of general cutting problems has been considered by relatively few authors in the literature and as far as we are aware, the only exact solution procedure for this problem that exists in the literature is due to Beasley. In the case of these two exact methods, the computational experience quoted by their authors suggests that their application is limited to problems of moderate size.

Although the theory of computational complexity might well mean that we will never be able to guarantee obtaining an optimal solution to the CSP without resorting to an exponentially increasing algorithm (such as a tree-search procedure), this does not mean that such solutions cannot be obtained relatively quickly for many large sized problems.

CHAPTER 2

TWO - DIMENSIONAL UNCONSTRAINED GUILLOTINE CUTTING (UGC)

2.1 Introduction

The general two - dimensional cutting problem is the problem of cutting a number of smaller rectangular pieces, each of a given size and value, from a large rectangular stock plate, so as to maximise the value of the pieces cut. By taking the value of a piece to be proportional to its area, the value maximisation problem becomes one of minimising the amount of waste produced by the cutting.

In a great many industrial situations, a cut in a piece of material must begin on one side of the material and traverse the material in a straight line to the other side. This is the kind of cut made by many types of machinery. One example is the guillotine cutter used in cutting paper sheets and for this reason this type of cut is referred to as a " guillotine " cut. In this special case of problems, the cutting is

performed as follows: one first performs a guillotine cut on the stock plate and then proceeds in the same way with the two resulting rectangles. Restricting the permissible cuts to be of " guillotine " type, severely limits the permissible cutting patterns, a pattern being defined as a cutting arrangement of a combination of pieces within the stock - plate. For example, the pattern of Fig. 2.1 cannot be produced by guillotine cuts. Fig. 2.2 shows a possible cutting pattern using guillotine cuts where the cuts are numbered in the order in which they could be made, although other sequences are obviously also possible.

A further restriction common in the literature is to limit the cutting that occurs to a number of " stages ". Regarding any two adjacent edges of the rectangle to be cut as x and y axes, as shown in Fig. 2.3, then the cuts are restricted to be made parallel to the x and y axes alternatingly. Fig. 2.3 illustrates the same cutting pattern presented in Fig. 2.2 generated in four stages, with the number by the cuts indicating the stage at which the cut is made. Thus, the cut direction at the first stage is parallel to the y - axis; at the second stage parallel to the x - axis; at the third stage parallel to the y - axis and at the fourth stage parallel to the x - axis.

Whilst the general two - dimensional cutting problem has been considered by relatively few authors in the literature, the restricted versions of the problem given above - guillotine and stage cutting - have been considered by a number of authors. This is in part due to the important role these problems play in practical applications as well as the development of such computer oriented optimization techniques as linear and dynamic programming. *Gilmore and Gomory* [1961, 1963, 1965, 1966] made effective use of both techniques to handle the one -, two - and three - dimensional rectangular cutting-stock problems when the cutting of the pieces was restricted to two - or three - stage guillotine cuts. The results they obtained when their algorithms were tested on a number of representative cutting problems

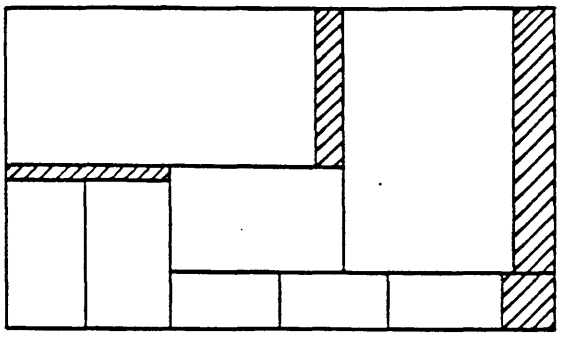


Figure 2.1 A cutting pattern infeasible with guillotine cuts.

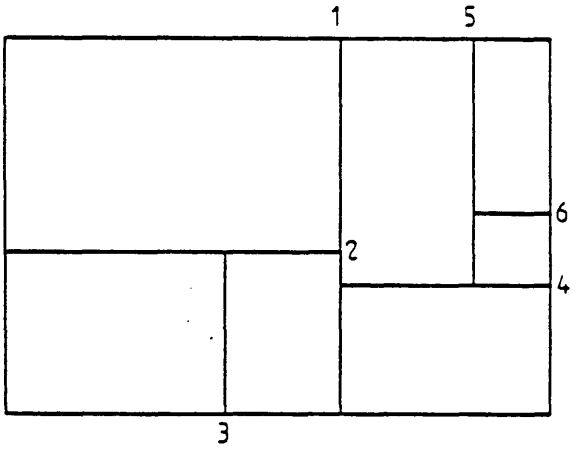


Figure 2.2 A guillotine cutting pattern.

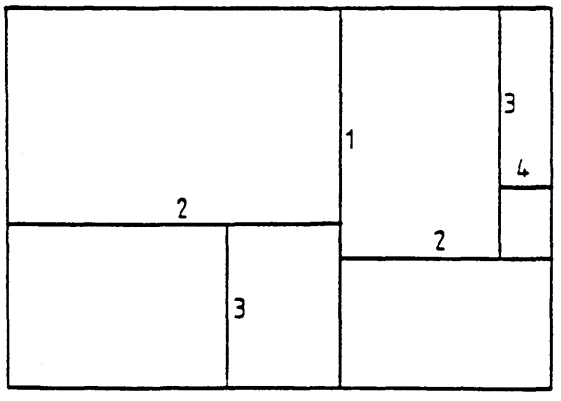


Figure 2.3 Four - Stage cutting.

were surprisingly good. It is shown that when cutting is done in stages, a generalised knapsack problem can be efficiently solved as a subproblem. *Hahn* [1968] considered three - stage problems where any cuts at the third stage produced pieces of identical dimensions and there were defects in the rectangle being cut. She used an extension of the approach for two - stage cutting that *Gilmore and Gomory* [1965] developed. *Herz* [1972] was also able to obtain multistage guillotine cut solutions using a recursive search approach which was shown to be an improvement over the exhaustive iterative type approach. The algorithm, however, is not an efficient procedure for solving problems of even medium size.

We note here that virtually all the approaches in the literature for two dimensional guillotine cutting involve the use of dynamic programming (see also *Haims and Freeman* [1970], *Beasley* [1985]). With most of these approaches it was assumed that there was no bound on the number of occurrences of any particular type of piece in the solution. *Christofides and Whitlock* [1977], however, presented a tree - search algorithm for guillotine cutting problems in which there is a constraint on the maximum number of each type of piece that is to be produced. The results reported in this paper indicate that the algorithm is an effective procedure for solving cutting problems of medium size.

In this chapter, we develop a new dynamic programming recursion for unconstrained two - dimensional guillotine cutting. The idea of normal cutting patterns, as defined by *Christofides and Whitlock* [1977], is used to improve computationally the basic recursion.

The use of dynamic programming is illustrated by an example presented in the last section of this chapter.

2.2 Definition of the Unconstrained Problem

The unconstrained two - dimensional guillotine cutting problem P_1 can be defined as follows: Let a large rectangle $A_0 = (\alpha_0, \beta_0)$ (i. e. of length α_0 and width β_0) be given, together with a set R of m smaller rectangular pieces, $R = \{ (\alpha_1, \beta_1), (\alpha_2, \beta_2), ..., (\alpha_m, \beta_m) \}$, each piece in R having associated with it a value v_i. The problem is to construct a guillotine cutting pattern for A_0 having the maximum value of pieces cut from A_0. Note here that there are no constraints on the number of pieces produced and that any piece cut from A_0 that is not of size (α_j, β_j) for some j ($j = 1, ..., m$) is taken to be of value zero.

In order to distinguish between the given pieces in set R and the rectangles produced by the cuts on A_0 at any stage during the cutting process, we will refer to the former as " pieces " and the latter as " rectangles ".

We will make the following assumptions for problem P_1:

(i) All dimensions (α_i, β_i) for $i = 0, 1, ..., m$ are integers and the cuts on the rectangles are to be made in integer steps along the x or y axes. Let $L=\{1, 2, ..., \alpha_0 - 1\}$ and $W = \{ 1, 2, ..., \beta_0 - 1 \}$ represent the sets of all possible lengths and widths, respectively, for guillotine cuts on A_0.

(ii) The orientation of the pieces is considered to be fixed, i. e. a piece of length l and width w is not the same as a piece of length w and width l ($l \neq s$). Problem P_1, as described above, will be referred to as the Unconstrained Guillotine Cutting (UGC) problem.

2.3 A Dynamic Programming (DP) Formulation of the UGC Problem

For several decades, dynamic programming has been proposed as an effective method of solving combinatorial problems of a sequential nature. It is considered to be computationally advantageous to use dynamic programming since the concept can provide convergence to an optimum solution without total enumeration.

In the development of dynamic programming recursion formulae, the problem is decomposed into stages which are evaluated independently . In the case of the UGC problem, a stage in the dynamic programming recursion corresponds to a stage in the process of generating a cutting pattern. In this problem, the cuts alternate at each stage between being parallel to the y-axis and being parallel to the x-axis. Hence, we can associate with each stage a cut direction. Note, however, that we do not require a cut to be made at each stage.

To formulate the problem we define $F_k (x, y)$ as the maximum value obtained at the k - stage cut of a rectangle of size (x, y) when the first - stage cut direction is parallel to the y - axis and $G_k (x, y)$ as the maximum value of a k - stage cut of a rectangle of size (x, y) when the first - stage cut direction is parallel to the x - axis. Thus, the size (x, y) of a rectangle to be cut at the kth stage of the cutting process, will correspond to a state in our DP formulation. A value of zero for k, corresponds to the fitting of one piece into rectangle (x, y) and hence

$$F_0 (x, y) = \max_i \ (v_i \mid \alpha_i \le x, \beta_i \le y, \ i = 1, ..., m) \qquad (1)$$

Note that $F_0(x, y) = G_0(x, y)$ for any rectangle of size (x, y) - See Fig. 2.4 (a).

For an optimal k - stage cut of a rectangle (x, y), where the first - stage cut direction is parallel to the y - axis, there are only two alternatives:

(a) There is at least one first - stage cut parallel to the y - axis at some $x' \varepsilon L$ - as in Fig. 2. 4 (b).

(b) There are no first - stage cuts parallel to the y - axis but at least one second - stage parallel to the x - axis (at some $y' \varepsilon W$) - as in Fig. 2. 4 (c). In this case we have a cutting pattern where the first - stage cut direction is parallel to the x - axis and there are (k - 1) stages to the cutting pattern.

Hence, the DP recursion of the UGC problem can be stated as follows:

$$F_k(x, y) = \max \left[G_{k-1}(x, y); \max_{\substack{x' < x \\ x' \varepsilon L}} \{ F_k(x', y) + G_{k-1}(x-x', y) \} \right] \quad (2)$$

A similar argument to the one given above can be used to show that

$$G_k(x, y) = \max \left[F_{k-1}(x, y); \max_{\substack{y' < y \\ y' \varepsilon W}} \{ G_k(x, y') + F_{k-1}(x, y-y') \} \right] \quad (3)$$

Equations (2) and (3) are the basic dynamic programming recursions for the optimal k - stage cutting of a rectangle and they apply for any $k \geq 1$ and any rectangle (x, y). Equation (1) provides initial conditions for the recursion. $F_n(\alpha_0, \beta_0)$ or $G_n(\alpha_0, \beta_0)$ gives us the value of an optimal n - stage cutting

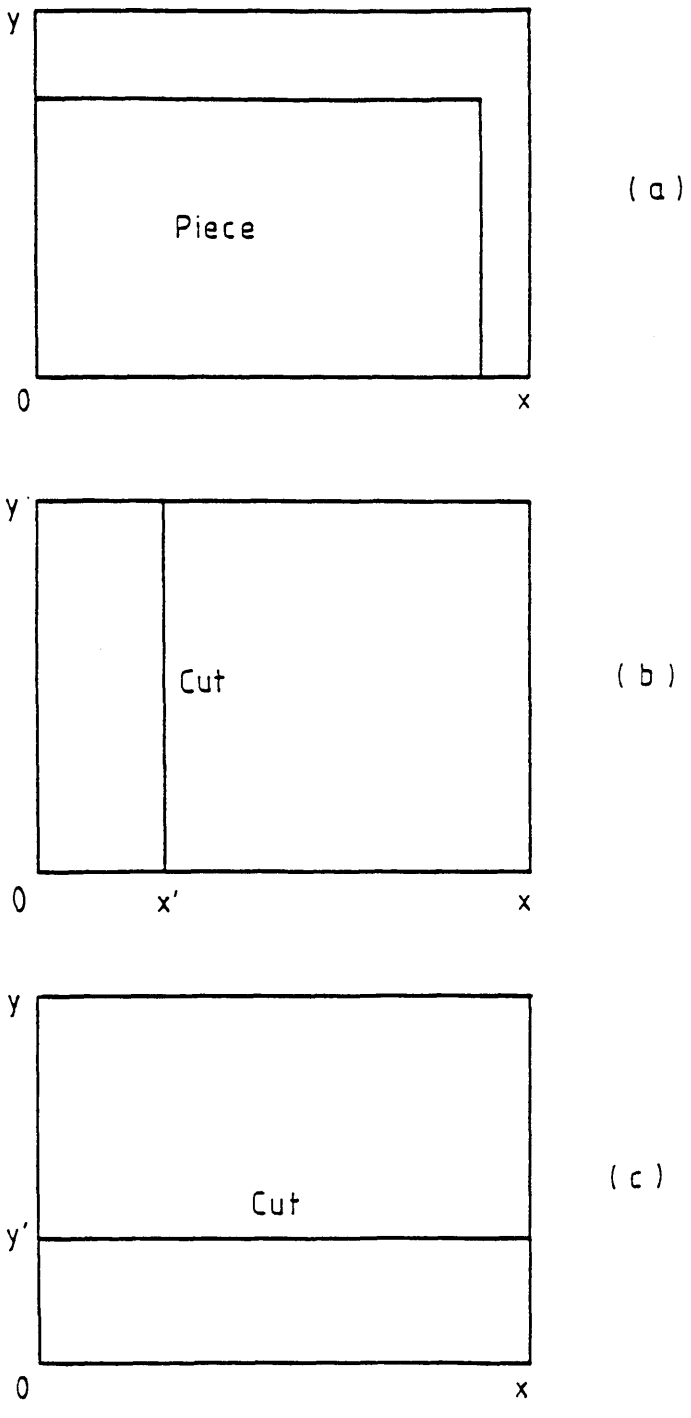


Figure 2.4 (a) Trim one piece (k = 0).
(b) First - stage cut parallel to the y - axis.
(c) No first - stage cut, but a second - stage cut parallel to the x - axis.

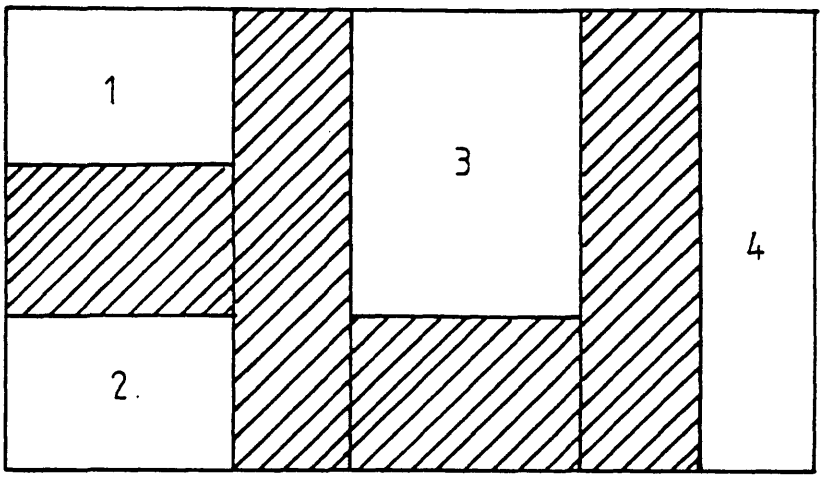
pattern for A_0, depending upon the first - stage cut direction specified. If, however, n is not known, the number of stages we perform in the cutting process is equal to that value of k (i. e. n = k) for which $F_k(\alpha_0, \beta_0) = F_{k+1}(\alpha_0, \beta_0)$ and $G_k(\alpha_0, \beta_0) = G_{k+1}(\alpha_0, \beta_0)$. In this case the optimal value of the n - stage pattern is given by max $[F_n(\alpha_0, \beta_0), G_n(\alpha_0, \beta_0)]$ if the first - stage cut direction is unspecified. It should be noted that when $F_n(\alpha_0, \beta_0)$ and $G_n(\alpha_0, \beta_0)$ have been computed, so have $F_k(x, y)$ and $G_k(x, y)$ for all $k, 0 \leq k \leq n$, all $x \in L$ and $y \in W$.

2.3.1 Normal Patterns

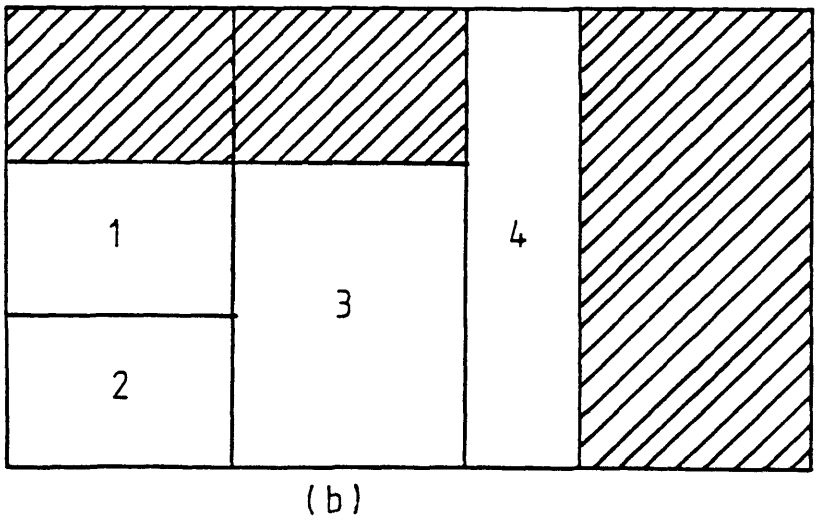
Normal patterns were used by *Hertz* [1972] (who called them canonical dissections) and *Christofides and Whitlock* [1977]. According to their definition of normal patterns, any cutting pattern can be normalised such that any piece cut has its left - hand edge and its bottom edge adjacent to other cut pieces or to the edges of A_0 as shown in Fig. 2.5. For any pattern there is a normal equivalent. A consequence of this is that the set L of possible lengths for any cuts parallel to the y - axis given in section 2. 2 can be restricted to the following:

$$L = \{ x \mid x = \sum_{i=1}^{m} \theta_i \alpha_i ;\ 1 \leq x \leq \alpha_0 , \theta_i \geq 0 \text{ and integer } \forall\, i = 1, ..., m \} \qquad (4)$$

Equation (4) essentially says that if $x \in L$ there exists a set of pieces whose lengths add up to x. The equivalent definition for W is:



(a)



(b)

Figure 2.5 (a) An unnormalised cutting pattern.
(b) A normalised cutting pattern.

$$W = \{\, y \mid y = \sum_{i=1}^{m} t_i \beta_i ,\ 1 \le y \le \beta_0 ,\ t_i \ge 0 \text{ and integer } \forall\, i = 1, \ldots, m \,\} \qquad (5)$$

Note that normality is a property of a cutting pattern that is relative to the set of pieces in R available for cutting from A_0. Sets L and W are easily calculated (*Christofides and Whitlock* [1977]).

Normal patterns as described above can be used to improve computationally the basic dynamic programming recursion given by equations (2) and (3) in the following way:

Let $l(x)$ represent the length nearest to x in the normalised set of lengths L $[l(x) \le x]$ for any $x \in L$, $l(x)$ being equal to zero if no such length exists. Similarly, we define $w(y)$ as the width nearest to y in the normalised set of widths W $[w(y) \le y]$ for any $y \in W$, $w(y)$ being equal to zero if no such width exists. Note that $l(\alpha_0) = \alpha_0$ and $w(\beta_0) = \beta_0$ may not be strictly necessary since, for example, there may not exist a set of piece lengths which add to α_0 (and similarly for β_0). Thus, we define:

$$l(x) = \max [\, 0,\ x' \mid x' \le x, x' \in L \,] \qquad (6) \quad \text{and}$$

$$w(y) = \max [\, 0, y' \mid y' \le y,\ y' \in W \,] \qquad (7)$$

We claim that we may calculate $F_k(l(x), w(y))$ instead of $F_k(x, y)$ since the optimal k - stage guillotine cutting pattern for (x, y) can be normalised into a k - stage pattern for $(l(x), w(y))$ so that

$$F_k(l(x), w(y)) = F_k(x, y) \qquad (8)$$

A similar argument holds for $G_k(x, y)$, namely $G_k(\iota(x), w(y)) = G_k(x, y)$. (9)

Equation (2) can now be modified to be

$$F_k(x, y) = \max [\, G_{k-1}(x, y);\ \max_{\substack{x' < x \\ x' \in L}} \{ F_k(x', y) + G_{k-1}(\iota(x-x'), y) \} \,] \quad \forall\, k \geq 1,\ x \in L \text{ and } y \in W \quad (10)$$

In a similar fashion, we can modify the recursion for $G_k(x, y)$ [equation (3)] to be:

$$G_k(x, y) = \max [\, F_{k-1}(x, y);\ \max_{\substack{y' < y \\ y' \in W}} \{ G_k(x, y') + F_{k-1}(x, w(y-y')) \} \,] \quad \forall\, k \geq 1,\ x \in L \text{ and } y \in W \quad (11)$$

Equations (10) and (11) are used in calculating $F_n(\alpha_0, \beta_0)$ and $G_n(\alpha_0, \beta_0)$. It is now clear that, it is not necessary to calculate $F_k(x, y)$ and $G_k(x, y)$ for all values of x and y, but it is sufficient to restrict attention to $x \in L$ and $y \in W$. This is so since any optimal k - stage cutting pattern has its normal equivalent. Thus, the modified recursion given above is computationally more effective than the recursion given previously [equations (2) and (3)]. To clarify the application of equations (10) and (11) we give in the next section a detailed procedure for solving the UGC problem.

2.3.2 The Dynamic Programming Procedure

Although the application of the above recursion [equations (8) and (9)] produces the optimal value, there still remains the question of finding the nature of

the cutting pattern associated with it . To do this we require four memory grids: $F_k(x, y)$, $G_k(x, y)$, $\Phi_k(x, y)$ and $\Psi_k(x, y)$. The grids $\Phi_k(x, y)$ and $\Psi_k(x, y)$ are used to record how the values $F_k(x, y)$ and $G_k(x, y)$ are achieved. When the computation is completed i.e. when $F_k(x, y) = F_n(\alpha_0, \beta_0)$ and $G_k(x, y) = G_n(\alpha_0, \beta_0)$, $\Phi_k(x, y)$ and $\Psi_k(x, y)$, being used for backtracking, are defined as follows: If the value $F_k(x, y)$ has been achieved by cutting a rectangle (x, y) at some normalised length $x' \varepsilon L$ giving two rectangles of sizes (x', y) and $(x - x', y)$, $\Phi_K(x, y) = x'$; if no such cut is made, $\Phi_k(x, y) = 0$. Formally, we define:

$$\Phi_k(x, y) = x' \text{ if } x' < x,\ x' \varepsilon L,\ F_k(x, y) = F_k(x', y) + G_{k-1}(l(x - x'), y)$$
$$= 0 \text{ otherwise}$$

and, similarly:

$$\Psi_k(x, y) = y' \text{ if } y' < y,\ y' \varepsilon W,\ G_k(x, y) = G_k(x, y') + F_{k-1}(x, w(y - y'))$$
$$= 0 \text{ otherwise.}$$

To help in describing the procedure, let $x_1, x_2, ..., x_p$ be the elements of L in order of increasing lengths and $y_1, y_2, ..., y_q$ be the elements of W in order of increasing widths.

Initialisation.

1. 1. Set $k = 0$.

1. 2. Set $F_0(x_s, y_t) = G_0(x_s, y_t) = \max_i (\upsilon_i \mid \alpha_i \le x_s, \beta_i \le y_t,\ i = 1, ..., m)$
for all $s = 1, 2, ..., p$ and $t = 1, 2, ..., q$.

1. 3. Set $k = 1$

1. 4. Set $s = 1$, $t = 1$.

First - stage cut parallel to the y - axis.

2. 1. Set $s' = 1$.

2. 2. If $s' < s$ and $F_k(x_s, y_t) < F_k(x_{s'}, y_t) + G_{k-1}(l(x_s - x_{s'}), y_t)$, set $F_k(x_s, y_t) = F_k(x_{s'}, y_t) + G_{k-1}(l(x_s - x_{s'}), y_t)$ and $\Phi_k(x_s, y_t) = x_{s'}$ and go to 2. 3. If $s' < s$ then go to 2. 3 ; else go to 2. 4.

2. 3. Set $s' = s' + 1$ and go to 2. 2.

2. 4. If $G_{k-1}(x_s, y_t) < F_k(x_s, y_t)$ go to 2. 5 ; otherwise set $F_k(x_s, y_t) = G_{k-1}(x_s, y_t)$, $\Phi_k(x_s, y_t) = 0$ and continue.

2. 5. If $s < p$ then set $s = s + 1$ and go to 2. 1; otherwise, if $t < q$, set $t = t + 1$, $s = 1$ and go to 2. 1, else go to 3. 1.

First - stage cut parallel to the x - axis.

3. 1. Set $s = 1$, $t = 1$.

3. 2. Set $t' = 1$.

3. 3. If $t' < t$ and $G_k(x_s, y_t) < G_k(x_s, y_{t'}) + F_{k-1}(x_s, w(y_t - y_{t'}))$, set $G_k(x_s, y_t) = G_k(x_s, y_{t'}) + F_{k-1}(x_s, w(y_t - y_{t'}))$ and $\Psi_k(x_s, y_t) = y_{t'}$ and go to 3. 4. If $t' < t$ then go to 3. 4 ; else go to 3. 5.

3. 4. Set $t' = t' + 1$ and go to 3. 3.

3. 5. If $F_{k-1}(x_s, y_t) < G_k(x_s, y_t)$ go to 3. 6; otherwise, set $G_k(x_s, y_t) = F_{k-1}(x_s, y_t)$, $\Psi_k(x_s, y_t) = 0$ and continue .

3. 6. If $t < q$ then set $t = t + 1$ and go to 3. 2. ; otherwise, if $s < p$, set $s = s + 1$, $t = 1$ and go to 3. 2., else go to 4. 1.

End of optimal k-stage cutting.

4. 1. If $F_k(x_s, y_t) = F_{k-1}(x_s, y_t)$ and $G_k(x_s, y_t) = G_{k-1}(x_s, y_t)$ then set $n = k - 1$, optimal value = $\max(F_n(x_s, y_t), G_n(x_s, y_t))$ and stop; otherwise, set $k = k + 1$ and to to 1. 4.

At the end of the above procedure $\max(F_n(\alpha_0, \beta_0), G_n(\alpha_0, \beta_0))$ is the value for the optimal n - stage guillotine cutting of (α_0, β_0). If this value is given

by $F_n(\alpha_0, \beta_0)$ then we have also calculated $F_n(x_s, y_t)$ for all $s = 1, 2, ..., p$ and $t = 1, 2, ..., q$ being the value for the optimal guillotine cutting of a rectangle of size (x_s, q_t). Note here that from equation (8) we have obtained $F_n(x, y)$ for all rectangles of size (x, y) where $0 \le x \le \alpha_0$ and $0 \le y \le \beta_0$. In this case, $\Phi_k(x, y)$ can be used to determine how $F_n(x, y)$ is achieved for any x and y. Similarly, $\Psi_k(x, y)$ can be used to determine how $G_n(x, y)$ is achieved for any x and y. Backtracking to find the structure of an optimal k - stage pattern (i. e. the guillotine cuts that are to be made) for a rectangle (x, y) can be carried out by determining a special tree - structure as illustrated in Fig. 2.7. This is a binary tree in which each node has either no other nodes below it in the tree (a terminal node) or has a left node or exactly two nodes immediately below it. A node labelled $F_k(x, y)$ with nodes below it labelled $F_k(x_1, y)$ and $G_{k-1}(x-x_1, y)$ means that the rectangle (x, y) should be divided by a k-stage cut into two rectangles (x_1, y) and $(x - x_1, y)$. Similarly, if a node labelled $G_k(x, y)$ has nodes labelled $G_k(x, y_1)$ and $F_{k-1}(x, y - y_1)$ immediately below it then the rectangle (x, y) should be divided into two rectangles (x, y_1) and $(x, y - y_1)$ at the k^{th} stage of the pattern. A node $G_k(x, y)$ or $F_k(x, y)$ with only one node labelled $F_{k-1}(x, y)$ or $G_{k-1}(x, y)$, respectively, below it means that no k - stage cuts are made on rectangle (x, y). Finally, a node labelled $F_k(x, y)$ or $G_k(x, y)$ is a terminal node if the rectangle (x, y) is not cut any further but a piece r_j in R can be allocated to it such that $j = \max_i \{ v_i \mid \alpha_i \le x, \beta_i \le y, i = 1, ..., m \}$.

Thus, the 3-stage pattern for a rectangle (x, y) shown in Fig. 2.6, can be represented by the binary tree presented in Fig. 2.7. A convenient data structure representing a binary tree is to name the nodes 1, 2, ..., N and to assign an arrow of records to them, each consisting of two fields corresponding to the left and right node deriving from node i. A value of 0 in either field indicates the absence of a left or right descendent node respectively. The binary tree of Fig. 2.7 can be

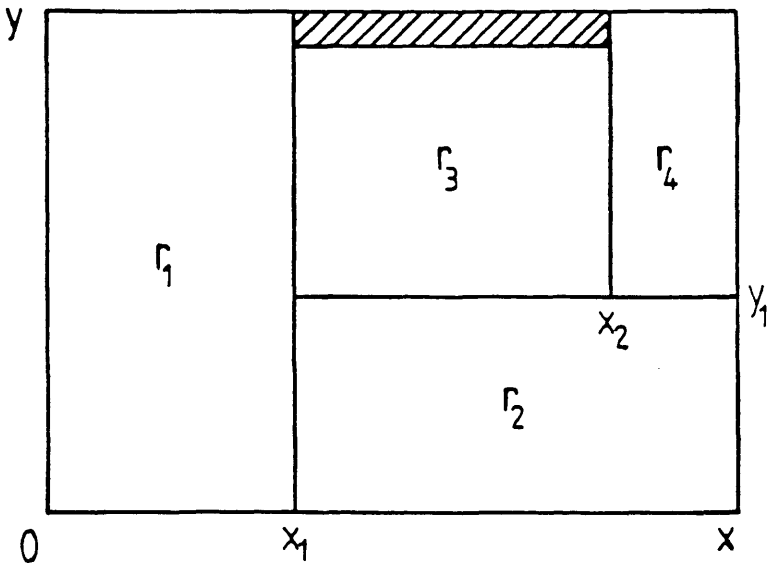


Figure 2.6 An optimal 3 - stage cutting pattern of rectangle (x, y).

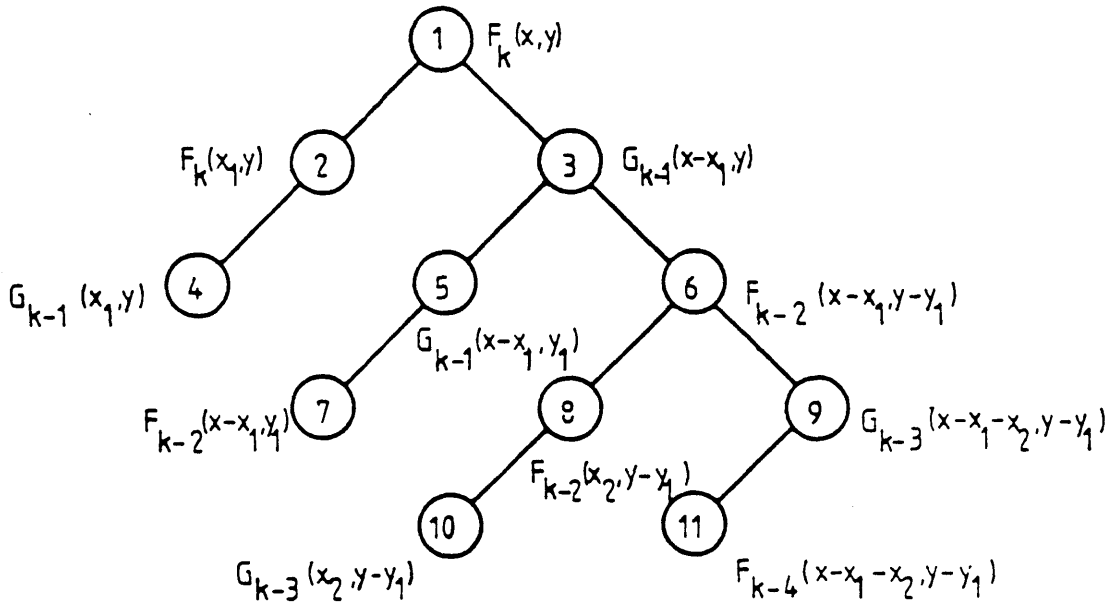


Figure 2.7 Tree - representation of pattern shown in Fig. 2.6.

	left node	right node
1	2	3
2	4	0
3	5	6
4	0	0
5	7	0
6	8	9
7	0	0
8	10	0
9	11	0
10	0	0
11	0	0

Table 2.1 Representation of the binary tree shown in Figure 2.7

NR	XD	YD	CR	AR
1	x	y	x_1	0
2	x_1	y	0	r_1
3	$(x-x_1)$	y	$-y_1$	0
4	$(x-x_1)$	y_1	0	r_2
5	$(x-x_1)$	$(y-y_1)$	x_2	0
6	x_2	$(y-y_1)$	0	r_3
7	$(x-x_1-x_2)$	$(y-y_1)$	0	r_4

Table 2.2 Representation of pattern shown in Figure 2.6

represented as shown in Table 2.1.

The structure described below can represent a list of rectangles produced by a cutting pattern. Consider each rectangle r_i with its lower left corner referred to by the coordinates (0, 0). Using this referencing method, each rectangle r_i is described with the following entries:

(1) XD_i represents the dimension of rectangle r_i parallel to the x - axis.

(2) YD_i represents the dimension of rectangle r_i parallel to the y - axis.

(3) CR_i is the coordinate of the cut which divides rectangle r_i into two further rectangles: positive, if the cut is perpendicular to the x - axis (x - cut) , negative, if perpendicular to the y - axis (y - cut) and zero if the rectangle is not cut.

(4) AR_i is the label of the piece r_j in R of maximum value that can be allocated to rectangle r_i when this is not cut any further.

The pattern shown in Fig. 2.6 would be represented by the list of rectangles shown in Table 2.2, using NR to head the column representing the number of the rectangle. The list begins with the large rectangle (x, y) to be cut. In this list, it can be seen that every rectangle is either (i) cut into two smaller rectangles or (ii) filled by one of the initial pieces given in R (a piece may not fit exactly in a cut rectangle since waste is not cut away by the Dynamic Programming procedure).

2.4 Example for UGC Problem

Consider the problem of cutting a stock plate A_0 of size (15, 10) into a number of smaller rectangles in set R with sizes and values as given below:

Piece i	size i	value i
1	(8, 4)	66
2	(3, 7)	35
3	(8, 2)	24
4	(3, 4)	17
5	(3, 3)	11
6	(3, 2)	8
7	(2, 1)	2

We will use the Dynamic Programming Procedure described in section 2.3.2 to compute an optimal cutting pattern for this problem.

Firstly, we compute the normalised sets of lengths and widths to be L = (2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15) and W = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10) respectively. The initial values for $F_0 (x, y) = G_0 (x, y)$ are given in Table 2. 3, for all $x \in L$ and $y \in W$. Any entry in this table, for instance, $F_0 (6, 7)$ is computed as follows:

$$F_0 (6, 7) = \max (\upsilon_2, \upsilon_4, \upsilon_5, \upsilon_6, \upsilon_7) = \max (35, 17, 11, 8, 2) = 35$$

We will use the value of cutting stage k to index the iterations of recursion (10) and (11) i. e. we will call iteration 1 the iteration which computes all $F_1(x, y)$ and $G_1 (x, y)$, iteration 2 the iteration which computes $F_2(x, y)$ and $G_2(x,y)$. e. t. c.

Iteration 1 (k = 1)

The values of $F_1 (x, y)$ and $G_1 (x, y)$ are given in Tables 2.4a and 2.4c

y \ x	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	2	2	2	2	2	2	2	2	2	2	2	2	2	2
2	2	8	8	8	8	8	24	24	24	24	24	24	24	24
3	2	11	11	11	11	11	24	24	24	24	24	24	24	24
4	2	17	17	17	17	17	66	66	66	66	66	66	66	66
5	2	17	17	17	17	17	66	66	66	66	66	66	66	66
6	2	17	17	17	17	17	66	66	66	66	66	66	66	66
7	2	35	35	35	35	35	66	66	66	66	66	66	66	66
8	2	35	35	35	35	35	66	66	66	66	66	66	66	66
9	2	35	35	35	35	35	66	66	66	66	66	66	66	66
10	2	35	35	35	35	35	66	66	66	66	66	66	66	66

Table 2.3 Initial values of the DP recursion for the UGC problem of example 2.4.

respectively, for all $x \varepsilon L$ and $y \varepsilon W$. The length x' at which a cut was made on a rectangle (x, y) to produce $F_1(x, y)$ and the width y' at which a cut was made on a rectangle (x, y) to produce $G_1(x, y)$ are also presented in Tables 2.4b and 2 4d, respectively.

Iteration 2 (k = 2)

Values $F_2(x, y)$, $\Phi_2(x, y)$, $G_2(x, y)$ and $\Psi_2(x, y)$ are presented in Tables 2.5a - 2. 5d, respectively.

Iteration 3 (k = 3)

Values $F_3(x, y)$, $\Phi_3(x, y)$, $G_3(x, y)$ and $\Psi_3(x, y)$ are presented in Tables 2.6a - 2.6d, respectively.

Iteration 4 (k = 4)

Values $F_4(x, y)$, $\Phi_4(x, y)$, $G_4(x, y)$ and $\Psi_4(x, y)$ are presented in Tables 2.7a - 2.7d, respectively.

Iteration 5(k = 5)

Values $F_5(x, y)$, $\Phi_5(x, y)$, $G_5(x, y)$ and $\Psi_5(x, y)$ are presented in Tables 2.8a - 2.8d, respectively.

It can be seen from Tables 2.7a, 2.7c, 2.8a and 2.8c, that $F_5(15, 10) = F_4(15, 10)$ and $G_5(15, 10) = G_4(15, 10)$. Thus, no more iterations are performed and the optimal value is given by $F_4(15, 10) = G_4(15, 10) = 249$. By backtracking through the given tables ,we obtain the binary tree corresponding to the optimal cutting pattern of (15, 10) associated with the above value of 249 as shown in Fig. 2.8. The nodes in this tree are generated in the sequence indicated by the numbers written within the circles. The nature of the optimal pattern is determined by traversing the above tree in a " preordered" way. The structure of the pattern is given in Table 2.9 and a diagrammatic presentation of it in Fig. 2.9.

y \ x	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	2	2	4	4	6	6	8	8	10	10	12	12	14	14
2	2	8	8	10	16	16	24	24	26	32	32	34	40	40
3	2	11	11	13	22	22	24	33	33	35	44	44	46	55
4	2	17	17	19	34	34	66	66	68	83	83	85	100	100
5	2	17	17	19	34	34	66	66	68	83	83	85	100	100
6	2	17	17	19	34	34	66	66	68	83	83	85	100	100
7	2	35	35	37	70	70	72	105	105	107	140	140	142	175
8	2	35	35	37	70	70	72	105	105	107	140	140	142	175
9	2	35	35	37	70	70	72	105	105	107	140	140	142	175
10	2	35	35	37	70	70	72	105	105	107	140	140	142	175

Table 2.4a $F_1(x,y)$

y \ x	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	0	2	2	2	4	4	6	6	8	8	10	10	12	12
2	0	0	3	2	3	3	0	6	2	3	3	5	6	6
3	0	0	3	2	3	3	5	6	6	3	9	9	6	12
4	0	0	3	2	3	3	0	8	2	3	3	5	6	6
5	0	0	3	2	3	3	0	8	2	3	3	5	6	6
6	0	0	3	2	3	3	0	8	2	3	3	5	6	6
7	0	0	3	2	3	3	5	6	6	8	9	9	11	12
8	0	0	3	2	3	3	5	6	6	8	9	9	11	12
9	0	0	3	2	3	3	5	6	6	8	9	9	11	12
10	0	0	3	2	3	3	5	6	6	8	9	10	11	12

Table 2.4b $\Phi_1(x,y)$

y \ x	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	2	2	2	2	2	2	2	2	2	2	2	2	2	2
2	4	8	8	8	8	8	24	24	24	24	24	24	24	24
3	6	11	11	11	11	11	26	26	26	26	26	26	26	26
4	8	17	17	17	17	17	66	66	66	66	66	66	66	66
5	10	19	19	19	19	19	68	68	68	68	68	68	68	68
6	12	25	25	25	25	25	90	90	90	90	90	90	90	90
7	14	35	35	35	35	35	92	92	92	92	92	92	92	92
8	16	37	37	37	37	37	132	132	132	132	132	132	132	132
9	18	43	43	43	43	43	134	134	134	134	134	134	134	134
10	20	46	46	46	46	46	156	156	156	156	156	156	156	156

Table 2.4c $G_1(x,y)$

y \ x	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0	0	0	0	0
3	2	0	0	0	0	0	1	1	1	1	1	1	1	1
4	3	0	0	0	0	0	0	0	0	0	0	0	0	0
5	4	1	1	1	1	1	1	1	1	1	1	1	1	1
6	5	2	2	2	2	2	2	2	2	2	2	2	2	2
7	6	0	0	0	0	0	3	3	3	3	3	3	3	3
8	7	1	1	1	1	1	4	4	4	4	4	4	4	4
9	8	2	2	2	2	2	5	5	5	5	5	5	5	5
10	9	3	3	3	3	3	6	6	6	6	6	6	6	6

Table 2.4d $\Psi_1(x,y)$

y \ x	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	2	2	4	4	6	6	8	8	10	10	12	12	14	14
2	4	8	8	12	16	16	24	24	28	32	32	36	40	40
3	6	11	12	17	22	23	28	33	34	39	44	45	50	55
4	8	17	17	25	34	34	66	66	74	83	83	91	100	100
5	10	19	20	29	38	39	68	68	78	87	88	97	106	107
6	12	25	25	37	50	50	90	90	102	115	115	127	140	140
7	14	35	35	49	70	70	92	105	106	127	140	141	162	175
8	16	37	37	53	74	74	132	132	148	169	169	185	206	206
9	18	43	43	61	86	86	134	134	152	177	177	195	220	220
10	20	46	46	66	92	92	156	156	176	202	202	222	248	248

Table 2.5a $F_2(x,y)$

y \ x	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	0	2	2	2	4	4	6	6	8	8	10	10	12	12
2	0	0	2	2	3	3	0	6	2	3	3	5	6	6
3	0	0	2	2	3	4	5	6	7	8	9	10	11	12
4	0	0	3	2	3	3	0	8	2	3	3	5	6	6
5	0	0	2	2	3	4	0	8	2	3	4	5	6	7
6	0	0	3	2	3	3	0	8	2	3	3	5	6	6
7	0	0	3	2	3	3	0	6	2	3	9	5	6	12
8	0	0	3	2	3	3	0	8	2	3	3	5	6	6
9	0	0	3	2	3	3	0	8	2	3	3	5	6	6
10	0	0	3	2	3	3	0	8	2	3	3	5	6	6

Table 2.5b $\Phi_2(x,y)$

y \ x	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	2	2	4	4	6	6	8	8	10	10	12	12	14	14
2	4	8	8	10	16	16	24	24	26	32	32	34	40	40
3	6	11	12	14	22	22	32	33	36	42	44	46	54	55
4	8	17	17	20	34	34	66	66	68	83	83	85	100	100
5	10	19	21	24	40	40	74	74	78	93	95	97	114	114
6	12	25	25	30	50	50	90	90	94	115	115	119	140	140
7	14	35	35	37	70	70	98	105	105	125	140	140	154	175
8	16	37	39	41	76	76	132	132	136	166	166	170	200	200
9	18	43	43	47	86	86	140	140	146	176	178	182	214	215
10	20	46	47	51	92	92	156	156	162	198	198	204	240	240

Table 2.5c $G_2(x.y)$

y \ x	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	1	0	1	0	0	0	0	0	0	0	0	0	0	0
3	2	0	1	1	1	1	1	0	1	1	1	1	1	0
4	3	0	0	2	0	0	0	0	0	0	0	0	0	0
5	4	1	1	3	1	1	1	1	1	1	1	1	1	1
6	5	2	2	4	2	2	2	2	2	2	2	2	2	2
7	6	0	0	0	0	0	3	0	0	3	0	0	3	0
8	7	1	1	1	1	1	4	4	4	4	4	4	4	4
9	8	2	2	2	2	2	5	5	5	5	5	5	5	2
10	9	3	3	3	3	3	6	6	6	6	6	6	6	6

Table 2.5d $\Psi_2(x,y)$

y \ x	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	2	2	4	4	6	6	8	8	10	10	12	12	14	14
2	4	8	8	12	16	16	24	24	28	32	32	36	40	40
3	6	11	12	17	22	23	32	33	38	43	44	49	54	55
4	8	17	17	25	34	34	66	66	74	83	83	91	100	100
5	10	19	21	29	40	40	74	74	84	93	95	103	114	114
6	12	25	25	37	50	50	90	90	102	115	115	127	140	140
7	14	35	35	49	70	70	98	105	112	133	140	147	168	175
8	16	37	39	53	76	76	132	132	148	169	171	185	208	208
9	18	43	43	61	86	86	140	140	158	183	183	201	226	226
10	20	46	47	66	92	93	156	156	176	202	203	222	248	249

Table 2.6a $F_3(x,y)$

y \ x	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	0	2	2	2	2	2	2	2	2	2	2	2	2	2
2	0	0	2	2	3	3	0	3	2	3	3	2	3	3
3	0	0	2	2	3	3	0	3	2	3	3	5	6	3
4	0	0	3	2	3	3	0	8	2	3	3	2	3	3
5	0	0	0	2	0	3	0	8	2	3	4	2	6	3
6	0	0	3	2	3	3	0	8	2	3	3	2	3	3
7	0	0	3	2	3	3	0	3	2	3	3	5	6	3
8	0	0	0	2	0	3	0	8	2	3	4	5	6	6
9	0	0	3	2	3	3	0	8	2	3	3	5	6	6
10	0	0	0	2	3	3	0	8	2	3	4	5	6	7

Table 2.6b $\Phi_3(x,y)$

y \ x	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	2	2	4	4	6	6	8	8	10	10	12	12	14	14
2	4	8	8	12	16	16	24	24	28	32	32	36	40	40
3	6	11	12	17	22	23	32	33	38	42	44	48	54	55
4	8	17	17	25	34	34	66	66	74	83	83	91	100	100
5	10	19	21	29	40	40	74	74	84	93	95	103	114	114
6	12	25	25	37	50	50	90	90	102	115	115	127	140	140
7	14	35	35	49	70	70	98	105	112	127	140	141	162	175
8	16	37	39	53	76	76	132	132	148	169	169	185	206	206
9	18	43	43	61	86	86	140	140	158	179	181	197	220	220
10	20	46	47	66	92	93	156	156	176	202	202	222	248	248

Table 2.6c $G_3(x,y)$

y \ x	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	1	0	1	0	0	0	0	0	0	0	0	0	0	0
3	1	0	1	0	1	0	1	0	1	1	1	1	1	0
4	1	0	0	0	0	0	0	0	0	0	0	0	0	0
5	1	1	1	1	1	1	1	1	1	1	1	1	1	1
6	1	2	2	2	2	2	2	2	2	2	2	2	2	2
7	1	0	0	0	0	0	1	0	1	0	0	0	0	0
8	1	1	1	1	1	1	4	4	4	0	0	0	0	0
9	1	2	2	2	2	2	1	1	1	1	1	1	1	1
10	1	3	1	3	1	3	2	2	2	0	0	0	0	0

Table 2.6d $\Psi_3(x,y)$

y \ x	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	2	2	4	4	6	6	8	8	10	10	12	12	14	14
2	4	8	8	12	16	16	24	24	28	32	32	36	40	40
3	6	11	12	17	22	23	32	33	38	43	44	49	54	55
4	8	17	17	25	34	34	66	66	74	83	83	91	100	100
5	10	19	21	29	40	40	74	74	84	93	95	103	114	114
6	12	25	25	37	50	50	90	90	102	115	115	127	140	140
7	14	35	35	49	70	70	98	105	112	133	140	147	168	175
8	16	37	39	53	76	76	132	132	148	169	171	185	208	208
9	18	43	43	61	86	86	140	140	158	183	183	201	226	226
10	20	46	47	66	92	93	156	156	176	202	203	222	248	249

Table 2.7a $F_4(x,y)$

y \ x	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	0	2	2	2	2	2	2	2	2	2	2	2	2	2
2	0	0	2	2	3	2	0	3	2	3	2	2	3	2
3	0	0	2	2	3	2	0	3	2	3	2	3	6	3
4	0	0	3	2	3	3	0	8	2	3	3	2	3	3
5	0	0	0	2	0	3	0	8	2	3	4	2	6	3
6	0	0	3	2	3	3	0	8	2	3	3	2	3	3
7	0	0	3	2	3	3	0	3	2	3	3	3	6	3
8	0	0	0	2	0	3	0	8	2	3	4	2	6	4
9	0	0	3	2	3	3	0	8	2	3	3	3	6	6
10	0	0	0	2	3	3	0	8	2	3	4	2	3	4

Table 2.7b $\Phi_4(x,y)$

y \ x	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	2	2	4	4	6	6	8	8	10	10	12	12	14	14
2	4	8	8	12	16	16	24	24	28	32	32	36	40	40
3	6	11	12	17	22	23	32	33	38	43	44	49	54	55
4	8	17	17	25	34	34	66	66	74	83	83	91	100	100
5	10	19	21	29	40	40	74	74	84	93	95	103	114	114
6	12	25	25	37	50	50	90	90	102	115	115	127	140	140
7	14	35	35	49	70	70	98	105	112	133	140	147	168	175
8	16	37	39	53	76	76	132	132	148	169	171	185	208	208
9	18	43	43	61	86	86	140	140	158	183	183	201	226	226
10	20	46	47	66	92	93	156	156	176	202	203	222	248	249

Table 2.7c $G_4(x,y)$

y \ x	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	1	0	1	0	0	0	0	0	0	0	0	0	0	0
3	1	0	1	0	1	0	1	0	1	0	1	0	1	0
4	1	0	0	0	0	0	0	0	0	0	0	0	0	0
5	1	1	1	1	1	1	1	1	1	1	1	1	1	1
6	1	2	1	2	2	2	2	2	2	2	2	2	2	2
7	1	0	0	0	0	0	1	0	1	0	0	0	0	0
8	1	1	1	1	1	1	4	4	4	0	0	0	0	0
9	1	2	1	2	2	2	1	1	1	0	1	0	0	0
10	1	3	1	3	1	3	2	2	2	0	2	0	2	0

Table 2.7d $\Psi_4(x,y)$

y \ x	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	2	2	4	4	6	6	8	8	10	10	12	12	14	14
2	4	8	8	12	16	16	24	24	28	32	32	36	40	40
3	6	11	12	17	22	23	32	33	38	43	44	49	54	55
4	8	17	17	25	34	34	66	66	74	83	83	91	100	100
5	10	19	21	29	40	40	74	74	84	93	95	103	114	114
6	12	25	25	37	50	50	90	90	102	115	115	127	140	140
7	14	35	35	49	70	70	98	105	112	133	140	147	168	175
8	16	37	39	53	76	76	132	132	148	169	171	185	208	208
9	18	43	43	61	86	86	140	140	158	183	183	201	226	226
10	20	46	47	66	92	93	156	156	176	202	203	222	248	249

Table 2.8a $F_5(x,y)$

y \ x	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	0	2	2	2	2	2	2	2	2	2	2	2	2	2
2	0	0	2	2	3	2	0	3	2	3	2	2	3	2
3	0	0	2	2	3	2	0	3	2	3	2	2	3	2
4	0	0	3	2	3	3	0	8	2	3	3	2	3	3
5	0	0	0	2	0	3	0	8	2	3	4	2	6	3
6	0	0	3	2	3	3	0	8	2	3	3	2	3	3
7	0	0	3	2	3	3	0	3	2	3	3	2	3	3
8	0	0	0	2	0	3	0	8	2	3	4	2	6	3
9	0	0	3	2	3	3	0	8	2	3	3	2	3	3
10	0	0	0	2	3	3	0	8	2	3	4	2	3	3

Table 2.8b $\Phi_5(x,y)$

y \ x	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	2	2	4	4	6	6	8	8	10	10	12	12	14	14
2	4	8	8	12	16	16	24	24	28	32	32	36	40	40
3	6	11	12	17	22	23	32	33	38	43	44	49	54	55
4	8	17	17	25	34	34	66	66	74	83	83	91	100	100
5	10	19	21	29	40	40	74	74	84	93	95	103	114	114
6	12	25	25	37	50	50	90	90	102	115	115	127	140	140
7	14	35	35	49	70	70	98	105	112	132	140	147	168	175
8	16	37	39	53	76	76	132	132	148	169	171	185	208	208
9	18	43	43	61	86	86	140	140	158	183	183	201	226	226
10	20	46	47	66	92	93	156	156	176	202	203	222	248	249

Table 2.8c $G_5(x,y)$

y \ x	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	1	0	1	0	0	0	0	0	0	0	0	0	0	0
3	1	0	1	0	1	0	1	0	1	0	1	0	1	0
4	1	0	0	0	0	0	0	0	0	0	0	0	0	0
5	1	1	1	1	1	1	1	1	1	1	1	1	1	1
6	1	2	1	2	2	2	2	2	2	2	2	2	2	2
7	1	0	0	0	0	0	1	0	1	0	0	0	0	0
8	1	1	1	1	1	1	4	4	4	0	0	0	0	0
9	1	2	1	2	2	2	1	1	1	0	1	0	0	0
10	1	3	1	3	1	3	2	2	2	0	2	0	2	0

Table 2.8d $\Psi_5(x,y)$

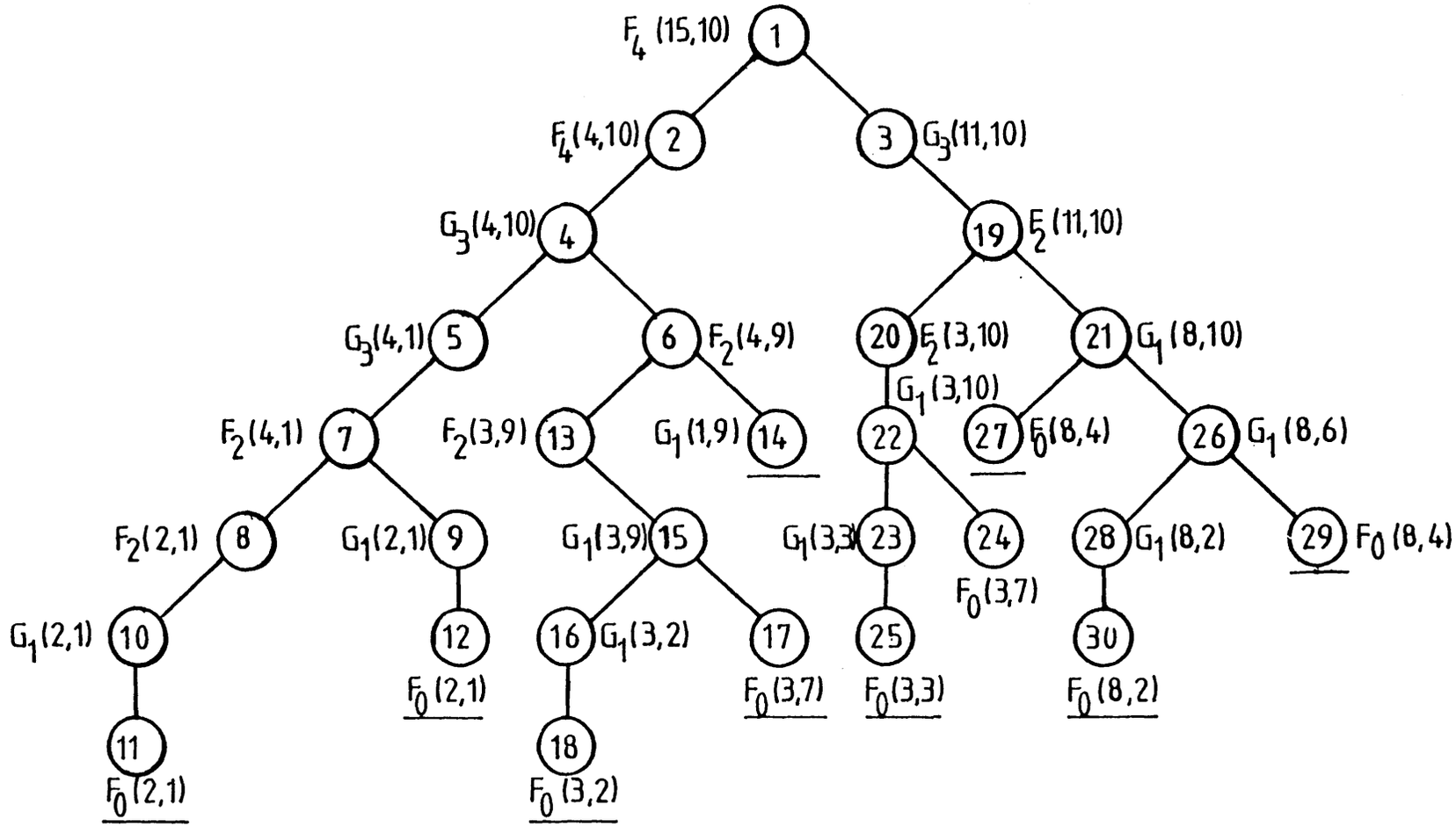


Figure 2.8 Tree - representation of optimal pattern for rectangle (15, 10) of example 2.4.

NR	XD	YD	CR	AR
1	15	10	4	0
2	4	10	-1	0
3	4	1	2	0
4	2	1	0	7
5	2	1	0	7
6	4	9	3	0
7	3	9	-2	0
8	3	2	0	6
9	3	7	0	2
10	11	10	3	0
11	3	10	-3	0
12	3	3	0	5
13	3	7	0	2
14	8	10	-6	0
15	8	6	-2	0
16	8	2	0	3
17	8	4	0	1
18	8	4	0	1

Table 2.9 Representation of the optimal cutting pattern for example 2.4.

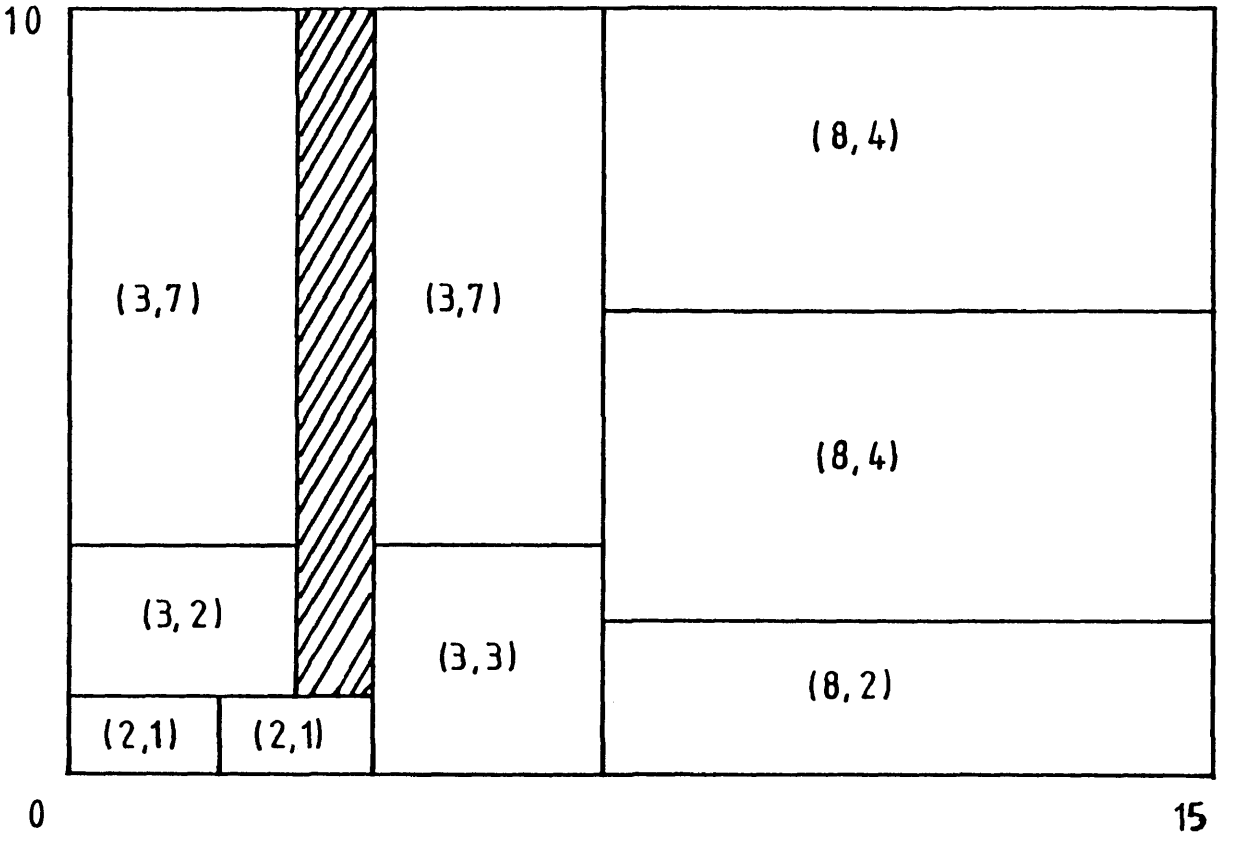


Figure 2.9 Optimal cutting pattern for example 2.4.

CHAPTER 3

AN ALGORITHM FOR THE TWO - DIMENSIONAL CONSTRAINED GUILLOTINE CUTTING (CGC) PROBLEM

3.1 Introduction

Dynamic Programming can be used to solve the Two - Dimensional Constrained Guillotine Cutting (CGC) Problem . This problem involves upper bounds on the maximum number of pieces of each size to be cut from a large stock rectangle. A dynamic programming procedure is then used, that introduces these constraints into additional state variables in the state vector.

The modified DP recursion, even for small size problems, requires too much storage and time. Thus, instead of obtaining an optimal solution to the problem, an upper bound is computed by solving the DP recursion on a smaller set of states. This corresponds to an idea recently developed and called " state - space relaxation " (SSR) in *Christofides et al* [1981a] where it is used for the vehicle

routing problem. State space relaxation is a generalisation of Lagrangean relaxation, and, hence, can be embedded in a tree - search procedure in order to solve optimally the original problem.

In this chapter, the application of SSR to the CGC problem is developed. " State Space Ascent " (SSA) methods are given for minimizing the resulting upper bound and are investigated computationally. A tree - search algorithm is then described for the solution of the CGC probem that uses the bound derived from SSR and improved by SSA methods. The computational performance of the algorithm is illustrated by tests performed on a number of randomly generated problems with constraints of varying " tightness ".

3.2 Definition of the Constrained Problem

The two - dimensional CGC problem can be defined as follows:

Let a large rectangle $A_0 = (\alpha_0, \beta_0)$ be given, together with a set $\bar{R}$ of m smaller rectangular pieces $\bar{R} = \{ (\alpha_1, \beta_1), (\alpha_2, \beta_2), ..., (\alpha_m, \beta_m) \}$, each piece in $\bar{R}$ having associated with it a value of υ_i and a maximum number Q_i of pieces of type i that can be cut from A_0. Let

$$M = \sum_{i=1}^{m} Q_i$$

be the total number of pieces in $\bar{R}$. The problem is to construct a guillotine cutting pattern for A_0 with the highest possible total value

$$z = \sum_{i=1}^{m} \xi_i \upsilon_i$$

so that ξ_i pieces of type i in $\bar{R}$ can be cut from A_0 where $\xi_i \leq Q_i$ for all i = 1, ..., m.

We will make the following assumptions for problem P:

(i) All dimensions (α_i, β_i) for i = 0, 1, ..., m are integers and the cuts on the rectangles (we refer to those pieces in set $\bar{R}$ that are produced by the cuts on A_0, at any stage during the cutting process, as " rectangles ") are to be made in integer steps along the x or y axes. Let L = { 1, 2, ..., α_0 - 1 } and W = { 1, 2, ..., β_0 - 1 } represent the sets of all possible lengths and widths, respectively, at which guillotine cuts can be performed on A_0.

(ii) The orientation of the pieces is considered to be fixed.

As it has been explained in chapter 2, the n-stage guillotine cutting problem, where the number of stages involved in the cutting process is unknown, can be regarded as a non-staged guillotine cutting problem, with the optimal value of the former being expected to attain the optimal value of the latter as n increases. Therefore, in order to solve the general CGC problem, we start by developing a dynamic programming recursion for the k-stage constrained cutting problem which is presented in the following section.

3.3 A Dynamic Programming (DP) Formulation of the CGC Problem

In this section, we will modify the basic DP recursion given in chapter 2 [equations (2) and (3)] for the k-stage unconstrained cutting problem in order to include the constraints on the number of pieces of type i in $\bar{R}$ that can be cut from

A_0 (for all $i = 1,..., m$).

Define a feasible set S_{xy} of rectangles produced by a guillotine cutting pattern for a rectangle (x, y) where $1 \le x \le \alpha_0$ and $1 \le y \le \beta_0$, as a subset of the set of all M pieces in $\bar{R}$ (i. e. $S_{xy} \subset \bar{R}$) by

$$S_{xy} = \{ i \mid \xi_i \le Q_i ;\ \alpha_i \le x,\ \beta_i \le y,\ i = 1,..., m \} \qquad (1)$$

Thus, the size (x, y) of a rectangle to be cut and an associated feasible set S_{xy} will correspond to a state vector in our DP formulation. For a state vector (x, y, S_{xy}) we define $F_k (x, y, S_{xy})$ as the maximum value of a k-stage cut of a rectangle of size (x, y) using any set $S' \subset S_{xy}$ of rectangles when the first-stage cut direction is parallel to the y-axis. Similarly, we define $G_k (x, y, S_{xy})$ as the maximum value of a k-stage cut of a rectangle (x, y) using any set $S' \subset S_{xy}$ of rectangles when the first-stage cut direction is parallel to the x-axis. The recursive function $F_k (x, y, S_{xy})$ can be stated as follows:

$$F_k (x, y, S_{xy}) = \max [G_{k-1} (x, y, S_{xy});\ \max_{x'<x,\ x' \varepsilon L,\ S' \subset S_{xy}} \{ F_k (x', y, S') + G_{k-1} (x-x', y, S_{xy} - S') \}] \text{ for } k \ge 1 \qquad (2)$$

This follows from the fact that the pattern yielding the value $F_k (x, y, S_{xy})$, $k \ge 1$, either does or does not perform a first-stage cut on a rectangle (x, y) parallel to the y-axis at some $x' \varepsilon L$. If such a first-stage cut is performed, then

$$F_k (x, y, S_{xy}) = \max_{x'<x,\ x' \varepsilon L,\ S' \subset S_{xy}} \{ F_k (x', y, S') + G_{k-1} (x-x', y, S_{xy}-S') \}.$$

If there are no first-stage cuts parallel to the y-axis but at least one second-stage cut parallel to the x-axis at some $y' \varepsilon W$ then $F_k (x, y, S_{xy}) = G_{k-1} (x, y, S_{xy})$. A similar argument to the one given above can be used to show that

$$G_k (x, y, S_{xy}) = \max [F_{k-1} (x, y, S_{xy}); \max_{y'<y,\, y'\varepsilon W,\, S'\subset S_{xy}} \{ G_k (x, y', S') + F_{k-1} (x, y-y', S_{xy}-S') \}] \quad \text{for } k \geq 1 \qquad (3)$$

Equations (2) and (3) are the basic dynamic programming recursions for the optimal k-stage constrained cutting of a rectangle (x, y) and they apply for any $k \geq 1$, $1 \leq x \leq \alpha_0$, $1 \leq y \leq \beta_0$ and S_{xy}. Initial conditions for the above recursion are provided by

$$F_0 (x, y, S_{xy}) = \max_i (v_i \mid i \varepsilon S_{xy} , i = 1, ..., m)$$

and

$$G_0 (x, y, S_{xy}) = F_0 (x, y, S_{xy})$$

for all $x \varepsilon L$, $y \varepsilon W$ and S_{xy}. Then $\max \{ F_n (\alpha_0, \beta_0, \bar{R}), G_n (\alpha_0, \beta_0, \bar{R}) \}$ gives us the value of an optimal n-stage cutting pattern for A_0 subject to the constraints on the number of rectangles produced, if the first-stage cut direction is unspecified. Note that when $F_n (\alpha_0, \beta_0, \bar{R})$ and $G_n (\alpha_0, \beta_0, \bar{R})$ have been computed, so have $F_k (x, y, S_{xy})$ and $G_k (x, y, S_{xy})$ for all $0 \leq k \leq n$, $1 \leq x \leq \alpha_0$, $1 \leq y \leq \beta_0$ and S_{xy}.

The computational problem encountered in the direct application of the above basic recursion to solving the CGC problem is the problem of dimensionality

of state variable S_{xy}. An optimum policy for cutting a rectangle (x,y) into a set S_{xy} of rectangles at the k^{th} stage of the cutting process consists of either performing a cut on (x,y) parallel to the x - or y - axis or performing no cut at that particular stage. However, the computer storage requirements increase rapidly since the third state variable S_{xy} of the basic recursion refers to combinations of rectangles that can be produced by cutting a rectangle (x,y), i.e. involves all subsets of S_{xy}. This explosive increase in the state space dimension is critical to the computational efficiency of the given DP recursion.

In the present chapter we are not concerned with the exact solution of this recursion but with associated recursions based on relaxations of the state variable S_{xy} in order to reduce the state space dimension of the dynamic program. Such a relaxation procedure is presented in the next section

3.4 State Space Relaxation for the CGC Problem

3.4.1 Definition

In this section, we develop a general relaxation procedure whereby the state - space associated with the DP recursion given for the CGC problem in section 3.3 [equations (2) and (3)] is relaxed in such a way that the solution to the relaxed recursion provides a bound which could be embedded in a branch - and - bound scheme for the solution of the CGC problem. This state space relaxation (SSR) method is analogous to Lagrangean relaxation in integer programming. A survey of this new methodology is given in *Christofides et al* [1981a] where valid state space relaxations are presented for the travelling salesman problem (TSP), the time

constrained TSP and vehicle routing problem (VRP). A more detailed survey of the above prodedure can be found in *Christofides et al* [1981b].

Let us consider then the DP formulation for the CGC problem defined by equations (2) and (3). Now, let $\Delta (S_{xy})$ denote the set of all possible states $S' \subset S_{xy}$ associated with the cutting of a rectangle (x, y). Let g (.) be the mapping function from the domain of (x, y, S_{xy}) to some other vector space (x, y, g (S_{xy})) and let Ω (g (S_{xy})) be a set satisfying the condition:

$$\text{if } S' \varepsilon \Delta (S_{xy}) \quad \text{then} \quad g (S') \varepsilon \Omega (g (S_{xy})) \tag{4}$$

Recursion (2) and (3) can now be written as:

$$F_k (x, y, g(S_{xy})) = \max [G_{k-1}(x, y, g(S_{xy})); \max_{\substack{x'<x,\ x' \varepsilon L \\ g(S') \varepsilon \Omega (g(S_{xy}))}} \{ F_k (x', y, g(S')) + G_{k-1} (x-x', y, g(S_{xy}-S')) \}] \quad \forall k \geq 1 \tag{5}$$

$$G_k (x, y, g(S_{xy})) = \max [F_{k-1} (x, y, g(S_{xy})); \max_{\substack{y'<y,\ y' \varepsilon W \\ g(S') \varepsilon \Omega (g(S_{xy}))}} \{ G_k (x, y', g(S')) + F_{k-1} (x, y-y', g(S_{xy}-S')) \}] \quad \forall k \geq 1 \tag{6}$$

where

$$S' \varepsilon \Delta (S_{xy}) \tag{7}$$

From the above it is clear that :

$$F_k (x, y, g(S_{xy})) \leq F_k (x, y, S_{xy})$$

$$G_k (x, y, g(S_{xy})) \leq G_k (x, y, S_{xy}) \qquad (8)$$

for all $x \varepsilon L$, $y \varepsilon W$ and $0 \leq k \leq n$. Note that max { $F_n (x, y, g(S_{xy}))$, $G_n (x, y, g(S_{xy}))$ } (from (8) above) can produce bounds which can be embedded into any tree search for solving the original CGC problem.

The state - space relaxation which produced recursion (5) and (6) is useful only if the function $g (.)$ is such that set $\Omega (g (S_{xy}))$ can be computed easily from (4) above. This condition is satisfied if $g (.)$ is chosen to be a separable function, so that given $g(S_{xy})$ and $g(S')$, $g(S_{xy} - S')$ can be computed. In this case, (5) and (6) become:

$$F_k (x, y, g(S_{xy})) = \max [G_{k-1}(x, y, g(S_{xy})); \max_{\substack{x'<x,\ x'\varepsilon L \\ t \varepsilon \Omega (g(S_{xy}))}} \{ F_k (x', y, t) + G_{k-1} (x-x', y, g(S_{xy}) - t) \}] \quad \forall k \geq 1 \qquad (9)$$

$$G_k (x, y, g(S_{xy})) = \max [F_{k-1} (x, y, g(S_{xy})); \max_{\substack{y'<y,\ y'\varepsilon W \\ t \varepsilon \Omega (g(S_{xy}))}} \{ G_k (x, y', t) + F_{k-1} (x, y-y', g(S_{xy}) - t) \}] \quad \forall k \geq 1 \qquad (10)$$

where $t = g(S')$ and $S' \varepsilon \Delta (S_{xy})$.

In the following section, we will illustrate some possible useful forms of the function g (.).

A diagrammatic illustration of state - space relaxation is shown in Fig. 3.1.

3.4.2 Forms of the Mapping Function g (.)

As suggested by Fig. 3.1, the state - space relaxation is defined by g and Ω. The function g can be any separable function. A suitable form of the mapping function is presented below:

$$g(S_{xy}) = \sum_{i \in S_{xy}} q_i \xi_i \quad (\text{q - path relaxation}) \text{ S S R}$$

Let us associate a non - negative integer weight q_i with every piece i in set $\tilde{R}$ (i = 1, ..., m) and define

$$g(S_{xy}) = q = \sum_{i \in S_{xy}} \xi_i q_i.$$

Equations (9) and (10) are then modified as follows:

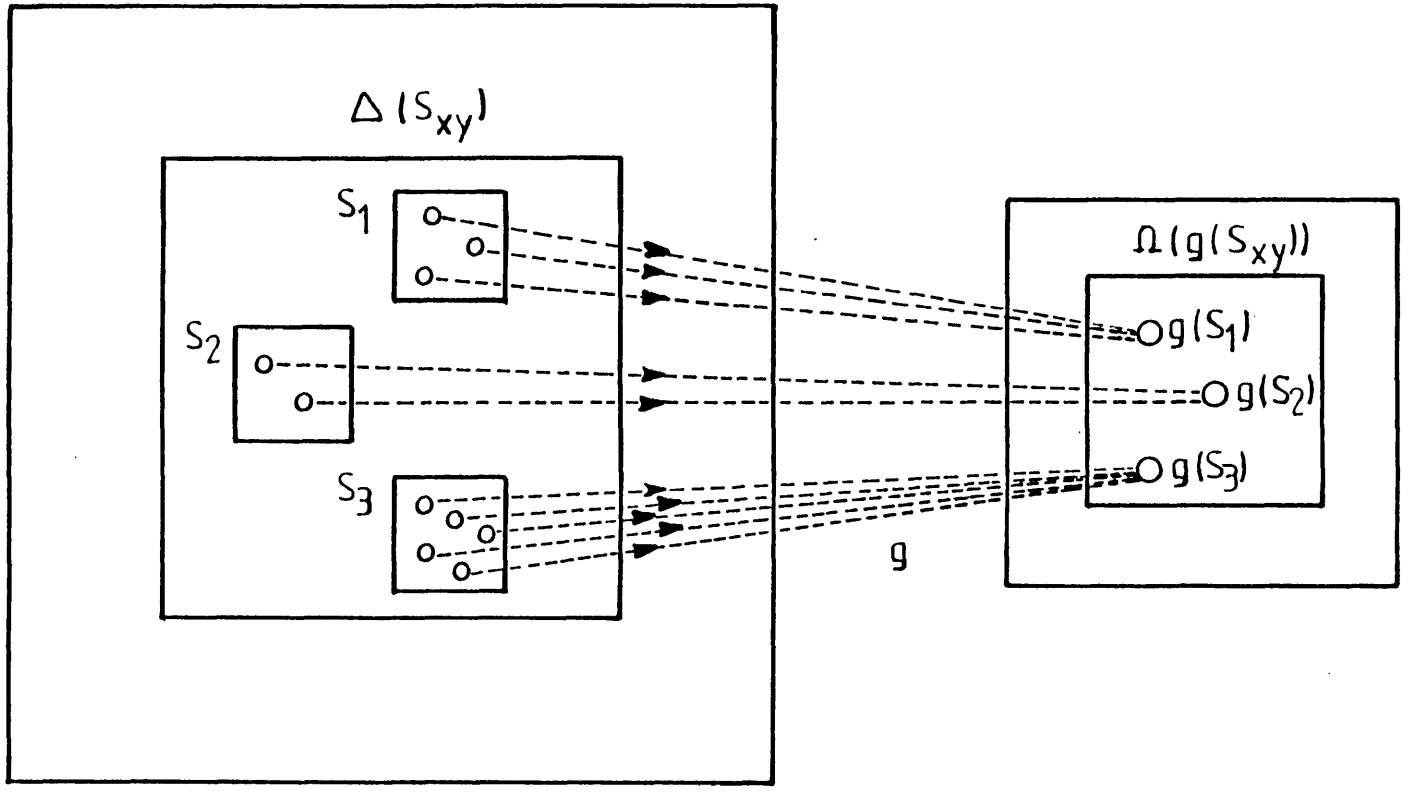


Figure 3.1 Graphical presentation of State-Space Relaxation

$$F_k(x, y, q) = \max [\, G_{k-1}(x, y, q);\ \max_{\substack{x'<x,\ x'\varepsilon L \\ q'=0,\ \ldots,\ q}} \{ F_k(x', y, q') + G_{k-1}(x-x', y, q-q') \} \,] \quad \forall k \geq 1 \qquad (14)$$

$$G_k(x, y, q) = \max [\, F_{k-1}(x, y, q);\ \max_{\substack{y'<y,\ y'\varepsilon W \\ q'=0,\ \ldots,\ q}} \{ G_k(x, y', q') + F_{k-1}(x, y-y', q-q') \} \,] \quad \forall k \geq 1 \qquad (15)$$

and initialised by

$$F_0(x, y, q) = \max_i (\upsilon_i \mid \alpha_i \leq x, \beta_i \leq y, q_i \leq q,\ i = 1, \ldots, m)$$

$$G_0(x, y, q) = F_0(x, y, q) \qquad (16)$$

The above recursion applies for all $x \varepsilon L$, $y \varepsilon W$ and $q = 0, \ldots, Q$ where

$$Q = \sum_{i=1}^{m} q_i Q_i.$$

3.5 A Bound from State Space Relaxation (SSR)

It is clear from (8) that the state space relaxations of the DP recursion of the CGC problem can be used to obtain upper bounds on the value of the solution to this problem. In this section, we will describe how such a bound can be derived

from the relaxed recursion given by equations (14) to (16) [SSR].

We must note that $Z_{UB} = \max [(F_n (\alpha_0, \beta_0, Q) , G_n (\alpha_0, \beta_0, Q)]$ represents the optimal value of an n-stage cutting pattern for the stock rectangle A_0 generated by using pieces in set $\bar{R}$ when the first - stage cut direction is unspecified. (Note that Q is the maximum value that state variable q can take relative to a given set of weights q_i 's). The value Z_{UB} provides an upper bound on the solution of the CGC problem and is obtained from the relaxed DP recursion given by equations (14) to (16), this recursion being enhanced computationally in the following way :

Let sets L and W of possible lengths and widths for any cuts to be performed on A_0 be modified as described by equations (4) and (5) of chapter 2, respectively, to represent the normal sets relative to the set of pieces in $\bar{R}$ available for cutting. Sets L and W can be reduced even further for the constrained problem, by limiting the normal cuts to be performed at any point x or y, such that there exists a feasible set of pieces in $\bar{R}$ whose lengths add up to x, or whose widths add up to y, respectively; namely, these sets are redefined by:

$$L = \{ x \mid x = \sum_{i=1}^{m} \theta_i \alpha_i,\ 1 \le x \le \alpha_0,\ 0 \le \theta_i \le Q_i \quad \text{and} \quad \theta_i \text{ integer } \forall\ i = 1, ..., m \}$$

and

$$W = \{ y \mid y = \sum_{i=1}^{m} t_i \beta_i,\ 1 \le y \le \beta_0,\ 0 \le t_i \le Q_i \quad \text{and} \quad t_i \text{ integer } \forall\ i = 1, ..., m \}$$

Also, let $l(x)$ and $w(y)$ be defined by equations (6) and (7) of chapter 2, respectively. Using the normality property (section 2.3.1 of chapter 2), we claim that we may calculate $F_k(l(x), w(y), q)$ instead of $F_k(x, y, q)$, since $F_k(l(x), w(y), q) = F_k(x, y, q)$. Equation (14) then becomes:

$$F_k(x, y, q) = \max [G_{k-1}(x, y, q); \max_{\substack{x'<x,\ x'\varepsilon L \\ q'=0,\ \ldots,\ q}} \{ F_k(x', y, q') + G_{k-1}(l(x-x'), y, q-q') \}] \quad \forall k \geq 1 \qquad (17)$$

Here, we have modified the second term in the recursion, arising from the optimal k - stage cutting pattern of (x, y) with a first - stage cut being made at some $x' \varepsilon L$, so that the pattern for the two resulting rectangles of sizes (x', y) and $(x - x', y)$ can be normalised. In a similar fashion, we modify equation (15) to be:

$$G_k(x, y, q) = \max [F_{k-1}(x, y, q); \max_{\substack{y'<y,\ y'\varepsilon W \\ q'=0,\ \ldots,\ q}} \{ G_k(x, y', q') + F_{k-1}(x, w(y-y'), q-q') \}] \quad \forall k \geq 1 \qquad (18)$$

It is now clear that $F_k(x, y, q)$ and $G_k(x, y, q)$ are calculated only for those values of x and y that belong to the normalised sets L and W respectively (i. e. $x \varepsilon L$ and $y \varepsilon W$) and for all values of q (i.e. $q = 0, 1, \ldots, Q$). Equations (17) and (18) form the relaxed DP recursions used to obtain the value of bound Z_{UB} relative to a given set of q_i 's. Initialisation conditions for the recursions are given by equation (16).

3.5.1 The Dynamic Programming Procedure

Once the value of bound Z_{UB} is found by solving the relaxed DP recursion given by equations (16) to (18) in section 3.5, backtracking to discover the nature of the associated cutting pattern is necessary. The full sets of maximum value functions and their corresponding optimal decisions are stored as soon as they are computed and retrieved at the time of backtracking. Thus, six matrices are used, namely, F_k (x, y, q), G_k (x, y, q), Φ_k (x, y, q), γ_k (x, y, q), Ψ_k (x, y, q) and δ_k (x, y, q). When the computation is completed F_k (x, y, q) = F_n (α_0, β_0, Q) and G_k (x, y, q) = G_n (α_0, β_0, Q). The four remaining matrices are defined as follows :

$$\Phi_k (x, y, q) = x',\ \gamma_k (x, y, q) = q' \text{ if } F_k (x, y, q) = F_k (x', y, q') + G_{k-1} (l (x-x'), y, q-q')$$

$$\Phi_k (x, y, q) = 0,\ \gamma_k (x, y, q) = 0 \quad \text{otherwise} \qquad (19)$$

and

$$\Psi_k (x, y, q) = y',\ \delta_k (x, y, q) = q' \text{ if } G_k (x, y, q) = G_k (x, y', q') + F_{k-1} (x, w (y-y'), q-q')$$

$$\Psi_k (x, y, q) = 0,\ \delta_k (x, y, q) = 0 \quad \text{otherwise} \qquad (20)$$

To help in describing the procedure for calculating Z_{UB} and finding the associated cutting pattern, let $x_1, x_2, ..., x_u$ be the elements of the normalised set L in order of increasing lengths and $y_1, y_2, ...y_v$ be the elements of W in order of increasing widths.

Initialisation

1. 1 Set $k = 0$.

1. 2 Set $F_0(x_s, y_t, q) = G_0(x_s, y_t, q) = \max_i (v_i \mid \alpha_i \le x_s, \beta_i \le y_t, q_i \le q,$ $i = 1, ..., m)$ for all $s = 1, ..., u$, $t = 1, ..., v$ and $q = 0, ..., Q$.

1. 3 Set $k = 1$.

1. 4 Set $s = 1$, $t = 1$, $q = 0$.

First - stage cut parallel to the y - axis.

2. 1 Set $s' = 1$.

2. 2 Set $q' = 0$.

2. 3 If $q' \le q$ and $F_k(x_s, y_t, q) < F_k(x_{s'}, y_t, q') + G_{k-1}(\mid (x_s - x_{s'}), y_t, q - q')$ then set $F_k(x_s, y_t, q) = F_k(x_{s'}, y_t, q') + G_{k-1}(\mid (x_s - x_{s'}), y_t, q - q')$, $\gamma_k(x_s, y_t, q) = q'$ and go to 2. 4. If $q' \le q$ then go to 2. 4 ; else go to 2. 5.

2. 4 Set $q' = q' + 1$ and go to 2. 3.

2. 5 Set $q_{max} = \max(0, \gamma_k(x_s, y_t, q))$.

2. 6 If $s' < s$ and $F_k(x_s, y_t, q) < F_k(x_{s'}, y_t, q_{max}) + G_{k-1}(\mid (x_s - x_{s'}), y_t, q - q_{max})$ then set $F_k(x_s, y_t, q) = F_k(x_{s'}, y_t, q_{max}) + G_{k-1}(\mid (x_s - x_{s'}), y_t, q - q_{max})$, $\Phi_k(x_s, y_t, q) = x_{s'}$ and go to 2. 7. If $s' < s$ then go to 2. 7 ; else go to 2. 8.

2. 7 Set $s' = s' + 1$ and go to 2. 6.

2. 8 If $G_{k-1}(x_s, y_t, q) < F_k(x_s, y_t, q)$ go to 2. 9 ; else set $F_k(x_s, y_t, q) = G_{k-1}(x_s, y_t, q)$, $\Phi_k(x_s, y_t, q) = 0$, $\gamma_k(x_s, y_t, q) = 0$ and go to 2. 9.

2. 9 If $q < Q$ then set $q = q + 1$ and go to 2. 1 ; else if $s < u$ then set $s = s + 1$, $q = 0$ and go to 2. 1 ; else if $t < v$ then set $t = t + 1$, $s = 1$, $q = 0$ and go to 2. 1 ; else go to 3. 1.

First - stage cut parallel to the x-axis.

3. 1 Set $s = 1$, $t = 1$, $q = 0$.

3. 2 Set $t' = 1$, $q' = 0$.

3. 3 If $q' < q$ and $G_k(x_s, y_t, q) < G_k(x_s, y_{t'}, q') + F_{k-1}(x_s, w(y_t - y_{t'}), q - q')$ then set $G_k(x_s, y_t, q) = G_k(x_s, y_{t'}, q') + F_{k-1}(x_s, w(y_t - y_{t'}), q - q')$, $\delta_k(x_s, y_t, q) = q'$ and go to 3. 4. If $q' \leq q$ then go to 3. 4 ; else go to 3. 5.

3. 4 Set $q' = q' + 1$ and go to 3. 3.

3. 5 Set $q_{max} = \max(0, \delta_k(x_s, y_t, q))$.

3. 6 If $t' < t$ and $G_k(x_s, y_t, q) < G_k(x_s, y_{t'}, q_{max}) + F_{k-1}(x_s, w(y_t - y_{t'}), q - q_{max})$, then set $G_k(x_s, y_t, q) = G_k(x_s, y_{t'}, q_{max}) + F_{k-1}(x_s, w(y_t - y_{t'}), q - q_{max})$, $\Psi_k(x_s, y_t, q) = y_{t'}$ and go to 3. 7. If $t' < t$ then go to 3. 7 ; else go to 3. 8.

3. 7 Set $t' = t' + 1$ and go to 3. 6.

3. 8 If $F_{k-1}(x_s, y_t, q) < G_k(x_s, y_t, q)$ go to 3. 9 ; else set $G_k(x_s, y_t, q) = F_{k-1}(x_s, y_t, q)$, $\Psi_k(x_s, y_t, q) = 0$, $\delta_k(x_s, y_t, q) = 0$ and go to 3. 9.

3. 9 If $q < Q$ then set $q = q + 1$ and go to 3. 2 ; else if $t < v$ then set $t = t + 1$, $q = 0$ and go to 3. 2 ; else if $s < u$ then set $s = s + 1$, $t = 1$, $q = 0$ and go to 3. 2 ; else go to 4. 1.

End of optimal k - stage cutting

4. 1 If $F_k(x_s, y_t, q) = F_{k-1}(x_s, y_t, q)$ and $G_k(x_s, y_t, q) = G_{k-1}(x_s, y_t, q)$ then set $n = k - 1$, value of bound $Z_{UB} = \max(F_n(\alpha_0, \beta_0, Q), G_n(\alpha_0, \beta_0, Q))$ and stop ; else set $k = k + 1$ and go to 1. 4.

At the end of the above procedure, $\max(F_n(\alpha_0, \beta_0, Q), G_n(\alpha_0, \beta_0, Q))$ is the value of bound Z_{UB} corresponding to the optimal n - stage guillotine cutting of A_0 relative to a given set of weights (q_i 's) associated with the pieces in set $\bar{R}$.

Note that values $F_k(x_s, y_t, q)$ and $G_k(x_s, y_t, q)$ have also been calculated for all $0 \leq k \leq n$, $s = 1, 2, ..., u$, $t = 1, 2, ..., v$ and $q = 0, 1, ..., Q$.

$\Phi_k(x, y, q)$ and $\gamma_k(x, y, q)$ can then be used to indicate for each set (x, y, q) the terms in the relaxed recursion that led to the value $F_k(x, y, q)$. Similarly, $\Psi_k(x, y, q)$ and $\delta_k(x, y, q)$ can be used to determine how $G_k(x, y, q)$ is achieved for any set (x, y, q). Thus, discovering the nature of an optimal k - stage pattern requires the use of a binary tree, having a similar structure to the tree that has been used by the backtracting procedure of section 2. 3. 2 (See Fig. 2.7). The structure of this tree is described by the following three characteristics :

(i) A node labelled $F_k(x, y, q)$ or $G_k(x, y, q)$ with exactly two nodes immediately below it represent a k - stage cut made on rectangle (x, y) parallel to the y - or the x - axis, respectively, using a set of rectangles S_{xy} in $\bar{R}$ such that

$$\sum_{i \in S_{xy}} q_i \xi_i = q.$$

(ii) A node labelled $F_k(x, y, q)$ or $G_k(x, y, q)$ with only a left node below it, labelled $G_{k-1}(x, y, q)$ or $F_{k-1}(x, y, q)$ respectively, imply that no cut is performed on rectangle (x, y) at the k^{th} stage of the cutting process.

(iii) A terminal node (i.e. a node that has no other nodes below it in the tree) labelled $F_k(x, y, q)$ or $G_k(x, y, q)$ represents a rectangle (x, y) that is not cut any further and a piece j in $\bar{R}$ can be allocated to it such that $j = \max_i \{ v_i \mid \alpha_i \leq x, \beta_i \leq y, q_i \leq q, i = 1, ..., m \}$.

The 3 - stage cutting pattern for the rectangle (x, y), shown in Fig. 3.2, associated with the optimal value F_k (x, y, q) can be represented by the binary tree shown in Fig. 3.3. A convenient way for describing this tree is presented in Table 2.1 of Chapter 2.

A " preorder " traversal of the binary tree is used to determine the sequence of cuts performed on A_0. A data structure, as described in section 2.3.2, is employed to represent the list of rectangles produced during the cutting process. Two additional entries are now included for each rectangle r_i in the list which are given below :

(i) q_i indicates the " weight " of the set of pieces in $\bar{R}$ used to cut rectangle r_i

(ii) q_i' is the " weight " of a subset of the above set of pieces, associated with the cutting of rectangle r_i into two further rectangles ($q_i' \leq q_i$).

The pattern shown in Fig. 3.2 would then be represented by the list of rectangles shown in Table 3.1, using NR to head the column corresponding to the number of rectangle. The list begins with cutting A_0. In this list, every rectangle is either cut into smaller rectangles or filled by one of the initial pieces given in $\bar{R}$ (waste is not cut out by the relaxed DP procedure).

Generating the above list of rectangles, we obtain the number c_i of pieces of type i that have been used by the DP solution in cutting A_0. If $c_i \leq Q_i$ for all types i in $\bar{R}$ (i. e. i = 1, ..., m) (i.e. a feasible solution is found) then the optimal solution to the CGC problem has been obtained. Otherwise, the value Z_{UB} of the associated n - stage pattern, serves as an upper bound on the solution of the original problem.

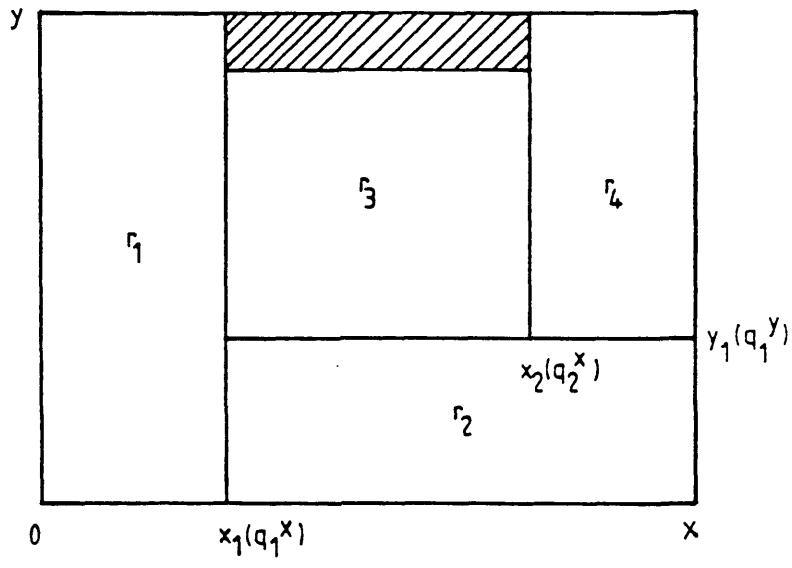


Figure 3.2 An optimal 3-stage cutting pattern of rectangle (x,y)

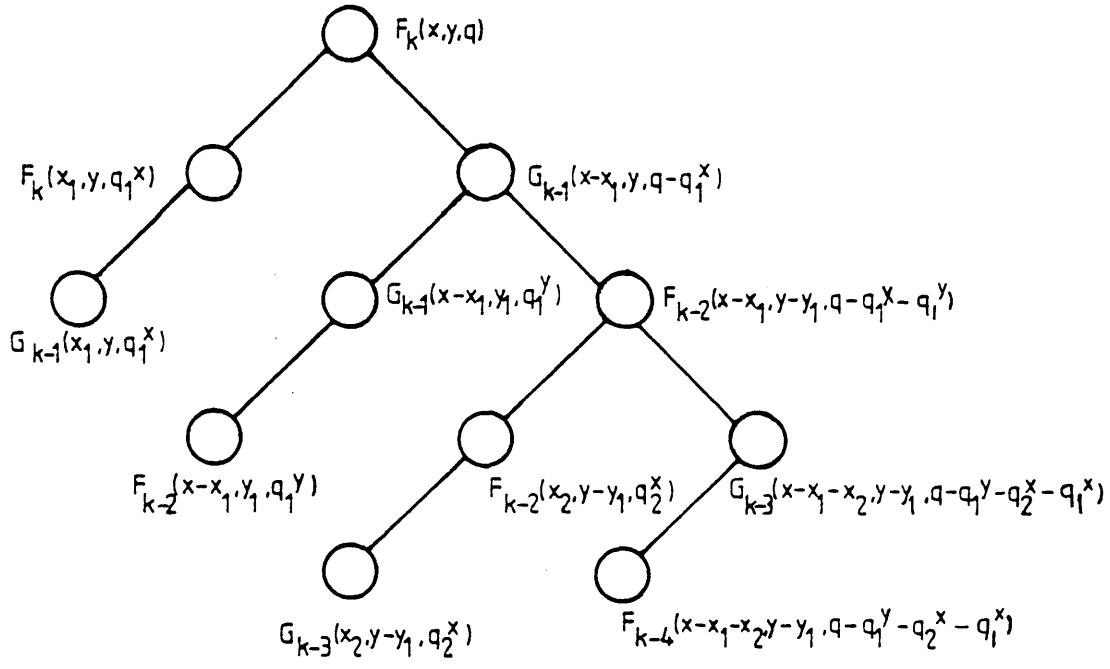


Figure 3.3 Tree representation of pattern shown in Figure 3.2

NR	XD	YD	q	CR	q'	AR
1	x	y	q	x_1	q_1^x	0
2	x_1	y	q_1^x	0	0	r_1
3	$(x-x_1)$	y	$(q-q_1^x)$	$-y_1$	q_1^y	0
4	$(x-x_1)$	y_1	q_1^y	0	0	r_2
5	$(x-x_1)$	$(y-y_1)$	$(q-q_1^y-q_1^x)$	x_2	q_2^x	0
6	x_2	$(y-y_1)$	q_2^x	0	0	r_3
7	$(x-x_1-x_2)$	$(y-y_1)$	$(q-q_1^x-q_1^y-q_2^x)$	0	0	r_4

Table 3.1 Representation of pattern shown in Figure 3.2

3.5.2 An Example

Consider the CGC problem whose set $\bar{R}$ of pieces available to be cut from a stock rectangle A_0 of size (10, 10) is as given below :

piece i	size i	value i	constraint i
1	(2, 2)	5	1
2	(5, 3)	15	3
3	(6, 7)	52	2
4	(4, 7)	44	2
5	(2, 4)	12	1

We will use the state - space relaxation [(equations (16) - (18)] to compute an upper bound to the value of the optimal solution to this CGC problem. Let us choose (arbitrarily) a set of weights q_i to apply to the pieces i = 1, ..., 5. Let these weights be given by :

piece i	1	2	3	4	5
q_i	0	0	0	0	1

and hence

$$Q = \sum_{i=1}^{5} Q_i \, q_i = 1.$$

The normalised sets of lengths and widths are given by L = { 2, 4, 5, 6, 7, 8, 9,10} and W = { 2, 3, 4, 5, 6, 7, 8, 9,10}respectively. From (16) the

initialisation is given in Table 3.2.

We will use the value of k to index the iterations of recursion (17) and (18) i. e. we will call iteration 2 the iteration which computes all F_2 (x, y, q) and G_2 (x, y, q), iteration 3 the iteration which computes all F_3 (x, y, q) and G_3 (x, y, q) e.t.c.

Thus, four iterations (n = 4) are performed until F_4 (10, 10, 1) = F_3 (10, 10, 1) and G_4 (10, 10, 1) = G_3 (10, 10, 1) providing us with a value of Z_{UB} = 135 (= F_3 (10, 10, 1)=(G_3 (10, 10, 1)). The values of F_k (x, y, q), Φ_k (x, y, q) and γ_k (x, y, q), as defined in section 3.5.1, for all values of x ε L, y εW and q = 0, 1 are presented in Tables 3.3, 3.5 and 3.7 for k = 1, 2 and 3 respectively. Similarly, the values of G_k (x, y, q), Ψ_k (x, y, q) and δ_k (x, y, q) are given in Tables 3.4, 3.6 and 3.8 for k = 1, 2 and 3 respectively. By backtracking through the given tables we obtain the cutting pattern of (10, 10) associated with the above value of 135. The structure of this pattern is shown in Table 3.9. (Note that the backtracking procedure, as described in section 3.5.1, starts with the value G_3 (10, 10, 1)) and a diagrammatic presentation of it in Fig. 3. 4.

Generating the above pattern for the given CGC problem, a list of rectangles is produced given by { 7, 0, 0, 2, 1 }. Clearly, this is not a feasible solution to the problem since the number of pieces of type 1 that have been used by the current DP solution in cutting A_0 (c_1 = 7) exceeds the corresponding availability constraint for this type (Q_1 = 1). Thus, Z_{UB} = 135 is an upper bound on the solution of the problem. Note that this value happens to be the optimal, as shown in the Table 3.10 of computational results of section 3.6.3 (Problem 1).

y \ x	2		4		5		6		7		8		9		10	
q	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
2	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
3	5	5	15	15	15	15	15	15	15	15	15	15	15	15	15	15
4	5	12	5	12	15	15	15	15	15	15	15	15	15	15	15	15
5	5	12	5	12	15	15	15	15	15	15	15	15	15	15	15	15
6	5	12	5	12	15	15	15	15	15	15	15	15	15	15	15	15
7	5	12	44	44	44	44	52	52	52	52	52	52	52	52	52	52
8	5	12	44	44	44	44	52	52	52	52	52	52	52	52	52	52
9	5	12	44	44	44	44	52	52	52	52	52	52	52	52	52	52
10	5	12	44	44	44	44	52	52	52	52	52	52	52	52	52	52

Table 3.2 Initial Values of the DP relaxed recursion for the CGC Problem of Example 3.5.2.

y \ x		2			4			5			6			7			8			9			10		
	q	F	Φ	Υ	F	Φ	Υ	F	Φ	Υ	F	Φ	Υ	F	Φ	Υ	F	Φ	Υ	F	Φ	Υ	F	Φ	Υ
2	0	5	0	0	10	2	0	10	2	0	15	4	0	15	4	0	20	6	0	20	6	0	25	8	0
	1	5	0	0	10	2	0	10	2	0	15	4	0	15	4	0	20	6	0	20	6	0	25	8	0
3	0	5	0	0	10	2	0	15	0	0	15	4	0	20	2	0	20	2	0	25	4	0	30	5	0
	1	5	0	0	10	2	0	15	0	0	15	4	0	20	2	0	20	2	0	25	4	0	30	5	0
4	0	5	0	0	10	2	0	15	0	0	15	4	0	20	2	0	20	2	0	25	4	0	30	5	0
	1	12	0	0	17	2	0	17	2	0	22	4	0	27	2	1	27	2	1	32	4	1	32	4	1
5	0	5	0	0	10	2	0	15	0	0	15	4	0	20	2	0	20	2	0	25	4	0	30	5	0
	1	12	0	0	17	2	0	17	2	0	22	4	0	27	2	1	27	2	1	32	4	1	32	4	1
6	0	5	0	0	10	2	0	15	0	0	15	4	0	20	2	0	20	2	0	25	4	0	30	5	0
	1	12	0	0	17	2	0	17	2	0	22	4	0	27	2	1	27	2	1	32	4	1	32	4	1
7	0	5	0	0	44	0	0	44	4	0	52	0	0	52	6	0	88	4	0	88	4	0	96	4	0
	1	12	0	0	44	0	0	44	4	0	56	2	1	56	2	1	88	4	0	88	4	0	100	6	1
8	0	5	0	0	44	0	0	44	4	0	52	0	0	52	6	0	88	4	0	88	4	0	96	4	0
	1	12	0	0	44	0	0	44	4	0	56	2	1	56	2	1	88	4	0	88	4	0	100	6	1
9	0	5	0	0	44	0	0	44	4	0	52	0	0	52	6	0	88	4	0	88	4	0	96	4	0
	1	12	0	0	44	0	0	44	4	0	56	2	1	56	2	1	88	4	0	88	4	0	100	6	1
10	0	5	0	0	44	0	0	44	4	0	52	0	0	52	6	0	88	4	0	88	4	0	96	4	0
	1	12	0	0	44	0	0	44	4	0	56	2	1	56	2	1	88	4	0	88	4	0	100	6	1

Table 3.3 $F_1(x,y,q)$, $\Phi_1(x,y,q)$, $\gamma_1(x,y,q)$ $\forall$ x,y,q

x \ y		2			3			4			5			6			7			8			9			10		
	q	G	Ψ	δ	G	Ψ	δ	G	Ψ	δ	G	Ψ	δ	G	Ψ	δ	G	Ψ	δ	G	Ψ	δ	G	Ψ	δ	G	Ψ	δ
2	0	5	0	0	5	2	0	10	2	0	10	2	0	15	4	0	15	4	0	20	6	0	20	6	0	25	8	0
	1	5	0	0	5	2	0	12	0	0	12	4	1	17	2	0	17	2	0	22	4	0	22	4	0	27	6	0
4	0	5	0	0	5	2	0	10	2	0	10	2	0	15	4	0	44	0	0	44	7	0	49	2	0	49	2	0
	1	5	0	0	5	2	0	12	0	0	12	4	1	17	2	0	44	0	0	44	7	0	49	2	0	49	2	0
5	0	5	0	0	15	0	0	15	3	0	20	2	0	30	3	0	44	0	0	44	7	0	49	2	0	59	3	0
	1	5	0	0	15	0	0	15	3	0	20	2	0	30	3	0	44	0	0	44	7	0	49	2	0	59	3	0
6	0	5	0	0	15	0	0	15	3	0	20	2	0	30	3	0	52	0	0	52	7	0	57	2	0	67	3	0
	1	5	0	0	15	0	0	15	3	0	20	2	0	30	3	0	52	0	0	52	7	0	57	2	0	67	3	0
7	0	5	0	0	15	0	0	15	3	0	20	2	0	30	3	0	52	0	0	52	7	0	57	2	0	67	3	0
	1	5	0	0	15	0	0	15	3	0	20	2	0	30	3	0	52	0	0	52	7	0	57	2	0	67	3	0
8	0	5	0	0	15	0	0	15	3	0	20	2	0	30	3	0	52	0	0	52	7	0	57	2	0	67	3	0
	1	5	0	0	15	0	0	15	3	0	20	2	0	30	3	0	52	0	0	52	7	0	57	2	0	67	3	0
9	0	5	0	0	15	0	0	15	3	0	20	2	0	30	3	0	52	0	0	52	7	0	57	2	0	67	3	0
	1	5	0	0	15	0	0	15	3	0	20	2	0	30	3	0	52	0	0	52	7	0	57	2	0	67	3	0
10	0	5	0	0	15	0	0	15	3	0	20	2	0	30	3	0	52	0	0	52	7	0	57	2	0	67	3	0
	1	5	0	0	15	0	0	15	3	0	20	2	0	30	3	0	52	0	0	52	7	0	57	2	0	67	3	0

Table 3.4 $G_1(x,y,q)$, $\Psi_1(x,y,q)$, $\delta_1(x,y,q)$ $\forall$ x,y,q

y \ x		2			4			5			6			7			8			9			10		
	q	F	Φ	γ	F	Φ	γ	F	Φ	γ	F	Φ	γ	F	Φ	γ	F	Φ	γ	F	Φ	γ	F	Φ	γ
2	0	5	0	0	10	2	0	10	2	0	15	4	0	15	4	0	20	6	0	20	6	0	25	8	0
	1	5	0	0	10	2	0	10	2	0	15	4	0	15	4	0	20	6	0	20	6	0	25	8	0
3	0	5	0	0	10	2	0	15	0	0	15	4	0	20	2	0	20	2	0	25	4	0	30	5	0
	1	5	0	0	10	2	0	15	0	0	15	4	0	20	2	0	20	2	0	25	4	0	30	5	0
4	0	10	0	0	20	2	0	20	2	0	30	4	0	30	4	0	40	6	0	40	6	0	50	8	0
	1	12	0	0	22	2	0	22	2	0	32	4	0	32	4	0	42	6	0	42	6	0	52	8	0
5	0	10	0	0	20	2	0	20	2	0	30	4	0	30	2	0	40	6	0	40	6	0	50	8	0
	1	12	0	0	22	2	0	22	2	0	32	4	0	32	2	1	42	6	0	42	4	1	52	8	0
6	0	15	0	0	30	2	0	30	2	0	45	4	0	45	2	0	60	6	0	60	4	0	75	8	0
	1	17	0	0	32	2	0	32	2	0	47	4	0	47	2	1	62	6	0	62	4	1	77	8	0
7	0	15	0	0	44	0	0	44	4	0	59	2	0	59	2	0	88	4	0	88	4	0	103	6	0
	1	17	0	0	44	0	0	44	4	0	61	2	1	61	2	1	88	4	0	88	4	0	105	6	1
8	0	20	0	0	44	0	0	44	4	0	64	2	0	64	2	0	88	4	0	88	4	0	108	6	0
		22	0	0	44	0	0	44	4	0	66	2	1	66	2	1	88	4	0	88	4	0	110	6	1
9	0	20	0	0	49	0	0	69	4	0	69	2	0	98	2	0	98	4	0	98	4	0	118	6	0
	1	22	0	0	49	0	0	71	4	0	71	2	1	98	2	1	98	4	0	98	4	0	120	6	1
10	0	25	0	0	50	2	0	59	0	0	75	4	0	84	2	0	100	6	0	109	4	0	125	8	0
	1	27	0	0	52	2	0	59	0	0	77	4	0	86	2	1	102	6	0	111	4	1	127	8	0

Table 3.5 $F_2(x,y,q)$, $\Phi_2(x,y,q)$, $\gamma_2(x,y,q)$ $\forall$ x,y,q

x \ y		2			3			4			5			6			7			8			9			10		
	q	G	Ψ	δ	G	Ψ	δ	G	Ψ	δ	G	Ψ	δ	G	Ψ	δ	G	Ψ	δ	G	Ψ	δ	G	Ψ	δ	G	Ψ	δ
2	0	5	0	0	5	2	0	10	2	0	10	2	0	15	4	0	15	4	0	20	6	0	20	6	0	25	8	0
	1	5	0	0	5	2	0	12	0	0	12	4	1	17	2	0	17	2	0	22	4	0	22	4	0	27	6	0
4	0	10	0	0	10	2	0	20	2	0	20	2	0	30	4	0	44	0	0	44	7	0	54	2	0	54	2	0
	1	10	0	0	10	2	0	20	2	0	20	2	0	30	4	0	44	0	0	44	7	0	54	2	0	54	2	0
5	0	10	0	0	15	0	0	20	2	0	25	2	0	30	3	0	44	0	0	44	7	0	54	2	0	59	3	0
	1	10	0	0	15	0	0	20	2	0	25	2	0	30	3	0	44	0	0	44	7	0	54	2	0	59	3	0
6	0	15	0	0	15	2	0	30	2	0	30	2	0	45	4	0	52	0	0	60	6	0	67	2	0	75	8	0
	1	15	0	0	15	2	0	30	2	0	30	2	0	45	4	0	56	0	0	60	6	0	71	2	0	75	8	0
7	0	15	0	0	20	0	0	30	2	0	35	2	0	45	4	0	52	0	0	60	6	0	67	2	0	75	8	0
	1	15	0	0	20	0	0	30	2	0	35	2	0	45	4	0	56	0	0	60	6	0	71	2	0	76	3	0
8	0	20	0	0	20	2	0	40	2	0	40	2	0	60	4	0	88	0	0	88	7	0	108	2	0	108	2	0
	1	20	0	0	20	2	0	40	2	0	40	2	0	60	4	0	88	0	0	88	7	0	108	2	0	108	2	0
9	0	20	0	0	25	0	0	40	2	0	45	2	0	60	4	0	88	0	0	88	7	0	108	2	0	113	3	0
	1	20	0	0	25	0	0	40	2	0	45	2	0	60	4	0	88	0	0	88	7	0	108	2	0	113	3	0
10	0	25	0	0	30	0	0	50	2	0	55	2	0	75	4	0	96	0	0	100	6	0	121	2	0	126	3	0
	1	25	0	0	30	0	0	50	2	0	55	2	0	75	4	0	100	0	0	100	6	0	125	2	0	130	3	0

Table 3.6 $G_2(x,y,q)$, $\Psi_2(x,y,q)$, $\delta_2(x,y,q)$ $\forall$ x,y,q

y \ x		2			4			5			6			7			8			9			10		
	q	F	Φ	Υ	F	Φ	Υ	F	Φ	Υ	F	Φ	Υ	F	Φ	Υ	F	Φ	Υ	F	Φ	Υ	F	Φ	Υ
2	0	5	0	0	10	2	0	10	2	0	15	2	0	15	2	0	20	2	0	20	2	0	25	2	0
	1	5	0	0	10	2	0	10	2	0	15	2	0	15	2	0	20	2	0	20	2	0	25	2	0
3	0	5	0	0	10	2	0	15	0	0	15	2	0	20	2	0	20	2	0	25	2	0	30	5	0
	1	5	0	0	10	2	0	15	0	0	15	2	0	20	2	0	20	2	0	25	2	0	30	5	0
4	0	10	0	0	20	2	0	20	2	0	30	2	0	30	2	0	40	2	0	40	2	0	50	2	0
	1	12	0	0	22	2	0	22	2	0	32	2	1	32	2	1	42	2	1	42	2	1	52	2	1
5	0	10	0	0	20	2	0	25	0	0	30	2	0	35	2	0	40	2	0	45	2	0	55	0	0
	1	12	0	0	22	2	0	25	0	0	32	2	1	37	2	1	42	2	1	47	2	1	55	0	0
6	0	15	0	0	30	2	0	30	2	0	45	2	0	45	2	0	60	2	0	60	2	0	75	2	0
	1	17	0	0	32	2	0	32	2	0	47	2	1	47	2	1	62	2	1	62	2	1	77	2	1
7	0	15	0	0	44	0	0	44	4	0	59	2	0	59	2	0	88	4	0	88	4	0	103	2	0
	1	17	0	0	44	0	0	44	4	0	61	2	1	61	2	1	88	4	0	88	4	0	105	2	1
8	0	20	0	0	44	0	0	44	4	0	64	2	0	64	2	0	88	4	0	88	4	0	108	2	0
	1	22	0	0	44	0	0	44	4	0	66	2	1	66	2	1	88	4	0	88	4	0	110	2	1
9	0	20	0	0	54	0	0	54	4	0	74	2	0	74	2	0	108	4	0	108	4	0	128	2	0
	1	22	0	0	54	0	0	54	4	0	76	2	1	76	2	1	108	4	0	108	4	0	130	2	1
10	0	25	0	0	54	0	0	59	0	0	79	2	0	84	2	0	108	4	0	113	4	0	133	2	0
	1	27	0	0	54	0	0	59	0	0	81	2	1	86	2	1	108	4	0	113	4	0	135	2	1

Table 3.7 $F_3(x,y,q)$, $\Phi_3(x,y,q)$, $\Upsilon_3(x,y,q)$ $\forall$ x,y,q

x \ y		2			3			4			5			6			7			8			9			10		
	q	G	Ψ	δ	G	Ψ	δ	G	Ψ	δ	G	Ψ	δ	G	Ψ	δ	G	Ψ	δ	G	Ψ	δ	G	Ψ	δ	G	Ψ	δ
2	0	5	0	0	5	2	0	10	2	0	10	2	0	15	2	0	15	2	0	20	2	0	20	2	0	25	2	0
	1	5	0	0	5	2	0	12	0	0	12	4	1	17	2	0	17	2	0	22	2	0	22	2	0	27	2	0
4	0	10	0	0	10	2	0	20	2	0	20	2	0	30	2	0	44	0	0	44	7	0	54	2	0	54	2	0
	1	10	0	0	10	2	0	22	0	0	22	4	1	32	2	0	44	0	0	44	7	0	54	2	0	54	2	0
5	0	10	0	0	15	0	0	20	2	0	25	2	0	30	2	0	44	0	0	44	7	0	54	2	0	59	3	0
	1	10	0	0	15	0	0	22	0	0	25	2	0	32	2	0	44	0	0	44	7	0	54	2	0	59	3	0
6	0	15	0	0	15	2	0	30	2	0	30	2	0	45	2	0	59	0	0	64	0	0	74	2	0	79	2	0
	1	15	0	0	15	2	0	32	0	0	32	4	1	47	2	0	61	0	0	66	0	0	76	2	0	81	2	0
7	0	15	0	0	20	0	0	30	2	0	35	2	0	45	2	0	59	0	0	64	0	0	74	2	0	84	0	0
	1	15	0	0	20	0	0	32	0	0	35	2	0	47	2	0	61	0	0	66	0	0	76	2	0	86	0	0
8	0	20	0	0	20	2	0	40	2	0	40	2	0	60	2	0	88	0	0	88	7	0	108	2	0	108	2	0
	1	20	0	0	20	2	0	42	0	0	42	4	1	62	2	0	88	0	0	88	7	0	108	2	0	108	2	0
9	0	20	0	0	25	0	0	40	2	0	45	2	0	60	2	0	88	0	0	88	7	0	108	2	0	113	4	0
	1	20	0	0	25	0	0	42	0	0	45	2	0	62	2	0	88	0	0	88	7	0	108	2	0	113	4	0
10	0	25	0	0	30	0	0	50	2	0	55	2	0	75	2	0	103	0	0	108	0	0	128	2	0	133	2	0
	1	25	0	0	30	0	0	52	0	0	55	2	0	77	2	0	105	0	0	110	0	0	130	2	0	135	2	0

Table 3.8 $G_3(x,y,q)$, $\Psi_3(x,y,q)$, $\delta_3(x,y,q)$ $\forall$ x,y,q

NR	XD	YD	q	CR	q'	AR
1	10	10	1	-2	0	0
2	10	2	0	8	0	0
3	8	2	0	6	0	0
4	6	2	0	4	0	0
5	4	2	0	2	0	0
6	2	2	0	0	0	1
7	2	2	0	0	0	1
8	2	2	0	0	0	1
9	2	2	0	0	0	1
10	2	2	0	0	0	1
11	10	8	1	6	1	0
12	6	8	1	2	1	0
13	2	8	1	-4	0	0
14	2	4	0	-2	0	0
15	2	2	0	0	0	1
16	2	2	0	0	0	1
17	2	4	1	0	0	5
18	4	8	0	-7	0	0
19	4	7	0	0	0	4
20	4	8	0	-7	0	0
21	4	7	0	0	0	4

Table 3.9 Representation of the cutting pattern corresponding to the value of 135 for Example 3.5.2

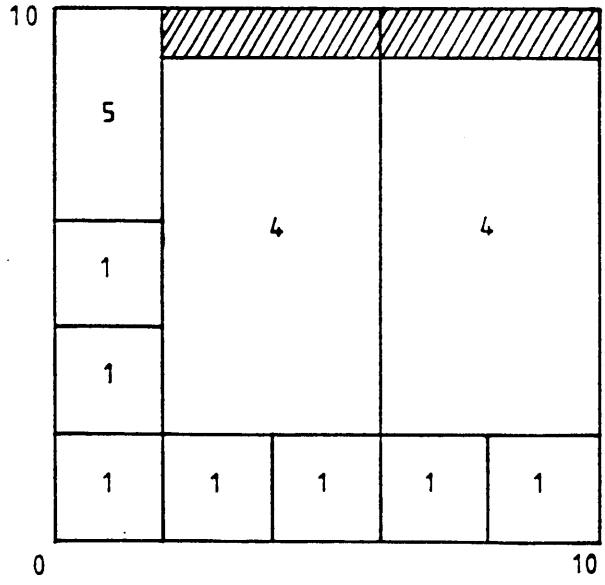


Figure 3.4 3-stage cutting pattern for Example 3.5.2

3.6 State Space Ascent (SSA)

In section 3.4, we described the form of the function g (.) which produces a valid state -space relaxation for the CGC problem, and in section 3.5 a bound on the solution of this problem which is derived from this relaxation. In this section we describe how a procedure can be used to decrease the resulting upper bound (Z_{UB}) obtained by SSR using state - space modifications.

The general objective in this approach is to force the solution of the relaxed recursion closer to feasibility, and naturally improve the upper bound. The form of the mapping function used and the respective relaxation (SSR) are described in Section 3. 4. 2:

$$(\text{SSR}) \qquad g(S_{xy}) = \sum_{i \in S_{xy}} q_i \xi_i$$

where q_i is a non - negative weight chosen to be associated with a piece of type i in set S_{xy} ($S_{xy} \subset \bar{R}$).

In choosing the function g (.) to be as given above, nothing has been said about the choice of the parameters q_i . Denoting by f (q) the upper bound produced by SSR (bound Z_{UB} of equations (17) - (18)) for a given vector q (= q_i), the new problem to be dealt with is

$$f(q^*) = \min_{q \geq 0} f(q) \qquad (21)$$

subject to

$$\sum_{i \in R} q_i Q_i \leq Q \qquad (22)$$

The procedure for minimising f (q) in the above expression is referred to as state - space ascent. This minimisation is, in general, difficult since f (q) is a discontinuous function of q. However, simple ways for performing the minimisation, are described and tested computationally in the following sections.

3.6.1 Modification of the weights q_i

Four different formulae are presented in this section, for modifying the weights q_i as part of the procedure used to solve problem (21) - (22) . The modification of the weights q_i is based on the following idea: Let c_i be the number of pieces of type i that have been used by a DP solution [equations (16) to (18)] in cutting A_0, for a given vector q. If c is feasible for the original CGC problem (i.e. $c_i \leq Q_i \quad \forall \quad i = 1, ..., m$), then the problem has been solved ; otherwise, it would be reasonable to try to reduce the number of pieces used for any type i for which $c_i > Q_i$ in order to move toward feasibility. A straightforward way to perform state space modifications is then to increase the value of the q_i corresponding to any type i for which $c_i > Q_i$ and at the same time, decrease the weight for the pieces for which the associated constraints are satisfied. One hopes, that by increasing the weight for a piece of type i such that $c_i > Q_i$, fewer pieces of type i will be used by the modified DP solution.

In the first three formulae adopted, weights q_i are modified accordingly by fixed amounts (independently of $| c_i - Q_i |$). However, the last formula provides a

general procedure for modifying the weights in a normal " subgradient " fashion.

Formula A

Set $q_i = q_i + 1 \quad \forall \quad i \quad$ s. t. $\quad c_i > Q_i, \quad i = 1, ..., m$

$q_i = q_i - 1 \quad \forall \quad i \quad$ s. t. $\quad q_i > 0, \quad c_i \leq Q_i - 5, \quad i = 1, ..., m$

Formula B

Set $q_i = q_i + 2 \quad \forall \quad i \quad$ s. t. $\quad c_i > Q_i, \quad i = 1, ..., m$

$q_i = q_i - 1 \quad \forall \quad i \quad$ s. t. $\quad q_i > 0, \quad c_i \leq Q_i, \quad i = 1, ..., m$

Formula C

Let i^* be that type i for which $(c_i - Q_i)$ is maximum. Trying to reduce the number of pieces used for type i^*, it is reasonable to increase the corresponding weight by the largest amount compared to the other weights, since this would produce the largest step toward feasibility. Thus, weights q_i are modified as follows :

Set $q_i = q_i + 3, \quad i = i^*$ s.t. $(c_{i*} - Q_{i*}) = \max \{ (c_i - Q_i) \mid c_i > Q_i, i = 1, ..., m \}$

$q_i = q_i + 1 \quad \forall \quad i \quad$ s.t. $\quad c_i > Q_i, \quad i \neq i^*, \quad i = 1, ..., m$

$q_i = q_i - 1 \quad \forall \quad i \quad$ s.t. $\quad q_i > 0, \quad c_i \leq Q_i, \quad i = 1, ..., m$

Formula D

A subgradient method is used to determine the values of the weights at iteration number j as follows:

$$\text{Set} \quad q_i^{j+1} = q_i^j + t^j \sqrt{(c_i^j - Q_i)} \quad \forall \; i \quad \text{s.t.} \quad c_i^j > Q_i, \quad i = 1, ..., m$$

$$= \max (0, q_i^j - t^j \sqrt{(Q_i - c_i^j)}) \quad \forall \; i \quad \text{s.t.} \quad c_i^j \leq Q_i, \quad i = 1, ..., m$$

where t^j is a positive scalar step size. A formula for t^j that has been proved effective in practice is given by:

$$t^j = \sqrt{\frac{\pi^j (Z_{UB}^{\ j} - Z_{LB})}{\sum_{i=1}^{m} (Q_i - c_i^j)^2}}$$

In this formula, the parameter π^j is initially set equal to 2.0 for a fixed number of consecutive iterations (6) being reduced to half of its value every 3 iterations until the resulting π^j falls below 0.06. Z_{LB} is the value of a feasible solution to the CGC problem obtained by the manual heuristic described in chapter 4 using interactive graphics. $Z_{UB}^{\ j}$ is the value of the upper bound obtained by SSR at the jth iteration. Note that if $Z_{UB}^{\ j} = Z_{LB}$, then the SSA procedure is terminated with Z_{LB} being the optimum solution.

3.6.2 SSA Procedure

In this section we present the complete procedure used in an attempt to minimize the upper bound Z_{UB} [equations (17) - (18)] obtained from the SSR of the CGC problem (section 3. 4) i. e. to solve problem (21) - (22). The SSA procedure is then as follows :

(1) Choose initial values for the weights q_i. No good indication exists on how to determine good starting values - we used $q_i = 0 \quad \forall \quad i = 1, ..., m$.

(2) Solve the relaxed DP recursion [equations (16) - (18)] to obtain the

optimal value Z_{UB} relative to the current set of q_i' s , by performing the DP procedure of section 3.5.1.

(3) Check if the DP solution c (= c_i) is a feasible solution to the original problem, i. e. if $c_i \leq Q_i \quad \forall \; i = 1, ..., m$. If yes, stop ; otherwise continue. If $Z_{UB} < Z_{min}$ (the minimum upper bound obtained so far) then update Z_{min} with $Z_{min} = Z_{UB}$.

(4) Modify the set of q_i' s using one of the four Formulae of section 3.6.1.

(5) Go to (2) to resolve the relaxed dynamic program with this new set of weights unless a sufficient number of SSA iterations has been performed.

At the end of the SSA procedure, the optimal solution to the original CGC problem may have been found (Step 3), but if not the best bound (Z_{min}) on the problem has been obtained which can then be used in a tree search procedure to solve the problem (Note that $f(q^*) = Z_{min}$ in problem (21) - (22)).

3.6.3 Computational Results

In implementing the DP recursion given for the relaxed CGC problem [equations (16) - (18)] on a computer, the strategies for generating the states as well as for sequencing of recursive computations will significantly affect the memory and processing time. The approach proposed for the implementation is based on a trade off between time and space requirements.

Once $F(\alpha_0, \beta_0, Q)$ and $G(\alpha_0, \beta_0, Q)$ are found at the end of a SSA

iteration, backtracking to obtain the generated cutting pattern for A_0 can be done easily when the full sets of maximum value functions and their corresponding optimal decisions are stored in RAM. However, our empirical results suggest that only small - size CGC problems can be solved using this computer implementation, since a DP solution memory space is almost always the dominating limiting factor. One possible alternative to circumvent the difficulty, is to store the value and the optimal decision for each set (x, y, S_{xy}) on a peripheral device when $F (x, y, S_{xy})$ and $G (x, y, S_{xy})$ are computed and retrieve them at the time of backtracking. In this case, we consider the increase in processing time for backtracking be well worth its cost. Note that reading from and writing to a secondary storage device can be computationally very costly.

The computer code for the SSA procedure described in Section 3.6.2 has been written in FORTRAN and tested computationally on a variety of problems run on a CYBER - 855 machine. In the DP implementation, the six memory grids $F_k (x, y, q)$, $\Phi_k (x, y, q)$, $\gamma_k (x, y, q)$, $G_k (x, y, q)$, $\Psi_k (x, y, q)$ and $\delta_k (x, y, q)$, defined in section 3.5.1, are stored on a secondary storage device for all $k = 0, 1, ..., n$ for a particular SSA iteration.

Tables 3.10 and 3.11 present the computational comparison of the four Formulae given in section 3.6.1 for the modification of the weights q_i. The SSA procedure was applied on a set of 9 test problems randomly generated, including up to 20 types of pieces in $\bar{R}$ required to be cut from a large rectangle of size (40,70).

The first five columns in Tables 3.10 and 3.11 give details about the test problems. These details include for each problem: the size of A_0, the number of types of pieces in $\bar{R}$, the sizes of the normal sets and value of the optimal solution,

Problem	Details of Test Problems					Results of Rule 1				Results of Rule 2				Results of Rule 3			
Number	(α_0, β_0)	m	|L|	|W|	Optimal solution (Z_{opt})	Best Upper Bound (UB1)	Number of Itera-tions	Q(UB1)	Time to obtain UB1	Best Upper Bound (UB2)	Number of Itera-tions	Q(UB2)	Time to obtain UB2	Best Upper Bound (UB3)	Number of Itera-tions	Q(UB3)	Time to obtain UB3
1	(10,10)	5	8	9	135	135*	2	2	0.5	135*	2	4	0.7	135*	2	4	0.8
2	(15,10)	7	12	10	244	244*	4	5	3.4	244*	3	6	3.2	244*	2	5	1.6
3	(20,20)	7	17	17	500	517	20	48	680	517	20	14	280.4	517	20	36	862.7
4	(20,30)	10	11	23	1755	1755*	4	3	4.8	1755*	3	3	4.3	1755*	3	5	6.3
5	(30,30)	7	23	15	1074	1117[b]	17	47	1177	1117	20	30	1114.7	1117[b]	18	47	1305.6
6	(30,40)	8	26	17	1351	1421[a]	19	41	1500	1444[a]	17	36	1500	1435[a]	16	36	1500
7	(30,50)	10	26	17	1653	1716	20	27	1037.5	1790	20	16	613.8	1761	20	17	724.1
8	(40,70)	10	29	56	2892	2902[a]	7	19	1500	2902[a]	5	20	1500	2893[a]	5	19	1500
9	(40,70)	20	25	55	1860	1860*	9	20	1500	1940[b]	5	23	787.2	1940[b]	4	17	404.9

Table 3.10 The State Space Ascent (SSA) Procedure Using Three Different Formulae for the modification of the weights q_i

a Time Limit

b Q attained its maximum value allowed for the computation of Bound

* Optimum solution found by the SSA Procedure

Problem Number	Details of Test Problems					Results of Rule 4					
	(α_0,β_0)	m	|L|	|W|	Optimal Solution (Z_{opt})	Best Upper Bound (UB4)	Lower Bound (Z_{LB})	Duality gap (r) %	No. of SSA iters	Q(UB4)	Time in CYBER-855 seconds
1	(10,10)	5	8	9	135	135*	135	-	2	2	0.2
2	(15,10)	7	12	10	244	244*	244	-	5	3	3.6
3	(20,20)	7	17	17	500	517	467	3.4%	20	11	152.3
4	(20,30)	10	11	23	1755	1755*	1755	-	8	13	63.4
5	(30,30)	7	23	15	1074	1117	1020	4.0%	20	21	555.8
6	(30,40)	8	26	17	1351	1445[a]	1351	6.9%	15	36	1500.0
7	(30,50)	10	26	17	1653	1720[a]	1643	4.0%	17	31	1500.0
8	(40,70)	10	29	56	2892	2893[a]	2698	0.0%	6	35	1500.0
9	(40,70)	20	25	55	1860	1860*	1860	-	3	15	214.2

Table 3.11 The State Space Ascent (SSA) Procedure Using a Subgradient Method for the modification of the weights q_i.

a Time Limit

* Optimum solution found by the SSA Procedure

this being found by the exact algorithm described in the following section of this chapter, which solves the CGC problem. To form an idea of the computational performance of each formula for modifying the weights, we also give for each problem, the best upper bound (Z_{min}) obtained from the SSA procedure (let UB1, UB2, UB3 and UB4 represent the value of Z_{min} when Formulae A, B, C and D are used respectively), the number of iterations required and the time taken to reach this value (the time is given in CYBER - 855 seconds) and the maximum value of Q attained during the SSA procedure

$$(Q = \sum_{i=1}^{m} Q_i q_i).$$

(For the " subgradient " type formula, the value of the feasible solution - Z_{LB} - used is also given for each problem).

A maximum number of 20 iterations and a time limit of 25 minutes (CYBER 855 - FTN5 compiler) were imposed. A maximum value of Q was also set for each test problem, depending on its size, as a result of limiting memory requirements. Whenever the SSA procedure succeded in finding the optimal solution, a star (*) is added to the value of Z_{min}. A label (a) or (b) added to the value of Z_{min} mean that the respective SSA procedure is terminated by a preset time limit or the maximum value of Q allowed for the computation of the bound is attained, respectively.

From Table 3.10 it is clear that the first three formulae perform very well for the 9 problems. The quality of the bound is the same for the first five problems. For the four largest problems, the best out of the three values is always within 5 % of the optimal solution. In particular, the SSA procedure found the optimal solution for problems 1, 2 and 4 using either of the three formulae. For problems 6, 7 and 9, UB1 performed considerably better than UB2 and UB3, namely the

value of UB1 is nearer to the optimum solution by approximately 4 % and 3 % on average, than the corresponding values of UB2 and UB3, respectively, for these problems.

To test these formulae even further, we need to compare the rate of convergence and number of SSA iterations performed. Figures 3.5 - 3.13 plot the various bounds obtained by the SSA procedure for all test problems using the four formulae, one at a time, as a function of the number of iterations in the ascent procedure. These plots clearly show that UB1 and UB4 converge much earlier than UB2 or UB3, particularly for problems 3, 5, 6, 7 and 9. As a result of better convergence, Formula A and D are more likely to require a lower computational cost. Furthermore, the figures suggest that for most of the problems, the largest step towards optimality is achieved during the first few iterations (4 to 5) indicating quick convergence of the SSA procedure.

The above results suggest that a " subgradient " type formula would work better than the first three. Indeed, from Table 3.11 it is clear that UB4 performed better than UB1. The quality of these two bounds is almost the same for all problems. However, the number of iterations in the ascent procedure is smaller for the last four problems when Formula D is used, resulting in quicker convergence of UB4. Furthermore, the subgradient method requires lower computational cost, particularly for problems 3, 5 and 9. The maximum value of Q required for the computation of UB4 is lower for 5 out of the 9 problems tested, meaning that the SSA procedure is more likely to solve larger CGC problems when Formula D is used (as a result of limiting memory requirements). Note that the value of Q obtained at consecutive SSA iterations, when UB1 is used, can be expressed as a monotonically increasing function of the number of iterations.

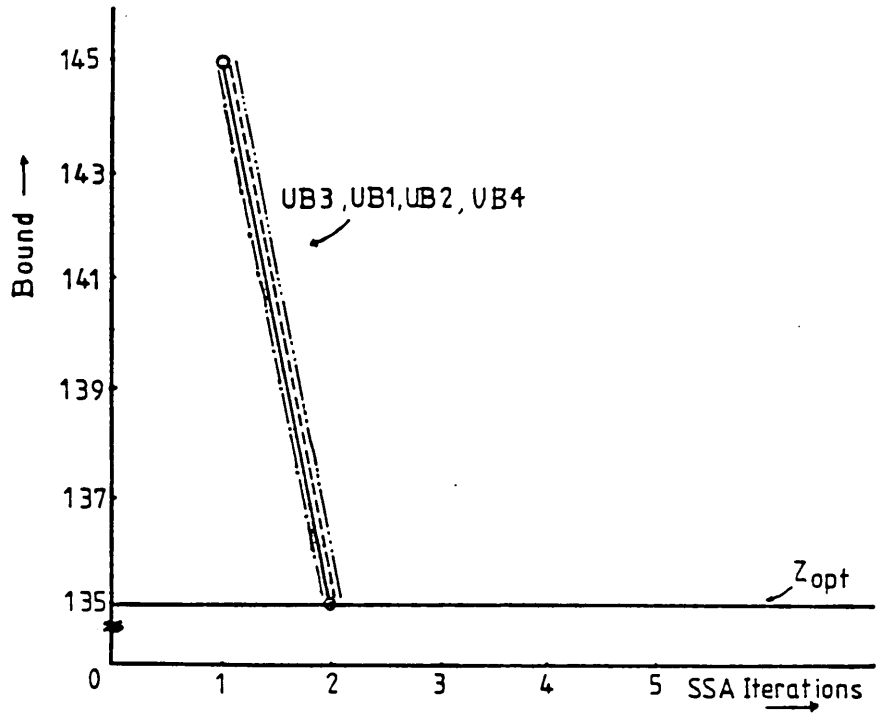


Figure 3.5 State-Space Ascent for Problem 1

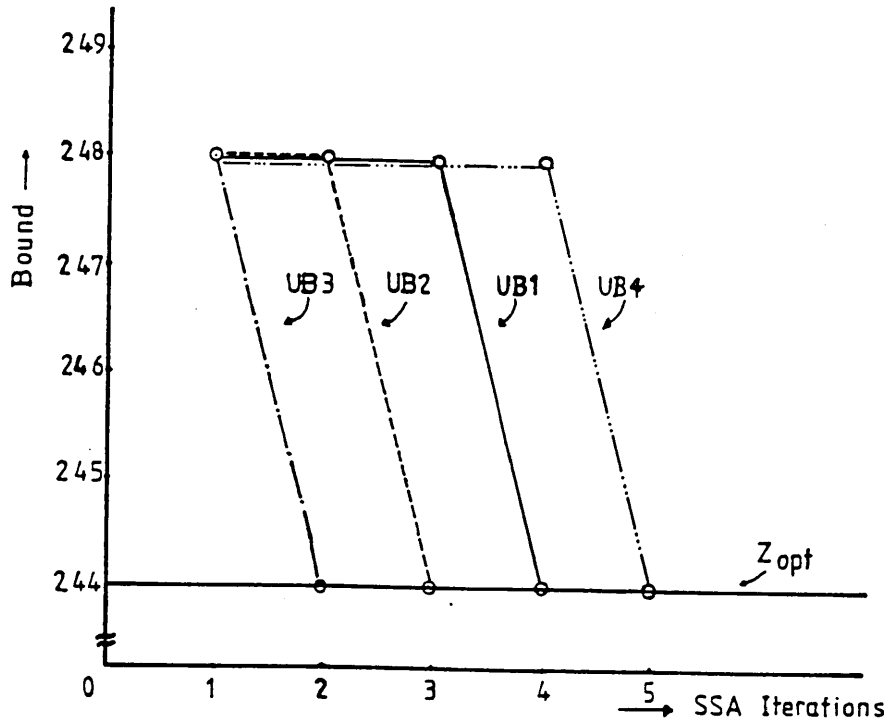


Figure 3.6 State-Space Ascent for Problem 2

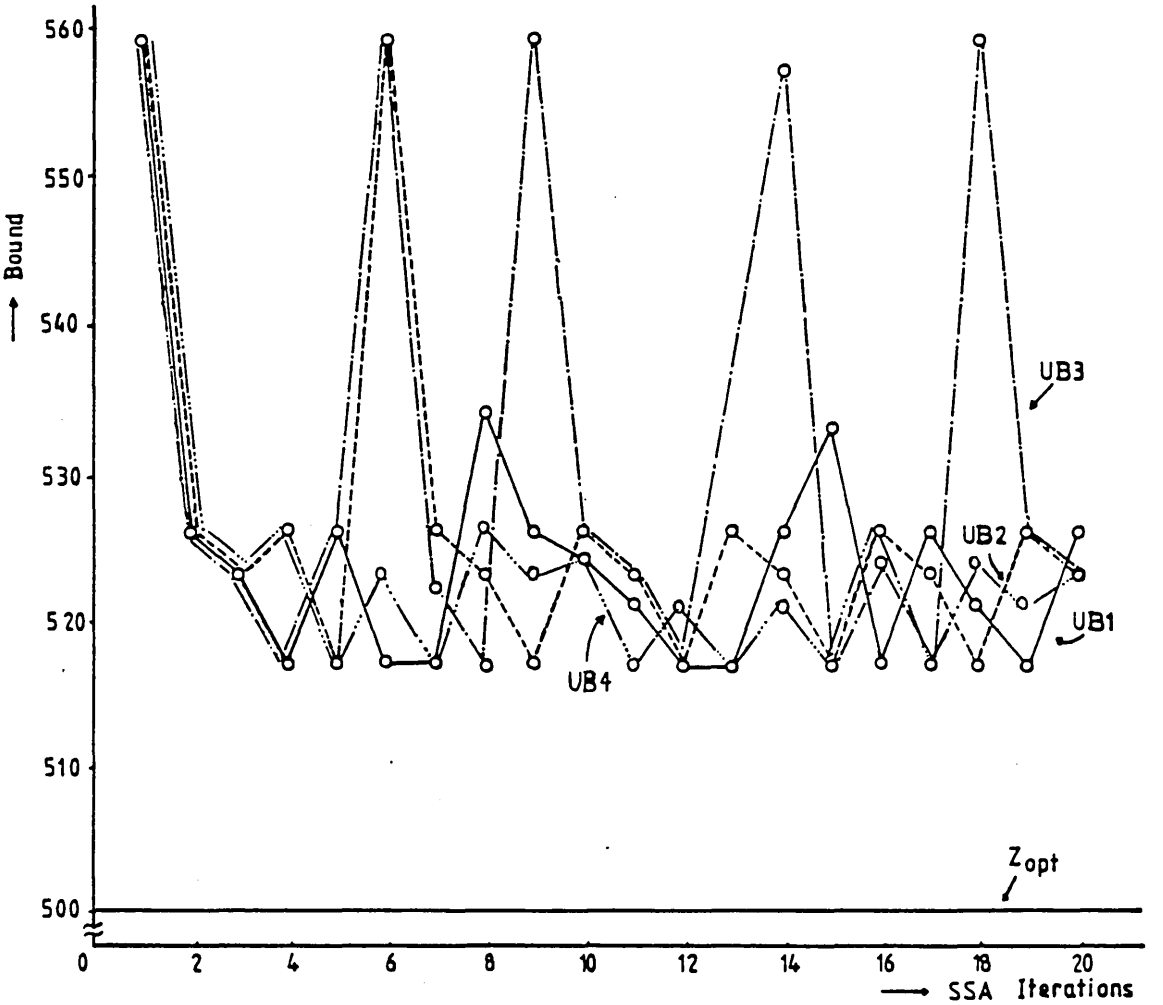


Figure 3.7 State-Space Ascent for Problem 3

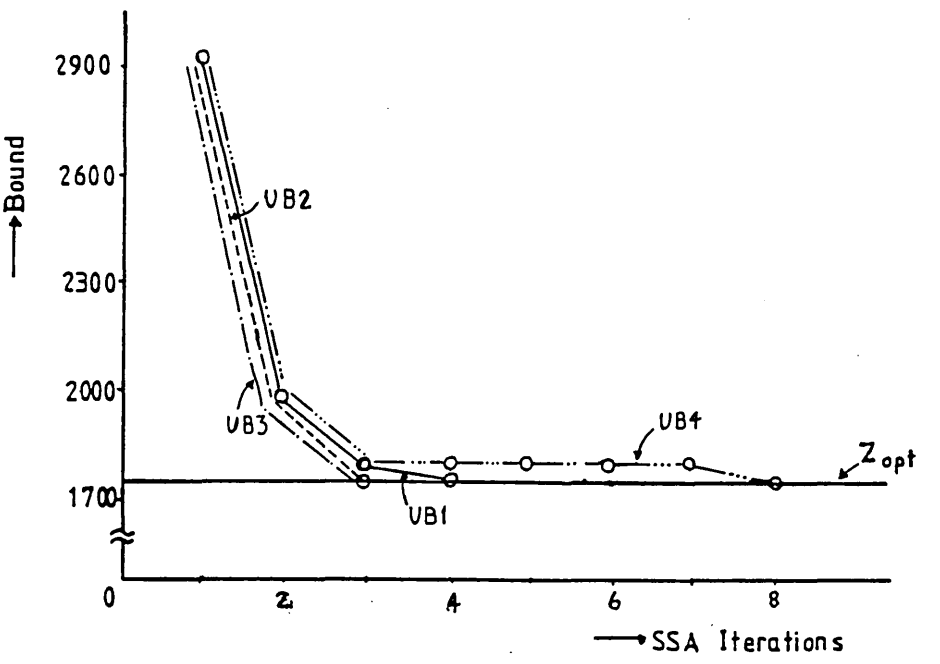


Figure 3.8 State-Space Ascent for Problem 4

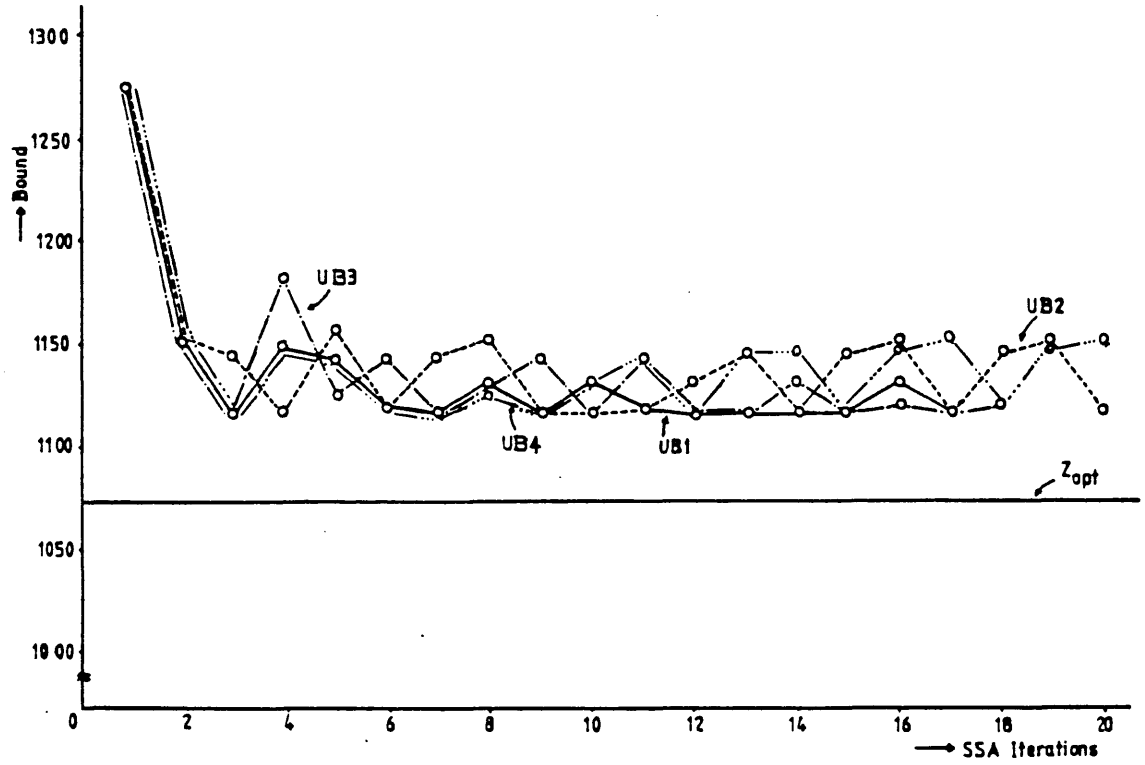


Figure 3.9 State-Space Ascent for Problem 5

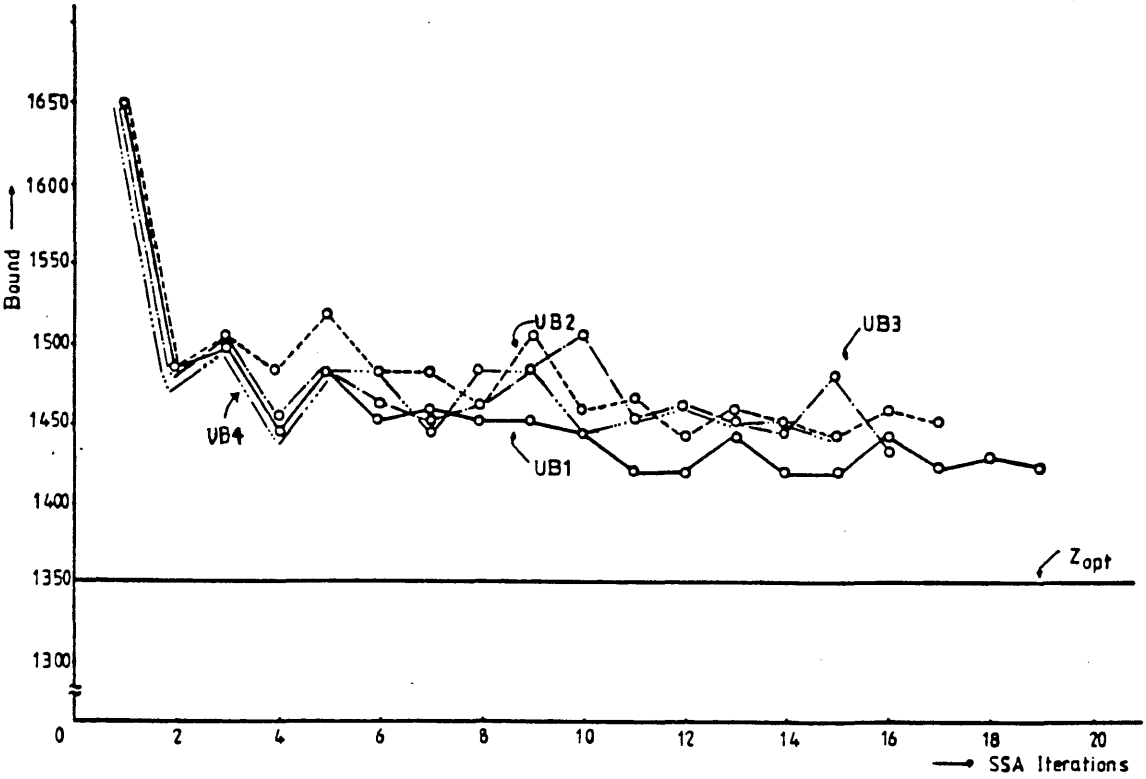


Figure 3.10 State-Space Ascent for Problem 6

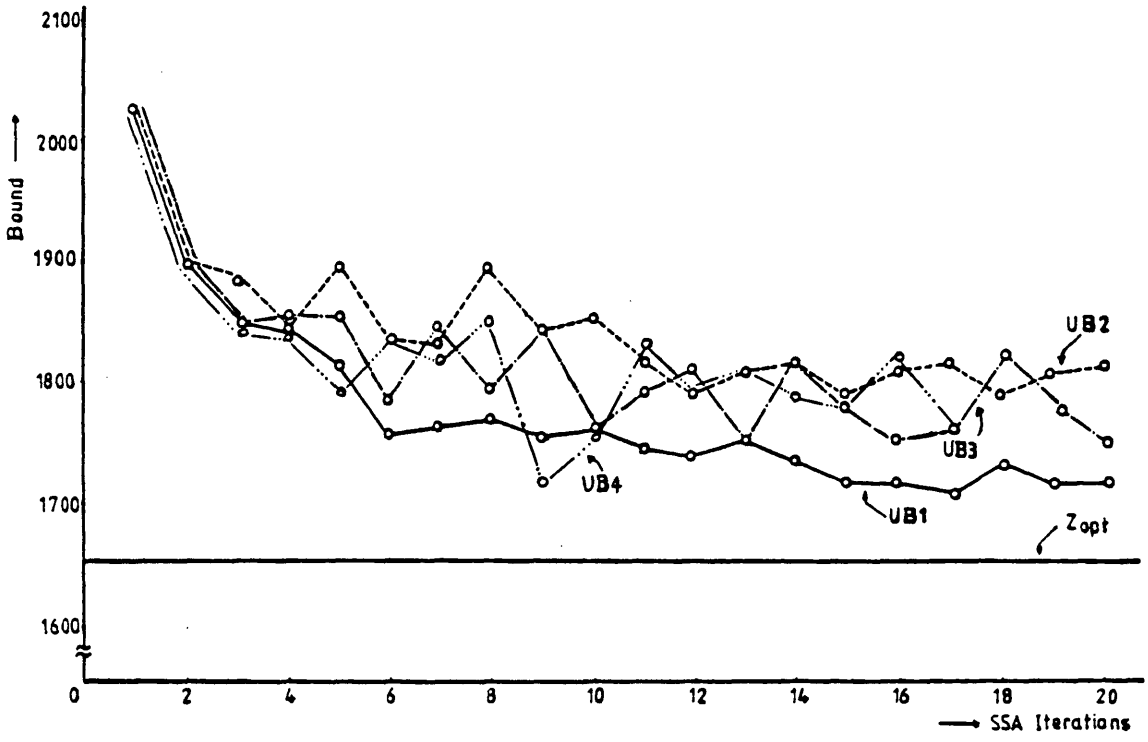


Figure 3.11 State-Space Ascent for Problem 7

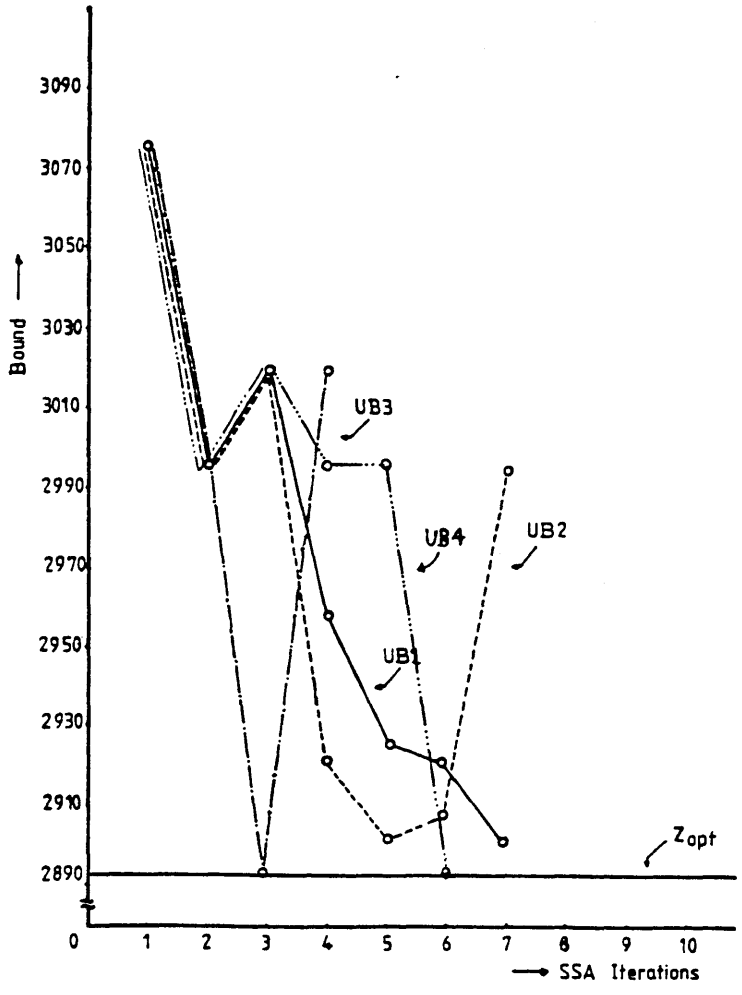


Figure 3.12 State-Space Ascent for Problem 8

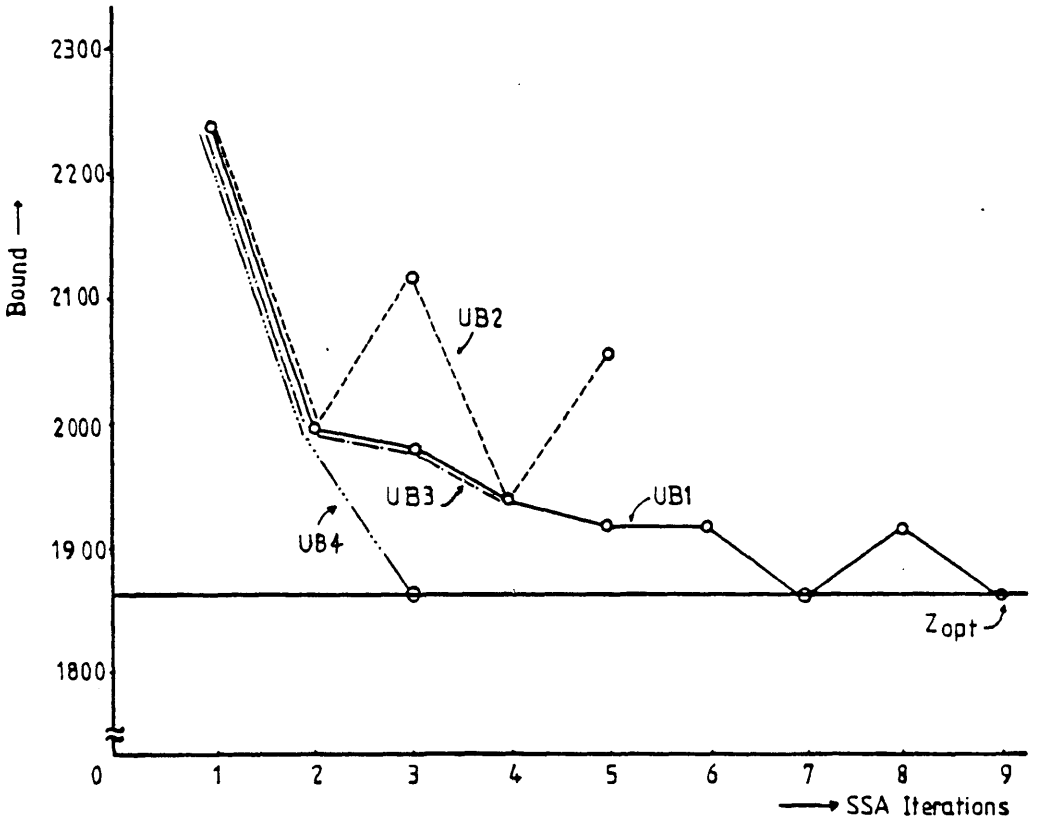


Figure 3.13 State-Space Ascent for Problem 9

The computational experience derived from the results of the SSA procedure applied to the problems presented above, shows that this method finds high quality bounds for medium-sized CGC problems, provided an efficient way of modifying the weights assigned to the pieces in $\bar{R}$ can be found. Formulae A or D seemed to be a good choice for satisfying this requirement.

For problems, in which the optimal solution has not been found by the procedure described so far in this chapter, the bounds obtained can be embedded in a tree - search algorithm used to solve these problems exactly. Such an algorithm is developed in the following sections.

3.7 An Enumerative Agorithm for the CGC Problem

An enumerative algorithm is a method guaranteed to find an optimal solution to a problem by reducing it to a finite number of solvable problems within a finite computation enabling us to use a branch - and - bound approach. In any branch - and - bound procedure the calculation of bounds on the value of the solution to a remaining problem (at some node of the tree) is of the utmost importance to the efficiency of the algorithm.

In the following sections, we apply such a solution procedure to the CGC problem as it has been defined in section 3.2. First we describe a tree - search procedure that generates all possible cutting patterns of the rectangles on A_0 without duplication. This enumerative algorithm is based on the procedure used by *Christofides and Whitlock* [1977] for solving the constrained two - dimensional guillotine cutting problem. We then show how a bound can be incorporated into the

above tree so as to limit the amount of search necessary in order to solve the CGC problem. A bound derived from the SSA procedure in the way described in section 3.6.2 is used during the search.

3.7.1 Enumerative Procedure

The process of cutting rectangles of various sizes from A_0 and allocating to them pieces in set $\bar{R}$, can be recorded in terms of a tree, which is described below. Branchings in this tree represent cuts on a rectangle. Thus, the branches emanating from the root - node of the tree correspond to all possible cuts on A_0, and each node at the end of a branch represents the rectangles produced by the corresponding cut on A_0. Each node n represents a state of rectangle A_0 after cutting has taken place. This is described by the list F of rectangles produced by the sequence of cuts corresponding to the path that leads from the root of the tree to node n . In List F each rectangle is identified by a four - part label (x, y, r, s) where (x, y) are the dimensions of the rectangle and r and s are integers representing the lengths and widths, respectively, at which cuts can be made on rectangle (x, y). A rectangle is then selected from list F , used to represent a node n, and branching occurs from this node by making all possible cuts on the chosen rectangle.

Let a rectangle e with label (x, y, r, s) be chosen for cutting by the enumerative procedure at a particular node n. Then the sets of " all possible cuts " that can be performed on rectangle e parallel to the y - and x - axis are given by L' = { 0 , 1, 2, ..., x - 1 } and W' = { 0, 1, 2, ..., y - 1 } respectively, i. e. $r \in L'$ and $s \in W'$. To generate all possible patterns, we must include an artificial cut, referred to as the " 0 - cut ", that leaves the rectangle intact. (Note that a rectangle that has been " cut " by a 0 - cut must not be a candidate for future cutting and must

- from that node onward - be considered fixed). If all X-cuts at any position $r \varepsilon L'$ and Y - cuts at any position $s \varepsilon W'$ are made, producing $(x + y - 1)$ branches, then the sets of rectangles represented at successive nodes will be duplicated at several nodes because of the appearance of symmetrical cutting patterns. These duplications can easily be removed in a way that is best shown by the simple example illustrated in Fig. 3.14. Let the chosen rectangle (x, y) be cut into smaller rectangles, A and B by an X - cut at $r = a (\varepsilon L')$. When the cut is made at $r = x - a ((x - a) \varepsilon L')$, which is symmetrically opposite to the cut at $r = a$ with respect to (x, y), the second pattern shown is produced. (Although this example appears simple it becomes far more complex when rectangles A and B are also cut further). In this case, duplication can be avoided without missing any unique cutting pattern by limiting the set of X - cuts to $L' = \{ 0 , 1, 2, ..., [x / 2] \}$ where [x / 2] means " the greatest integer not greater than ". Similarly, set W' of the Y - cuts can be redefined to be $W' = \{ 0, 1, 2, ..., [y / 2] \}$.

Central to the enumerative procedure is the concept of cut ordering. This is best shown by the example illustrated in Fig. 3.15. Let the chosen rectangle (x, y) be cut into two smaller rectangles (a, y) and (x - a, y) by an X - cut at $r = a$ $(\varepsilon L')$. A second X - cut performed on (x - a, y) at $r = b$ $(\varepsilon L')$ such that $a < b \leq [(x - a) / 2]$ at some successor node results in producing three rectangles A, B and C. The same set of rectangles is generated by the second pattern shown where the numbers next to the X - cuts indicate the order in which the cuts are made. This type of duplication can obviously be removed without missing any unique cutting patterns by introducing an arbitrary cut ordering so that if a rectangle (x, y) is cut at, say , $r = \alpha$, then all subsequent X - cuts on the two resultant rectangles must be greater than or equal to α.

The restrictions imposed by both the symmetry and cut ordering effects on

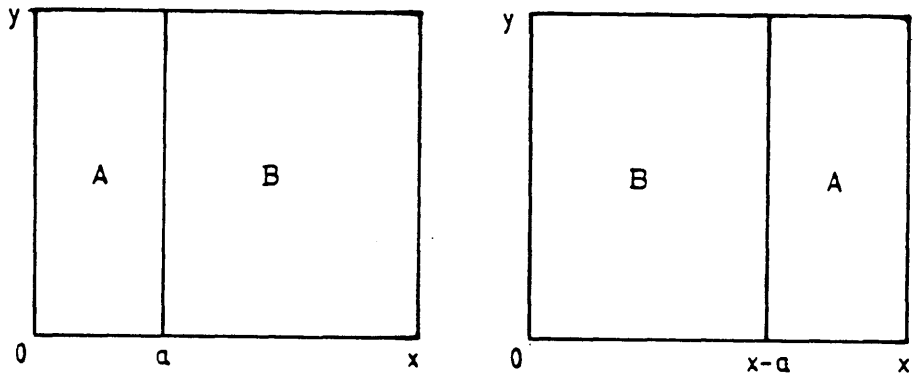


Figure 3.14 Effect of Symmetry

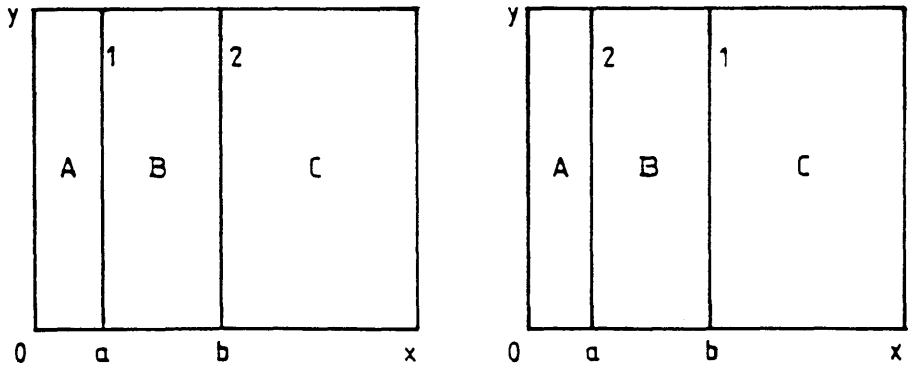


Figure 3.15 Effect of Cut Ordering

the cutting patterns produced by cutting a rectangle (x, y) into two smaller rectangles (α , y) and (x - α, y), imply that for the larger of the two resultant rectangles, (x - α, y), the range of X - cuts is now limited to $\alpha \leq r \leq [(x - \alpha)/2]$ and, in particular, if $[(x - \alpha) / 2] < \alpha$ no further X - cut on that rectangle need be made. For the smaller of the two resultant rectangles, (α, y), the restriction imposed by the cut ordering implies that no further X - cut is possible. A similar kind of restriction can be imposed on the Y - cuts.

The consequence of symmetry and cut ordering, as explained above, is to eliminate from explicit consideration different sequences of cuts when these lead to the same final cutting pattern. The search involved in the enumerative procedure is limited further by the use of the idea of Normal Patterns as these are explained in section 2.3.1 of chapter 2. Thus, without any loss of generality, we redefine the sets L' and W' described above to be given by $\tilde{L} = \{ r \mid r \in L', r \in L \}$ and $\tilde{W} = \{ s \mid s \in W', s \in W \}$ respectively, where L and W represent the sets of Normal cuts as these are defined in section 3.5 of this chapter.

3.7.2 Description of Enumerative Algorithm

In this section we present the implementation of the enumerative procedure as described in section 3.7.1. The data structure employed, is based on the representation of a rectangle (x, y) produced during the search at node n by a four - part label (x, y, r, s). The meaning of r and s, where, $r \in \tilde{L}$ and $s \in \tilde{W}$, is as follows :

If $1 \leq r \leq [x / 2]$, then the next cut to be considered on rectangle (x, y) - if this rectangle is chosen for cutting - is an X - cut at position r. If $r = [x / 2] + 1$ and $1 \leq s \leq [y / 2]$ then the next cut to be made on (x, y) is a Y - cut at position

s. If $r = [x/2] + 1$ and $s = [y/2] + 1$, all feasible X - cuts and Y - cuts on rectangle (x, y) have been performed and the next cut to be made is a 0 - cut. If $r = 0$, we infer that a 0 - cut on (x, y) has been made and this rectangle is not to be cut further by any branching following node n. Note that only one rectangle is cut at a node.

The state of the search procedure in the tree is described by the List F of four part labels corresponding to rectangles produced by the cuts so far. This list is updated for forward and backward branchings. Let a rectangle (x_j, y_j) from list F be chosen for cutting at node j, then two more values x'_j and y'_j are added to the vector associated with node j, where x'_j and y'_j represent future X - or Y - cuts on (x_j, y_j) being obtained as explained earlier.

The algorithm, based on an exhaustive search as a result of the introduction of the 0-cut, is then described as follows :

(1) Set LEVEL = 1 and $F = \{(\alpha_0, \beta_0)\}$.

(2) If there is a rectangle in F which has not been cut by a 0 - cut, then call it E and remove it from F ; otherwise if LEVEL = 1 stop, else go to 4.

(3) Forward branching : Set LEVEL = LEVEL + 1, perform next cut on E and add the resulting rectangles into F. Go to 2.

(4) Backtracking : Set LEVEL = LEVEL - 1, remove the rectangles produced by the last cut on E from List F . If the last cut made was a 0 - cut then go to (5) ; otherwise go to (3).

(5) Add rectangle E back into F since all cuts on E have been completed, and call the rectangle cut at level (LEVEL - 1) of the tree the new E. Go to (4).

3.8 A Tree - Search Algorithm for the CGC Problem

Section 3.7 gave an enumerative algorithm for the CGC problem that could generate all normal cutting patterns for rectangle A_0 with respect to a given set $\bar{R} = \{ (\alpha_i, \beta_i), i = 1, ..., m \}$ requiring at most Q_i pieces of type i (i = 1, ..., m) to be cut from A_0. The algorithm generates all cutting patterns without symmetric duplications and without explicitly considering different sequences of cuts when these lead to the same cutting pattern. However, in this algorithm no special consideration has yet been given to the fact that the CGC problem has been formulated as a dynamic program described by recursion (16) - (18) of section 3.5. The purpose of this section is to develop a tree search algorithm, that limits the enumerative search necessary to determine an optimum solution to the CGC problem, by using the bound derived from the SSA procedure described in section 3.6.2.

A description of the proposed algorithm is presented below :

(a) Perform the SSA procedure at the initial tree node obtaining an upper bound Z_{min} on the solution of the problem to be considered.

(b) Choose a node of the tree and pick an uncut piece associated with this node to branch on - i.e.we will investigate making all possible cuts on the chosen rectangle. Note that if no tree node can be found the search is terminated.

(c) Calculate an upper bound Z^* for the optimal completion of each new node in the tree.

(d) If a new feasible solution is found at step (c) with value Z^* greater than the value $\bar{Z}$ of the highest feasible solution currently available, then an improved solution has been obtained; Z* can then replace $\bar{Z}$ and

backtracking can occur.

(e) Discard any nodes in the tree that have an associated upper bound Z^* less than $\bar{Z}$ and go to (b).

3.8.1 The computation of Bound at the initial node

The SSA procedure described in section 3.6.2 is carried out at the initial node of the tree using Formula A for the modification of the weights q_i. Let N be the number of SSA iterations performed. Also, let Z_{UB}^j and $P^j = \{ q_i^j \mid i = 1, ..., m \}$ represent the value of the upper bound obtained by SSR at the j^{th} SSA iteration and the associated set of weight q_i's, respectively. Then

$$Q^j = \sum_{i=1}^{m} q_i^j Q_i$$

denotes the largest value that state q can take in the relaxed DP recursion [equation (16) - (18)] at the j^{th} iteration ($j = 1, ..., N$).

If no optimal solution to the CGC problem is found at the completion of the N iterations, the procedure determines the best upper bound on the solution by:

$$Z_{min} = \min \{ Z_{UB}^j \mid j = 1, ..., N. \}$$

Note that in case of more than one upper bound being equal to the minimum value, we choose the one associated with the smaller value of Q^j. The set P* of weights that gave Z_{min} is then recalled to be used in the calculation of the minimum upper bound

$$(Q^* = \sum_{\substack{i=1 \\ q_i \, \varepsilon \, P^*}}^{m} q_i \, Q_i).$$

Once $F_n (\alpha_0, \beta_0, Q^*)$ and $G_n (\alpha_0, \beta_0, Q^*)$ are computed for this iteration, a single dynamic programming table $V_c (x,y)$ is constructed in the following way :

$$V_c (x, y) = F_n (x, y, Q^*) \text{ if } Z_{min} = F_n (\alpha_0, \beta_0, Q^*)$$
$$= G_n (x, y, Q^*) \text{ if } Z_{min} = G_n (\alpha_0, \beta_0, Q^*)$$

$V_c (x, y)$ represents an upper bound on the constrained solution to any rectangle (x, y), $1 \leq x \leq \alpha_0$ and $1 \leq y \leq \beta_0$. The four memory grids $\Phi_k (x, y, q)$, $\gamma_k (x, y, q)$, $\Psi_k (x, y, q)$ and $\delta_k (x, y, q)$ associated with Z_{min}, stored on a secondary device for all $x \varepsilon L$, $y \varepsilon W$, $q = 0, ..., Q^*$ and $k = 0, 1, ..., n$ are retrieved by the DP procedure of section 3.5.1 to discover the nature of the cutting pattern associated with any $V_c (x, y)$ value. Thus, a matrix Rect (x, y, i) is generated for all $x \varepsilon L$, $y \varepsilon W$ and $i = 1, ..., m$ to represent the number of pieces c_i of type i in $\bar{R}$ required to cut any rectangle (x, y). All the entries for this matrix are obtained from the solution calculated for the initial rectangle A_0. Note that tables V_c and Rect are written to RAM so that they are available for later use.

3.8.2 The computation of the Bound at the Tree nodes

The state of the search procedure at a tree node n (as mentioned earlier) is described by the List F of rectangles produced by the sequence of cuts corresponding to the path that leads from the root of the tree to node n. Let those

rectangles that have had a 0 - cut made on them form the subset $H_0 \subseteq F$. The rectangles in H_0 will not be cut at any node below the current node n and in the final pattern will have some piece from the set $\bar{R}$ fitted in them - there is exactly one piece from set $\bar{R}$ fitted into each rectangle in H_0 - because the " waste " is not cut away by the algorithm itself. The allocation of pieces in $\bar{R}$ to rectangles in H_0 can be done in an optimal fashion as follows :

Form a matrix $[\, a_{ik} \,]$ with m rows corresponding to the pieces in $\bar{R}$ and u columns corresponding to the u (say) rectangles (x_k, y_k) in H_0. Set $a_{ik} = v_i$ if $\alpha_i \leq x_k$ and $\beta_i \leq y_k$ and $a_{ik} = -\infty$ otherwise.

The best feasible allocation of pieces to rectangles is then given by a solution to the transportation problem:

$$\text{Max } V_T = \sum_{k=1}^{u} \sum_{i=1}^{m} a_{ik} z_{ik}$$

subject to

$$\sum_{i=1}^{m} z_{ik} \leq 1$$

$$\sum_{k=1}^{u} z_{ik} \leq Q_i$$

$$z_{ik} \geq 0$$

The solution to this problem is made very simple by the special structure of the $[a_{ik}]$ matrix, and the problem can be solved by an efficient transportation routine (*Christofides and Whitlock* [1977]). V_T then represents the value of the transportation solution for all rectangles in H_0 and δ_i the number of pieces of type i in $\bar{R}$ $(=\sum_{k=1}^{u} z_{ik})$ used by the above solution. Note that at a terminal node (i. e. a node at which all rectangles in the List F have had 0 - cuts made on them), V_T is the value of the cutting pattern corresponding to that node.

The rectangles in F that have not had 0 - cuts made on them are liable, at future branchings, to be cut further into smaller rectangles and, hence, may be allocated several pieces from $\bar{R}$ in the final solution. It is, therefore, not possible to use the transportation routine to calculate an upper bound for these rectangles, as this only allocates one piece per rectangle. Thus, $V_c (x, y)$, derived from the solution for the initial rectangle A_0 and calculated only once at the initial tree node, as described in section 3.8.1., serves as an upper bound on the value obtainable from each rectangle (x, y) in $F - H_0$.

The procedure for calculating an upper bound at a node n is, therefore, to solve the transportation problem for all rectangles in H_0 and to use table $V_c (x, y)$ to obtain directly the constrained solutions for all other rectangles. The value of the upper bound Z^* is then given by :

$$V_T + \sum_{(x,y) \, \varepsilon \, F-H_0} V_c (x, y).$$

The solution associated with Z^* is feasible to the original CGC problem, if the number of pieces of any type i (i = 1, ..., m) used by the current solution does not exceed the maximum number available in $\bar{R}$, i.e

$$\delta_i + \sum_{(x,y)\ \varepsilon\ F\text{-}H_0} Rect(x, y, i) \leq Q_i \qquad \forall\ i = 1, ..., m.$$

Note that the major part of the computation of the upper bound at a tree node is the solution of the transportation problem and this solution need take place only at nodes resulting from some 0 - cut, i.e. only when the set H_0 of rectangles changes.

3.8.3 Node Selection Rule

We decided to develop the tree - search using a depth - first strategy, starting with the leftmost branch, progressively developing the top to bottom branches and working from left to right, slowly building up all complete normalised cutting patterns in A_0. By going to the lowest level in the tree as rapidly as possible, although at the expense of temporarily ignoring potentially more promising branches en route, feasible solutions are generated at early stages in the search, which can then be used to prune the tree and hence reduce the area of search necessary.

3.8.4 Branching Rule

The choice of which rectangle from the List F of available rectangles at a particular node n is to be cut has been left unspecified. Three possible ways in which this rectangle can be chosen are given below:

(i) One simple method is to select the " smallest " rectangle produced at node n using the following procedure: first pick that rectangle r_j with minimum x-dimension

x_j and if more than one such rectangles exist , then choose among these the one with the smallest y-dimension y_j . Similarly, the " largest " rectangle could be chosen.

(ii) An alternative method is to select that rectangle (x, y) for which the constrained DP solution gives the highest V_c(x, y) value; this is a simple and computationally inexpensive method since this value is used in the calculation of bounds.

(iii) A slightly more complex branching strategy that aims to obtain a feasible solution early on in the search is as follows. At node n, it is obvious that

$$\delta_i + \sum_{(x,y)\ \varepsilon\ F\text{-}H_0} \text{Rect}\,(x, y, i) > Q_i$$

for at least one i=1, ..., m ; otherwise no branching would have been possible since a feasible solution would have been obtained. Let i* be that piece i for which

$$\delta_i + \sum_{(x,y)\ \varepsilon\ F\text{-}H_0} \text{Rect}\,(x, y, i) - Q_i$$

is maximum. Then by reducing the number of pieces used for type i*, the largest step forward feasibility would be produced. With this in mind, one could then choose to cut the rectangle that uses the largest number of pieces of type i* in the constrained solution.

3.9 Computational Experience with the Algorithm

We tested the effectiveness of the tree-search algorithm, described in section 3.8, on 15 randomly generated problems using:

BOUNDS: The bound of DP recursion given by equations (16) - (18) with the State-Space Ascent described in section 3.6 (Note that formula A for modifying the weights in the SSA procedure is used).

BRANCHING: Branch selection rule (iii) of section 3.8.4.

The random problems were produced as follows: The dimensions α_i and β_i of each piece in $\bar{R}$ were generated by sampling two numbers from the uniform distributions [1, $0.75\alpha_0$] and [1, $0.75\beta_0$], respectively. The value of each piece v_i was calculated using the formula $v_i = k_i \alpha_i \beta_i$ where k_i is a uniformly distributed random number in the range 1 to 1.5. Finally, the constraints Q_i on each piece in $\bar{R}$ were sampled from the uniform distribution [1, 3]. All values α_i, β_i, v_i and Q_i were then rounded upward to the nearest integers.

The computer program used to produce the computational results was coded in FORTRAN and run on a CYBER 855 computer, under the FTN5 compiler. All computing times shown are in CP seconds.

Table 3.12 describes the performance of the tree-search algorithm on the 15 CGC problems. The first 9 problems have already been used to test the SSA procedure of section 3.6 (Tables 3.10 and 3.11). The table shows, for each problem, the size of the stock rectangle A_0, the number of types of pieces in $\bar{R}$ and the sizes of the normal sets L and W, being calculated once at the beginning of the solution procedure. The best upper bound Z_{min} obtained for each problem, by the

SSA procedure performed at the root node of the tree is given, together with the number of iterations required and the time taken to reach this value; the value of the optimum solution Z_{opt} is also given as well as the number of nodes generated in the search and the total time required to solve the problem (the total time recorded includes the time spent at the initial node of the tree). A measure of the gap between the value Z_{min} and the optimum is calculated for each problem. As a means of comparison, Table 3.13 shows the results obtained, by applying the algorithm described in chapter 2, to these 15 test problems when no constraints are placed on the maximum number of pieces to be cut (UGC problems). Note that the normal sets L and W are expected to be of larger sizes in the case of unconstrained cutting.

As described in section 3.6.3 of computational results obtained by the SSA procedure, a maximum number of 20 iterations was imposed and a maximum value of Q was set by the SSA procedure, for each test problem, as a result of limiting memory requirements. Time limits of 1500 and 800 seconds were imposed at the root node for Problems 1 to 9 and 10 to 15, respectively. The restriction for the second set of problems, of spending less time at the root node was based on the observation that a very fast ascent in the bound occurs within the first few SSA iterations and a small improvement in the value of the bound may be achieved during later iterations at the expense of extra computational cost (the results shown in figures 3.5 to 3.13 are typical of all the problems tested).

From Table 3.12, it is clear that most of the time required to solve a problem optimally, is spent at the root node of the tree. In other words, the major part of the computational cost is used by the SSA procedure, to obtain a good upper bound on the solution of the problem. In fact, 4 out of the 15 problems tested, were solved optimally without requiring any branching, namely Problems 1, 2, 4 and 9. In order

to compare these results with the performance of the tree search algorithm due to *Christofides and Whitlock* [1977], we ran the code for their algorithm on the CYBER 855 computer for Poblems 1 to 7. The optimum solutions for Problems 1, 2 and 4 were obtained by generating a tree search of 253, 3794 and 988 nodes, for each problem, in 2.3, 69.8 and 27.8 CP seconds, respectively. Furthermore, using the same algorithm, Problems 5, 6 and 7 required 58609, 93178 and 43919 nodes each, to be generated in 1430.5, 2519.4 and 1320 CP seconds, respectively, compared to 12067, 22556 and 23315 tree nodes obtained by our algorithm, in 1202.1, 1613.3 and 1270.1 CP seconds for each problem. Only in Problem 3, the SSA procedure required approximately 215 CP seconds more, to generate a bound within 3.4% of the optimum, than the overall time needed by their tree-search algorithm to find the optimal solution for the problem.

From the above results, it is clear that the SSA procedure is an efficient method for producing good upper bounds on the optimal solution of CGC problems. Clearly, the quality of the bound (described by the duality gap) depends on the size of the problem. For larger problems, the number of SSA iterations, performed at the root node, tends to be smaller as a result of limiting memory requirements and high computational cost, and the number of nodes generated in the search tree larger.

An example of the data and the optimum solution for a CGC problem (Problem 8) obtained by the tree-search algorithm, described in this chapter, is presented in the last section of chapter 4.

Problem Number	Problem Data				Initial Tree Node					Tree Search		
	(α_0, β_0)	m	|L|	|W|	Upper Bound (Z_{min})	Duality gap (r) %	Number of SSA Iters	Q(UB1)	Time CYBER-855 seconds	Optimum solution Z_{opt}	Number of Tree Nodes	Total time CYBER-855 seconds
1	(10,10)	5	8	9	135*	-	2	2	0.5	135	-	0.5
2	(15,10)	7	12	10	244*	-	4	5	3.4	244	-	3.4
3	(20,20)	7	17	17	517	3.4%	20	48	680	500	4970	695.7
4	(20,30)	10	11	23	1755*	-	4	3	4.8	1755	-	4.8
5	(30,30)	7	23	15	1117[b]	4%	17	47	1177	1074	12067	1202.1
6	(30,40)	8	26	17	1421[a]	5.1%	19	41	1500	1351	22556	1613.3
7	(30,50)	10	26	17	1716	3.8%	20	27	1037.5	1653	23315	1270.1
8	(40,70)	10	29	56	2902[a]	0.3%	7	19	1500	2892	31755	2101.4
9	(40,70)	20	25	55	1860*	-	9	20	1500	1860	-	1500.
10	(40,60)	5	12	13	2513	2%	20	20	86.1	2462	14	94.7
11	(50,70)	8	43	35	4238[a]	3.6%	6	16	800	4091	303,435	1552.3
12	(70,80)	8	54	38	6740[a]	4%	5	13	800	6478	138,697	1444.0
13	(60,80)	10	52	38	5957[a]	6.3%	5	16	800	5604	421,593	2170.7
14	(70,90)	10	54	38	7710[a]	7%	5	17	800	7200	171,801	1321.8
15	(80,100)	10	64	38	9811[a]	8%	4	13	800	9077	135,485	1653.7

Table 3.12 Computational Results of Tree Search Algorithm

a Time limit at Root node

b Q attained its maximum value allowed for the computation of Bound

* Optimum solution found at Root node

Problem Number	Problem Data				Optimum Solution	Total Time CYBER-855
	(α_0,β_0)	m	|L|	|W|	Z_{opt}	seconds
1	(10,10)	5	8	9	145	0.2
2	(15,10)	7	14	10	249	0.4
3	(20,20)	7	17	18	559	0.9
4	(20,30)	10	13	23	3920	0.6
5	(30,30)	7	23	21	1275	1.0
6	(30,40)	8	26	26	1650	1.7
7	(30,50)	10	26	30	2025	2.0
8	(40,70)	10	29	56	3076	7.6
9	(40,70)	20	26	55	2240	4.7
10	(40,60)	5	16	28	2910	1.3
11	(50,70)	8	50	50	4698	15.3
12	(70,80)	8	70	53	7616	37.9
13	(60,80)	10	60	53	6282	39.6
14	(70,90)	10	70	58	8526	42.7
15	(80,100)	10	80	61	10689	58.8

Table 3.13 Unconstrained Results for Problems 1 to 15 of Table 3.12.

3.10 Conclusions

In this chapter, we studied the application of the State-Space Relaxation technique to the CGC problem. A state space ascent method was used to optimise the bounds derived from it, which were then embedded into a tree-search algorithm developed to solve the problem optimally. The computational experience of the algorithm, shows that:

(i) SSR performs reasonably well for CGC problems of medium size.

(ii) The algorithm is an effective procedure capable of producing a considerable improvement on the results obtained by the algorithm presented in *Christofides and Whitlock* [1977].

CHAPTER 4

TWO-DIMENSIONAL RECTANGULAR LAYOUT GENERATION USING MICROCOMPUTER GRAPHICS

4.1 Introduction

A graphics problem of great interest to industry is that of optimum two-dimensional layout. In many practical applications, an operator is given a number of rectangular sheets and an order for a specified number of smaller rectangular pieces. The objective is to cut the pieces out of the sheets in such a way as to minimise the amount of waste produced and thus the number of sheets used. Such problems appear in the cutting of steel, wood or glass sheets. A generalised version of the above problem involves cutting from a number of large rectangles an order for a specified number of smaller pieces, each of given size and value, the objective being to maximise the total value of the pieces cut. If the " value " of a piece is proportional to its area, then " value" maximisation is equivalent to waste minimisation. In this chapter we will consider only cutting problems with a single

stock rectangle. Depending on the technological process involved in cutting and on the nature of the material to be cut, certain applications require the cut to be made along a straight line from one edge of the sheet to another. Such cuts, referred to as " guillotine cuts ", are always required in the case of cutting glass and quite often in the case of cutting wood or thin metals.

The increasing use of powerful interactive microcomputer systems allows a much wider use of graphics. In this chapter, we present an interactive system with graphical input-output for generating rectangular layouts for the Two-Dimensional Guillotine Cutting Problem (GCP) described in chapter 3. The structure and the main features of the system are described in this chapter. An experimental version of the system, referred to as the Graphical Layout Generator has been designed and implemented on an IBM-PC computer with a Colour Graphics Adapter and Screen. For testing the effectiveness of the package we carried out the following two experiments.

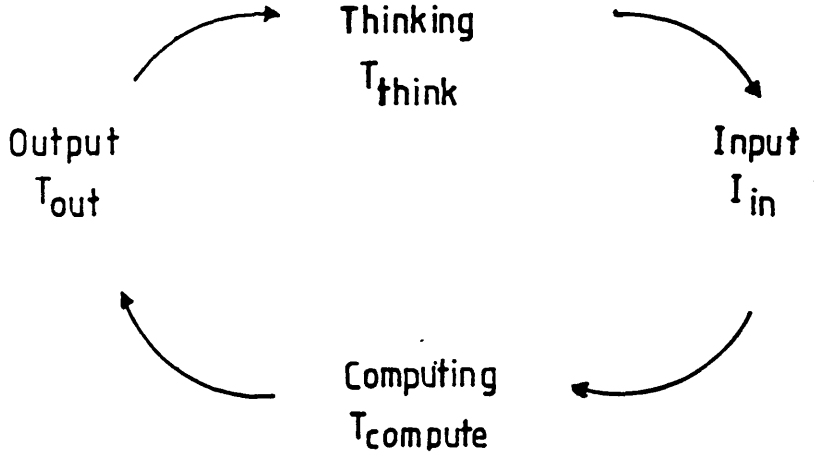
(i) A sample of 10 operators were required to solve 3 GCP's using the Graphical Layout Generator, each problem involving 10 types of pieces to be cut from a stock rectangle. The total time taken and the quality of the solution obtained by each operator were recorded.

(ii) Two more experienced operators were asked to solve 9 GCP's using the Graphical Layout Generator. The results recorded were compared with the computational results obtained by applying the algorithm, described in chapter 3, to the same set of test problems.

Before describing the proposed system, we present some basic characteristics of interactive computing using graphics terminals.

4.2 Computer Graphics

All use of computers can be regarded as a dialogue between a person and a machine - information is requested by a computer user and results are returned by the program, control parameters are changed and different results are returned and so on. This dialogue is shown below as a communication cycle, together with the time delays associated with the different processes in the cycle.



This model applies equally to interactive or batch systems. The total time taken for an idea to circulate round this loop is given by

$$T = T_{think} + T_{in} + T_{compute} + T_{out}$$

and the reciprocal of this is a measure of the speed at which the ideas are being developed.

The user's time is divided into four phases - absorbing the information returned by the computer, applying specialist knowledge to the problem, expressing the idea in the form required by the computer program and waiting for the results to

be returned. The computer system can assist the user in absorbing its results by suitable presentation and can speed the expression of new ideas by allowing the user freedom and flexibility in the language used.

The human operator has many peripheral devices for the reception and transmission of data. Although this task is carried out by the use of natural language, a wider view of communication with the computer can be achieved by returning results from the computer as pictures and allowing the user to present ideas by more general signals than simple typing.

Taking this brief introduction as a justification of computer graphics as a desirable means to enhance the environment of a computer user, we give an outline of the basic hardware and software available for use by an application program.

4.3 System Design

4.3.1 Background

Graphical methods were being used to illustrate the behaviour of O.R. models in the early 1960's. During the 1970's the major development in computer graphics was vector displays; these had a major impact on geometric applications such as computer aided design; however they had little impact on O.R. It was only with the arrival of raster displays (section 4.3.2) in the late 1970's and early 1980's that there was a significant growth in the number of O.R. graphics applications. This growth was motivated by the reduction in cost, increased speed, improved software and physical portability of colour raster display graphics devices -

especially when controlled by microcomputers.

4.3.2 Hardware

Firstly, a distinction is drawn between vector scan and raster scan output devices. Vector scan devices display a series of lines as if they had been drawn by a set of differently coloured pens, whereas raster scan devices generate solid blocks of coloured areas. Secondly, we can distinguish between hard copy and refreshed devices.

Since the Graphical Layout Generator application package was designed for computer screens making use of colour, as an aid in manipulating and interpreting the graphical data, in this section we will present the basic characteristics of a raster scan display. The underlying idea is to divide the image space into a uniform grid of small rectangles (called pixels), each of which is filled with a single colour. The smaller the size of these pixels the higher the resolution of the displayed image. The chromatic resolution is the number of different colours that can be put into an individual pixel.

An IBM-PC, having an attached device driver that supports the IBM Color / Graphics Monitor Adapter (CGA), was used for the implementation of the Graphical Layout Generator package. The graphical output on the IBM monitor, is buffered in an array of 320 pixels across (pixel columns) by 200 pixels down (pixel rows), with 4 bits per pixel producing the possibility of using 4 colours out of a total possible number of 16 (Medium Resolution Mode of the CGA).

4.3.3 Software

The past few years have seen the presentation of various graphics standards for graphical input and output. The design for manipulating the graphical data used by a GCP was based on an implementation of the Graphical Kernel System - GKS. This system restricts its attention to two-dimensional images. Its use renders the following structure: Picture components are specified using a model coordinate system; the procedure is then to clip the image to a specified window, store intermediate pictures in some device independent code, transform this to screen coordinates for display and allow several independent viewpoints to be maintained on a single screen.

It is possible to arrange the software that handles interactive graphics in such a way that an application program receives the information in a standard way, independently of the actual hardware used. The use of GKS, providing device independent graphics, led to an efficient design of the Graphical Layout Generator package; its implementation on the IBM-PC allows for various types of graphics devices to interface to the system.

In the following section, we will describe the structure and the main features of the Graphical Layout Generator package.

4.4 User Interface Design

The Graphical Layout Generator is an interactive graphics package for the two-dimensional Guillotine Cutting Problem (GCP). Its purpose is to facilitate

generating non-algorithmic solutions by having an interactive system with graphic input-output to assist in the tedious and time-consuming manual method of trying to cut optimally a stock rectangular sheet into a number of smaller rectangular pieces. The manual process becomes more time-consuming and prone to errors as the number of shapes to be cut from a sheet increases. The objective of the user interface design is to retain the flexibility of the manual system, whilst exploiting the processing power and methods of interaction offered by the computer.

The structure of the package is described below:

(i) The system displays the data for a particular problem.
(ii) The user gives information to the system interactively for generating a layout. Once an input command is executed by the system, the displayed result can not be modified while generating the current layout and the user can then only restart from the beginning if he wishes.
(iii) The system checks for possible error conditions during the generation of the current layout.
(iv) Once the solution procedure is terminated, the system calculates the total value associated with the generated layout and informs the user of the amount of deviation of his solution from optimality. The optimal solution is then displayed on the graphics screen.

4.4.1 Problem Description

The Graphical Layout Generator is designed to handle only rectangular shapes. The data for a particular problem is displayed on the screen and is described as follows (see Fig. 4.1): A large rectangular sheet A_0, described by its length α_0

and its width β_0 is given together with a set R of m types of smaller rectangular pieces $R = \{ (\alpha_1, \beta_1), ..., (\alpha_m, \beta_m) \}$ that can be cut from A_0. A type of piece is identified by a number, representing the order in which it is presented; this identification number will be used in the interactive stage. Every piece of type i in R is described by its length α_i and width β_i ; the reference point of a rectangular shape is assumed to be its lower lefthand corner. Two more integer numbers are associated with each type i, namely its value υ_i and the constraint Q_i on the number of pieces of this type that can be cut from A_0. An illustrative example of the graphical data is presented in Figure 4.1. The objective for the operator is then to construct a guillotine cutting pattern for A_0 with the highest possible total value using pieces from set R.

In order to distinguish between the given pieces in R and the rectangles produced by the cuts on A_0 at any stage during the cutting process, the former are henceforth referred to as ' pieces ' and the latter as ' rectangles '. To solve a given problem, the operator must have in mind the following assumptions:

(i) A coordinate system is introduced, for convenience, with x-axis along the bottom edge and y-axis along the left edge of the large sheet A_0. In this way the lower left-hand corner of A_0 is referenced as the point (0,0). Using this referencing method, cuts on the rectangles can be made in integer steps (all dimensions (α_i, β_i), i = 0, ..., m are integers) along the x of y axes. All cuts must be guillotine cuts (a cut goes from one edge to the opposite one). There are no constraints on the sequence of cutting.

(ii) The pieces are not allowed to rotate (90 degrees) i.e. the orientation of the pieces is fixed.

(iii) The value of a piece is not necessarily proportional to its area.

(iv) Not all pieces available need to be used.

4.4.2 Interactive Solution Approach

Once data for a particular GCP is displayed on the screen, the operator is looking for a method of generating a layout having value as high as possible. The apparently simple problem of cutting in the most efficient manner is, in fact, extremely complicated to solve optimally. The operator is immediately faced with a huge set of possible layouts, although restricting the permissible cuts to be of 'guillotine' type, drastically reduces the number of available combinations. The choice of a method for generating a layout clearly depends on the operator ; it can be intuitive in nature, since we are unable to characterise the optimum or near-optimum layouts. An experienced user, having a better insight of the GCP, may develop a more effective heuristic approach. However, the mechanisms that guide the search for an efficient method must be derived from the problem environment. Heuristic schemes are very dependent on the particular problem being solved.

The procedure available to the operator for the cutting of rectangles and the allocation of pieces using the Graphical Layout Generator package consists of the following three stages: rectangle generation (a rectangle is cut into two smaller rectangles), piece allocation (a rectangle is filled by one of the required pieces given in R) and termination of the current layout.. Each course of action is described below in conjunction with the command language used in the conversational process to carry out the relevant action. The notation used for the commands is simple. Each command is entered by the user through an input device (keyboard). Once it has been executed, the system displays its output and returns to a 'waiting state' ready to execute another command.

A. Rectangles Generation : At any stage of the cutting process on A_0, at most three parts can be distinguished : A part that is already filled during the allocation process,

a part that has been discarded as waste and a part still to be examined. The current part under examination consists of a number of rectangular shapes, each being identified by a unique label (single letter) assigned to it as soon as it is generated. The user then has to decide (i) which type to cut next, (ii) the type of guillotine cut to be performed (i.e. parallel to the x-axis or to the y-axis) and (iii) the position of the cut.

The format of the command used to input such a decision is given by " S t C " where

(i) S represents the label of the rectangle chosen for cutting.

(ii) t describes the type of cut made along a straight line from one edge of the rectangle to the other: t = v if the cut is vertical, t = h if the cut is horizontal with respect to the coordinate axes.

(iii) C is the co-ordinate of the cut which divides rectangle S into two further rectangles. Note that the coordinate system described in section 4.4.1 is used. C can take values in the range $\{1, \ldots, \alpha_0 - 1\}$ or $\{1, \ldots, \beta_0 - 1\}$. An example of the command input by the user in order to generate two new rectangles by making a vertical cut on a rectangle A, at a point lying 15 units away from its bottom left hand corner, is given by " A v 15 " . In response to this command, the current drawing on the screen is modified to display the new rectangles resulting from the cutting of rectangle A which are automatically assigned identification labels by the system.

B. Piece Allocation : Once a rectangle has been generated and labelled, the user then has to decide whether to cut it further or to fill it by one of the initial pieces given in R. In the latter case, the system allows the user to allocate a piece by inputting the command " S = i " where i is the identification number of the piece and S the identification label of the rectangle.

The generated rectangles can only be filled one at a time, and only one piece can be allocated per rectangle. If a rectangle and its allocated piece are of different sizes, then the unfilled area of the rectangle is discarded as waste and need not be cut away by the user. Once a rectangle has been through the allocation process, it is excluded from further consideration during the generation of the current layout.

C. Termination of the current layout : The user may terminate the generation of the current layout at any point in the interactive stage by inputting the command STOP when the system is in a 'waiting state' ready to execute a command. Termination is requested by the user in two cases :

(i) when the generation of the current layout is actually completed i.e. no more rectangles can be generated or there are no more pieces to be allocated or

(ii) when a modification of the curent drawing on the screen is desirable. In this case, the user has to regenerate the current layout from the beginning.

4.4.3 Checking of Error Conditions

The design of the Graphical Layout Generator calls a command interpreter routine in the interactive stage in order to handle the user commands and detect error conditions in their processing. When one of the following errors occurs a relevant message is displayed on the screen :

Rectangles generation : (i) The format of the input command is not correct.

(ii) The input identification label for the cut rectangle is wrong.

(iii) The coordinate of the input cut is out of the permissible range (infeasible cut).

Piece allocation : (i) The format of the input command is not correct.

(ii) Allocation is not possible because of a mismatch between the required piece and rectangle.

(iii) Allocation is not possible because there are no more pieces of the requested type available to be allocated.

4.4.4 Display of Optimum Solution

Once a complete interactive solution has been produced for a GCP, the system computes the total value associated with the generated layout and gives the value and the percentage deviation of this value from optimality. The optimal layout for the given problem is then displayed on the graphics monitor. The value and the structure of the optimal solution for a given GCP are obtained using the exact algorithm, described in chapter 3.

4.5 Experimental Results

In this chapter, a system has been described to produce interactively guillotine cutting patterns of two - dimensional rectangular shapes on larger rectangular sheets. The present program was designed so that an experimental version of the system would be available for demonstrative and educational purposes. An efficient operational package was then developed by implementing the Graphical Layout Generator on an IBM-PC computer in IBM FORTAN 2.00. The package was organised so that it can handle a GCP involving at most up to 10 different types of pieces to be cut from a single stock-plate of up to 100 units of size

in x and y dimensions.

Two types of experiments were performed using the package, allowing conclusions to be drawn on the quality of graphically generated solutions to GCP's. The first experiment examined the human performance in producing optimal solutions to three such problems. The second experiment evaluated the performance of two more experienced operators on a sequence of test problems.

4.5.1 Design of Experiments

The test problems used in both experiments have been randomly generated as described in section 3.9 of chapter 3 and the optimal solutions to these problems are given in Table 3.12 of computational results of the same chapter.

In the first experiment, 10 operators were tested. Each operator was asked to solve successively three GCP's of different complexity, each problem being attempted only once. These problems deal with stock-rectangles of sizes 30 by 50 units, 40 by 70 units and 70 by 90 units, respectively. Each problem involves 10 different types of pieces that can be used for cutting the given stock rectangle; there are 1, 2 or 3 pieces of a particular type available. The three test problems chosen were Problems 7, 8 and 14 of Table 3.12 (Chapter 3).

In the second experiment, each of two operators was asked to solve a set of 9 problems. The number of different types of pieces involved in each problem vary from 5 to 10 and the stock-rectangle sizes range from 10 by 10 units to 70 by 80 units. The 9 test problems chosen were Problems 1to 6 and 11 to 13 of Table 3.12.

4.5.2 Display Format

The main objective of an effective display design is the display of the qualitative and quantitative type of information being coded at a level suited to the viewer's needs, without allowing display complexity to interfere with the viewer's performance. The use of colour aids in coding the displayed information.

The same display format was used for all GCP's and a typical example is shown in Figure 4.1. A message " CUT: " followed by a black cursor is placed below the stock rectangle to prompt the operator for a response. As the operator types an input command to a 'waiting' state of the system, the response appears as white text in the black cursor.

4.5.3 Experimental Procedure

The operators tested were all volunteers. The group was drawn from Imperial College and consisted of academic staff as well as post-graduate students. 7 out of the 10 operators tested in the first experiment were familiar with computers. 3 out of these 7 had a considerable amount of experience in the field of combinatorial optimisation and O.R. techniques whereas the other 4 were involved in the fields of Engineering and Finance. The remaining 3 operators not familiar with computers were from the Behavioural Sciences.

The operators tested in the second experiment both carried out research in the area of O.R. It is obvious that the selection of operators in this case was biased. Indeed, the original intention was to obtain the best possible graphical solutions to a given set of test problems using the Graphical Layout Generator, so that the

effectiveness of an interactive graphical approach in solving GCP's can be evaluated. The above selection seemed to offer an expected improved performance.

Each experiment began with an introduction given by the experimenter (author) explaining what the operator had to do. The operator was then presented with a practical example used to establish the method of responding to the Graphical Layout Generator. Whilst the tests were being performed, the experimenter's role was not to interfere with the solution procedure but to offer assistance if needed. The operator was not restricted to solve a particular test problem within a certain period of time and each problem was attempted only once. The experimenter was interested in the overall performance of the operator as well as in the performance with time. Thus, time was divided into intervals of 5 minutes and the quality of a graphical solution generated by the operator was recorded for the first 3 time intervals. Once a solution procedure for a given problem is terminated, the operator's performance is evaluated based mainly on two figures: the % within optimality of the solution value attained and the total time taken.

During the tests, the experimenter recorded any comments the operator made regarding the test and his solution approach.

4.5.4 Method of Response

The operator indicated when a response has been selected by pressing the "ENTER" key. Prior to this a 'backspace' key allowed the operator to delete a displayed response without the computer executing the command. After "ENTER" was pressed, the system displayed the result (either a new cut was generated or a piece was allocated into a cut rectangle) on the screen and was then ready for the

next response.

If an operator realised an undesired response had been typed after pressing "ENTER" the package did not provide any facility for backtracking to the previous prompt to revise the response (this is an interesting improvement to be implemented).

4.5.5 Results of Experiments

The combined results for all 10 operators of experiment 1 are summarised in Table 4.1. The results obtained for the 9 test problems of the second experiment by 2 operators are given in Table 4.2.

Average Sol. Time (mins) = Total Sol. Time (mins) / 10

Average % deviation from optimality = 100% * (1 - Sum of all Sol. Values / (10 * Opt. Sol. Value))

Diff between Best & Worst Solutions (Range) = 100% * (Highest Sol. Value - Lowest Sol. Value) / Opt. Sol. Value

Results of the first experiment show that Problem 8 was clearly the most complicated of all three with an average solution value being 16.9% away from the optimum and requiring 5 more minutes on average to achieve. We must note that in the data of this problem, most of the given pieces are relatively small compared to the stock-rectangle. As the number of small pieces increases, a considerably larger number of combinations of using them in cutting the large rectangular sheet are available to the user, thus making the problem more difficult.

The average performance for all problems indicates the following:

(i) There was a delay of 5 minutes for 70% of the operators for each of the 3 problems tested, between the time the operator was initially given the problem until he actually started generating a solution. This interval of time was mainly spent by the user in thinking about the various combinations of cutting the pieces or which particular cutting policy to follow.

(ii) The three following types of solution approach to a GCP were adopted by the users:

(a) First the pieces of larger area were considered for cutting. The operator using this policy was basically interested in minimising the total amount of waste produced. However, minimising the waste by cutting a large rectangle into smaller pieces, or maximising the total value of the pieces produced will not necessarily lead to the same optimum cutting pattern. For example, as can be seen from Table 4.2, operator 1 obtained the optimum solution for test problem 1 with an associated waste rate of 2%. Operator 2 generated a graphical solution for the same problem with a value 6.7% away from optimality. In this case full utilisation of the stock rectangle was made.

(b) Guillotine cuts were generated in a sequence that allowed the pieces of larger value per unit area to be cut first i.e pieces were chosen in decreasing order of the ratio of the value of a piece over its area.

(c) Guillotine cuts were performed in a sequence that allowed pieces to be cut from the larger rectangle in decreasing order of value. This approach was applied successfully by operator 2 to solve all 9 test problems of the second experiment. The results show that 3 of these problems were solved optimally and the average solution value obtained for all problems is 3% away from the optimum. Clearly this heuristic proved to be the most efficient in solving graphically GCP's.

In general, the results of the first experiment show that the average solution

value obtained for a problem involving 10 types of pieces to be cut from a single stock rectangle (of up to size 100 by 100) is expected to be at least 9% away from optimality. The expected time taken by a user to generate such a solution using the Graphical Layout Generator is approximately 14 minutes.

An example (problem 8) is used to illustrate how the Graphical Layout Generator package works. The displayed data for the illustrative example is described in Figure 4.1. Figure 4.2 shows how an interactive solution to the problem is developed by presenting the various drawings on the screen (working display) of the large stock rectangle A after a user command is input to the system. 17 commands in total were required to produce a feasible solution to the problem. Each displayed layout results from the execution of the command shown underneath. The identification of a rectangle by a letter indicates that it is available for further cutting; identification by a number implies that a piece has already been allocated to it. The number next to a cut indicates the point of cutting. Waste is not cut away by the user; it is represented by the shaded area. The last layout represents the structure of the generated solution with an associated value being 8% away from optimality. The structure of the optimal layout for the problem is shown in Fig. 4.3.

4.5.6 Conclusions

A system has been described to produce solutions to two-dimensional GCP's using interactive graphics. No particular effort was made to design the program so that it can be available for specific applications. The effectiveness of an experimental version of the system was tested using a small sample of operators. Results show that the package developed seems to be sufficient for interesting interactions. However, experience and applications to real problems may suggest

both command extensions and modifications in the implementation of the interactive part of the package (the package may need to be oriented to specific industrial requirements).

Problem Number	Problem Data		Ave Time (minutes)	Ave % deviation from optimality	Diff between Best & Worst Solutions (%)
	(α_0, β_0)	m			
7	(30, 50)	10	12.2	4.4%	18.7%
8	(40, 70)	10	17.8	16.9%	33.6%
14	(70, 90)	10	12.7	8.1%	20.3%
Average Figures over Problems 7, 8 and 14			14.2	9.8%	24.2%

Table 4.1 Results of Experiment 1.

Problem Number	Problem Data		Results obtained by Operator 1		Results obtained by Operator 2	
	(α_0, β_0)	m	Total Time (minutes)	Ave % deviation from optim.	Total Time (minutes)	Ave % deviation from optim.
1	(10, 10)	5	2	0.0%	3	6.7%
2	(15, 10)	7	6	11.9%	8	0.0%
3	(20, 20)	7	3	6.6%	8	9.2%
4	(20, 30)	10	5	5.7%	5	0.0%
5	(30, 30)	7	6	7.7%	5	5.1%
6	(30, 40)	8	8	8.4%	7	0.0%
11	(50, 70)	8	6	1.4%	8	1.4%
12	(70, 80)	8	9	4.2%	8	1.9%
13	(60, 80)	10	13	7.4%	10	2.0%
Average Figures over All Problems			6.4	5.9%	6.9	2.9%

Table 4.2 Results of Experiment 2.

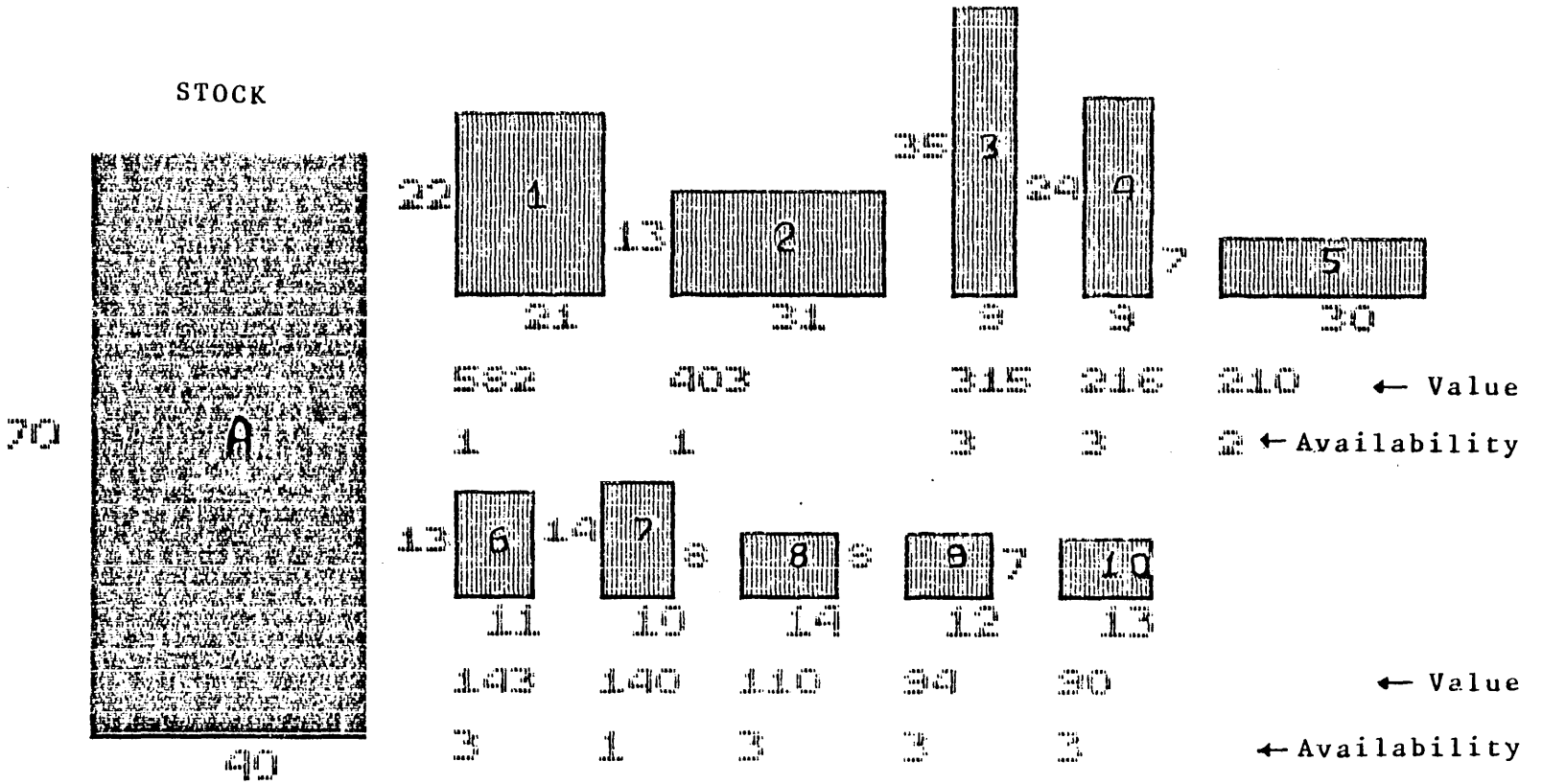


Figure 4.1 Display Format of Data for Illustrative Example (Problem 8).

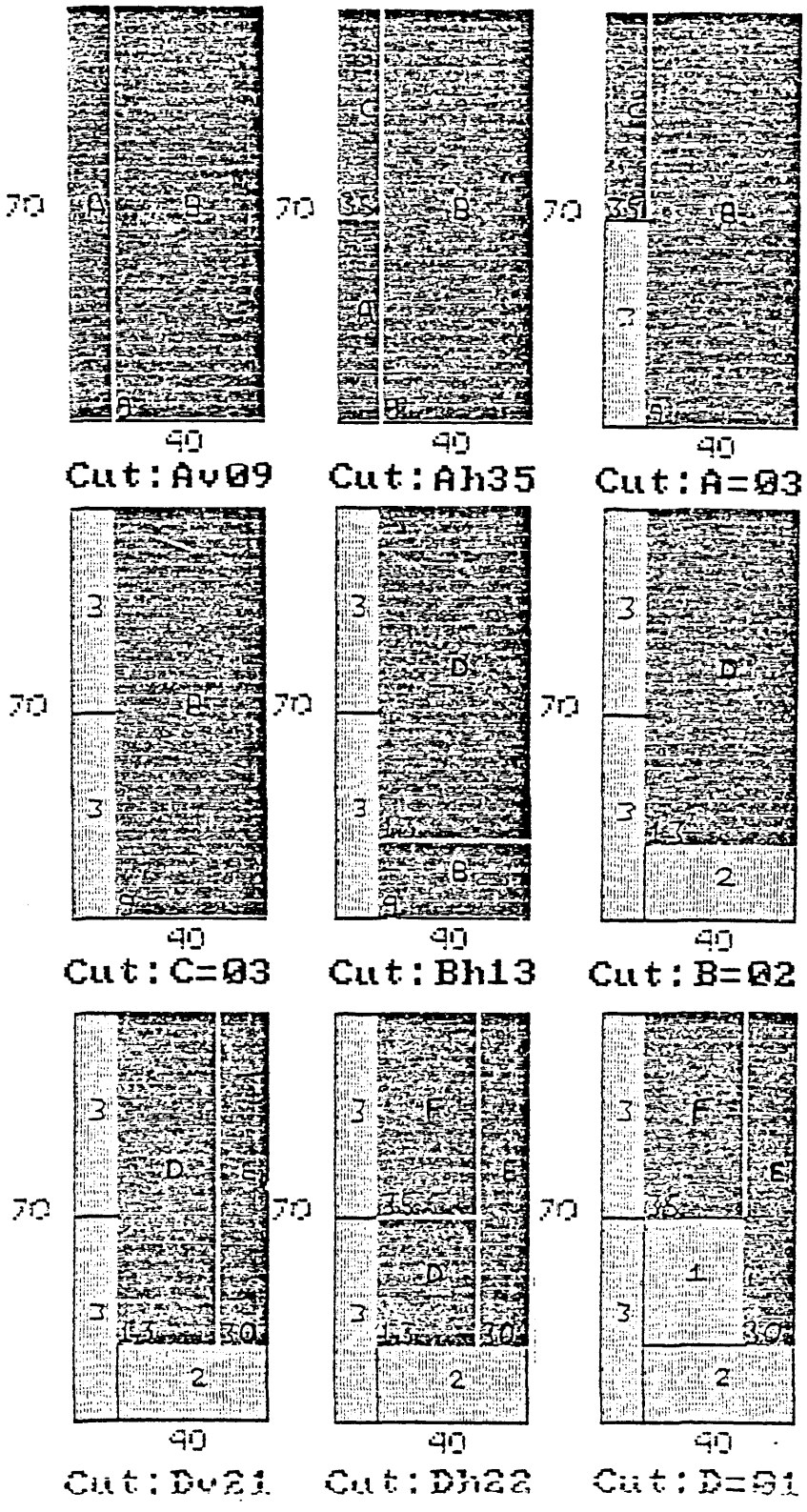


Figure 4.2 An Interactive Solution to Illustrative Example - Problem 8.

(continues on next page)

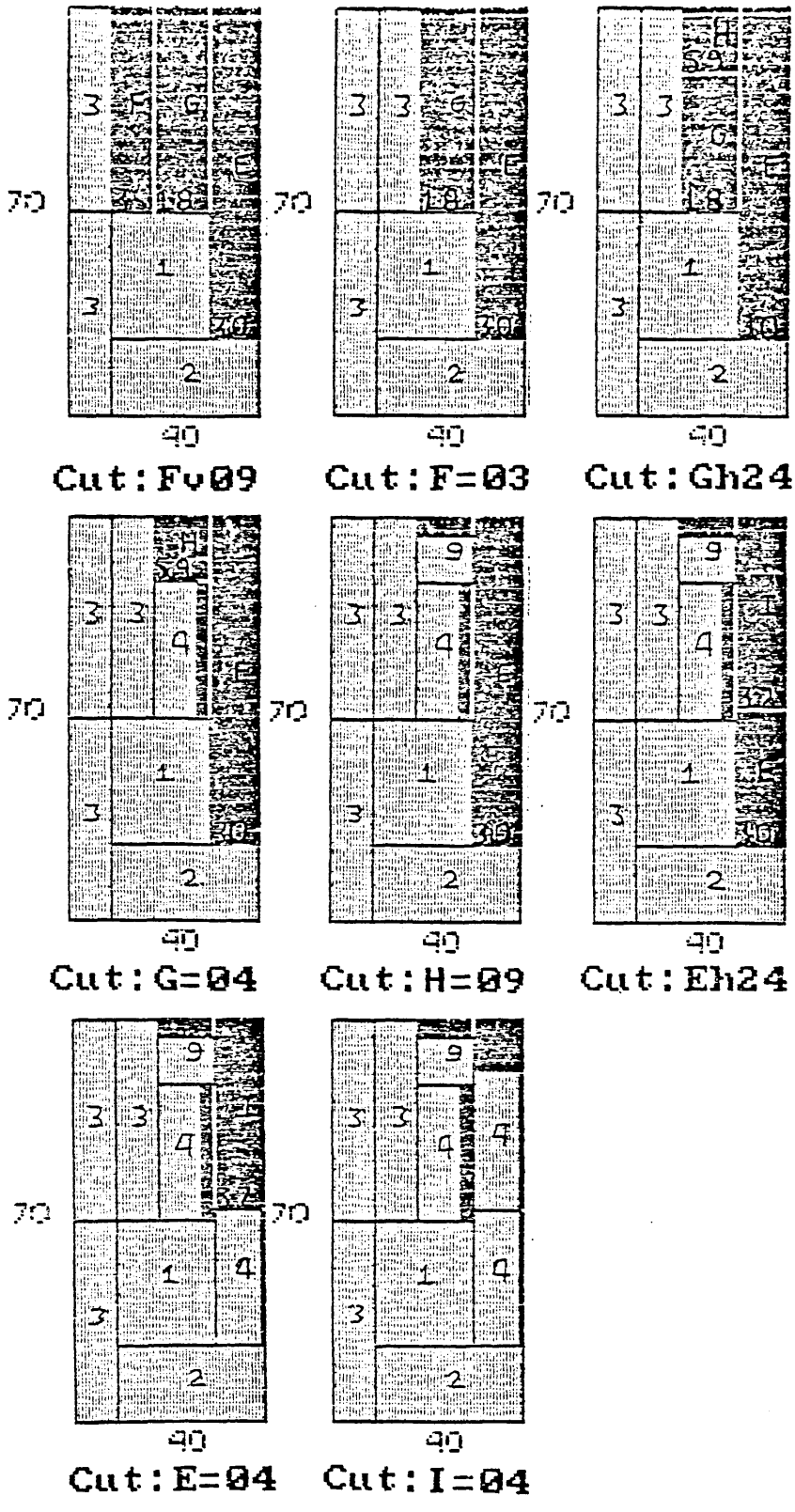


Figure 4.2 continued

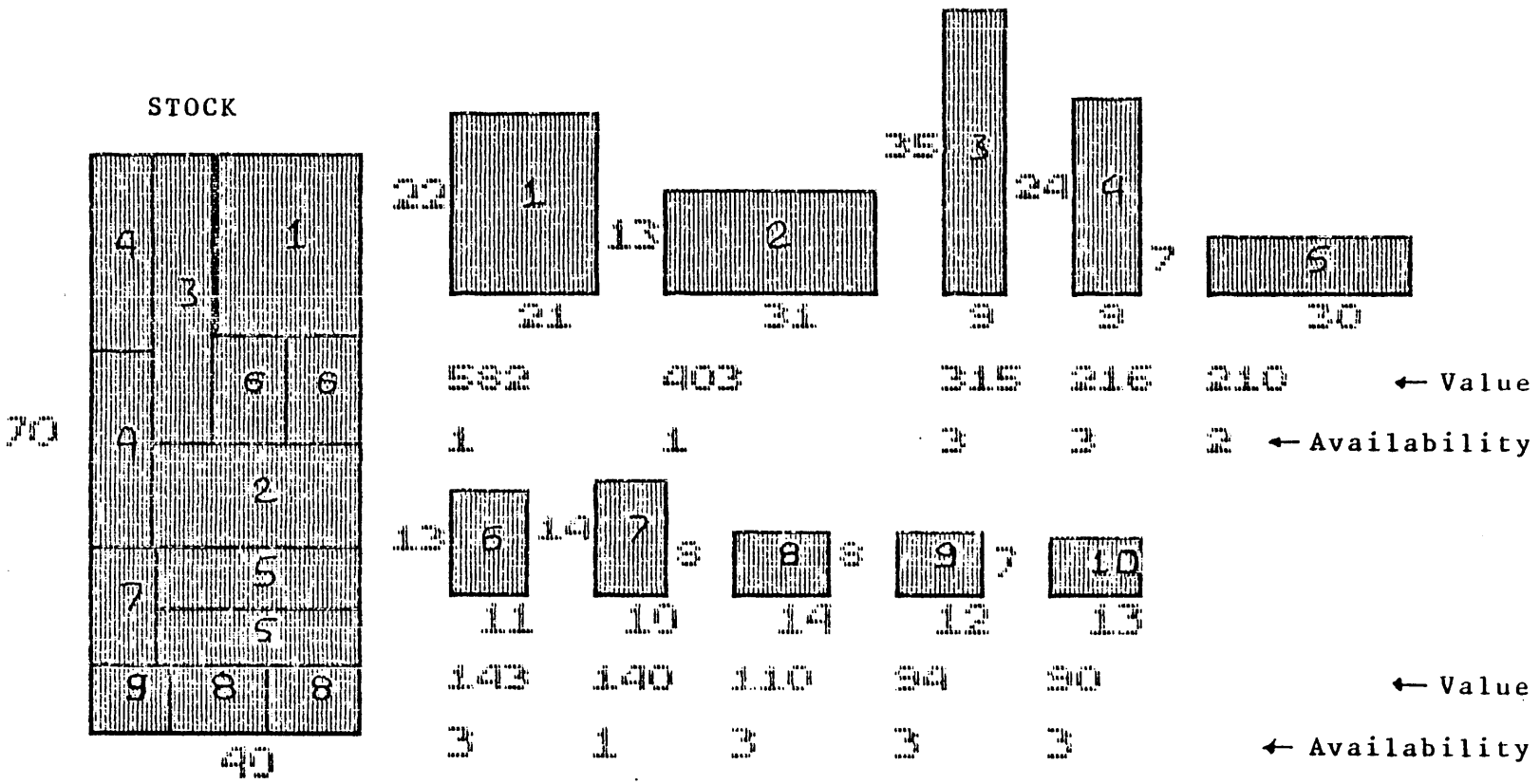


Figure 4.3 Optimal Cutting Pattern for Illustrative Example (Problem 8).

CHAPTER 5

SOME INTEGER PROGRAMMING FORMULATIONS AND BOUNDS FOR THE NON-GUILLOTINE CUTTING (NGC) PROBLEM

5.1 Introduction

In this chapter, we consider the following two-dimensional cutting problem, P. We are given a set of rectangular pieces $R = \{ R_1, ..., R_M \mid R_i$ is a rectangle of length α_i and width $\beta_i \}$ that may be cut out of a large rectangle A_0 of length α_0 and width β_0. Each piece (α_i, β_i) in R has associated with it a value υ_i. We assume that R contains M pieces in total, with each piece being cut from A_0 at most once. If a piece may be cut more than once, it is repeated in R as many times as necessary and given different labels, even though their dimensions are the same. We then require to find a cutting pattern of rectangles that maximises the value of the pieces cut from A_0.

In order to distinguish between the given pieces in set R and the rectangles produced by the cuts on A_0 at any stage during the cutting process, we will refer to the former as " pieces " and the latter as " rectangles ". We define a cutting pattern to be " orthogonal " if each rectangle R_i has each pair of edges parallel to the sides of A_0 and we place the restriction of only allowing orthogonal cutting patterns of the required rectangles on the stock rectangle.

As has already been explained in chapter 2, a guillotineable cutting pattern is an orthogonal pattern with the additional restriction that each cut made on a rectangle must start at one of its sides and then run parallel to an edge until it reaches the opposite side. In this chapter, we consider cutting patterns which may be of "non-guillotine" type. The potential direct applications of these more general problems are those which allow the cutting tool to turn within the material to be cut. Two examples are the cutting of carpet rolls and the sawing of wood plates to produce furniture.

We will make the following assumptions for problem P:

(i) All dimensions (α_i, β_i) for i = 0, ..., M are integers and the cuts on the rectangles are to be made in integer steps along the x or y axes.
(ii) Rotation (by 90°) of the pieces in R is not permitted.

These two limitations are not critical. It is clear that, in practice, the actual dimensions can be scaled up and truncated to integers.

Problem P, as described above, will be referred to as the Non-Guillotine

Cutting (NGC) problem. We underline that problem P is NP-hard and is a special case of the well-known bin-packing problem. It has been considered by relatively few authors in the literature who have proposed heuristic methods and approximation algorithms with worst-case performance bounds to solve various special cases of the problem (see Introduction and references *Baker et al* [1981], *Coffman et al* [1980], *Biro and Boros* [1984], *Smith* [1980]), *Bischoff and Dowsland* [1982] and *Dowsland* [1982]). *Beasley* [1985b]developed an exact tree-search procedure to solve this type of problem .

In this chapter, we first present two Mixed Integer Programming (MIP) formulations of the NGC problem, and investigate a possible method of solution based on the use of cutting planes. Five zero-one Integer Programming (IP) formulations of the same problem are also given and bounds are derived from their Linear Programming (LP) relaxations. These bounds are computationally evaluated on a number of randomly generated NGC problems of small size and results are presented in the last section.

5.2 A Mixed Integer Programming Formulation for the NGC Problem (MIP-1)

Let i and j represent two pieces in R and let the coordinates of their centres in a placement into A_0 be (x_i,y_i) and (x_j,y_j), respectively (taking the bottom left hand corner of A_0 as the origin - see Figure 5.1). The non-overlap conditions between i and j are given by:

$$|x_j - x_i| \geq a_{ij} \text{ and } |y_j - y_i| \geq b_{ij}$$

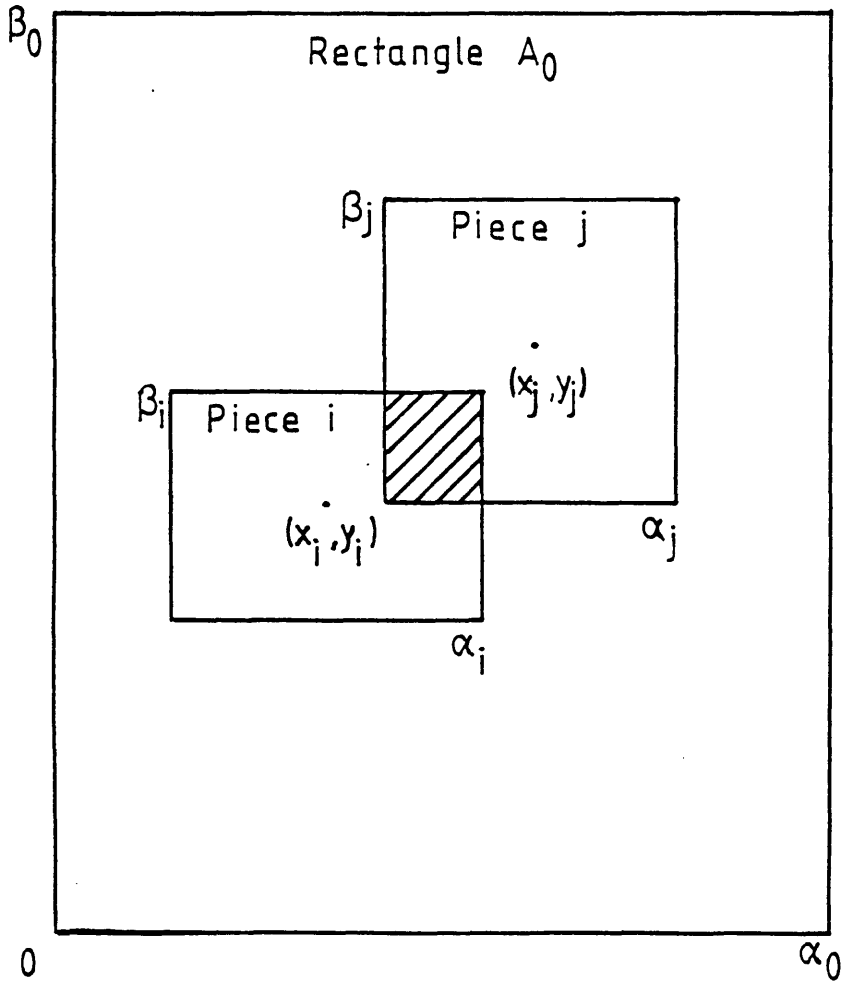


Figure 5.1 Non - overlapping conditions between pieces i and j.

where $a_{ij} = (\alpha_i + \alpha_j) / 2$ and $b_{ij} = (\beta_i + \beta_j) / 2$. These constraints are not in a very suitable form for optimisation by mathematical programming techniques and we need to define the following 0-1 integer variables:

Let δ_j = 1 if piece j is chosen to be cut out from A_0,
= 0 otherwise.

π_{ij} = 1 if piece j is cut to the right of piece i ($x_j \geq x_i$),
= 0 otherwise.

μ_{ij} = 1 if piece j is cut above piece i ($y_j \geq y_i$),
= 0 otherwise.

z_{ij} = 1 if pieces i and j are cut from A_0 in such a way that j lies either totally to the right or totally to the left of i ($| x_j - x_i | \geq a_{ij}$),
= 0 otherwise.

w_{ij} = 1 if pieces i and j are cut from A_0 so that j lies totally above or totally below i ($| y_j - y_i | \geq b_{ij}$),
= 0 otherwise.

The NGC problem can then be expressed as:

Problem P_1

$$\text{Max} \quad Z = \sum_{j=1}^{M} \upsilon_j \delta_j \tag{5.1}$$

subject to:

$$-\alpha_0 + a_{ij} z_{ij} \le x_j - x_i - \alpha_0 \pi_{ij} \le -a_{ij} z_{ij} \quad \forall\ j > i,\ i,j = 1, ..., M \qquad (5.2)$$

$$-\beta_0 + b_{ij} w_{ij} \le y_j - y_i - \beta_0 \mu_{ij} \le -b_{ij} w_{ij} \quad \forall\ j > i,\ i,j = 1, ..., M \qquad (5.3)$$

$$\delta_i + \delta_j - 1 \le z_{ij} + w_{ij} \quad \forall\ j > i,\ i,j = 1, ..., M \qquad (5.4)$$

$$\delta_i \alpha_i / 2 \le x_i \le \delta_i (\alpha_0 - \alpha_i / 2) \quad \forall\ i = 1, ..., M \qquad (5.5)$$

$$\delta_i \beta_i / 2 \le y_i \le \delta_i (\beta_0 - \beta_i / 2) \quad \forall\ i = 1, ..., M \qquad (5.6)$$

$$\delta_i \in \{ 0, 1 \} \quad \forall\ i = 1, ..., M$$
$$\pi_{ij}, \mu_{ij}, z_{ij}, w_{ij} \in \{ 0, 1 \} \quad \forall\ j > i,\ i, j = 1, ..., M \qquad (5.7)$$

The explanation of the formulation is as follows. Constraints (5.2) and (5.3) are the non-overlap conditions for rectangles i and j. (5.2) ensures that if:

$$z_{ij} = 1 \text{ and } \pi_{ij} = 1, \text{ then } x_j \ge x_i + a_{ij}$$
$$z_{ij} = 1 \text{ and } \pi_{ij} = 0, \text{ then } x_i \ge x_j + a_{ij}$$
$$z_{ij} = 0 \text{ and } \pi_{ij} = 1, \text{ then } x_j \ge x_i$$
$$z_{ij} = 0 \text{ and } \pi_{ij} = 0, \text{ then } x_i \ge x_j$$

Similarly for (5.3). (5.4) is an artificial constraint to ensure that if both pieces i and j are chosen to be cut from A_0 ($\delta_i = 1$ and $\delta_j = 1$), then at least one of z_{ij}, w_{ij} must be 1. (5.5) and (5.6) are the constraints regarding non-intersection of the rectangles

cut from A_0 with the edges of the stock rectangle (note that if piece k is not cut from A_0, then (x_k,y_k) is at (0,0)). (5.7) are the integrality constraints.

The above is a mixed integer program (MIP-1) whose size clearly depends on the number of pieces in R (it involves M (2M+1) integer and 2M continuous variables and (5M (M-1)/2 + 4M) constraints). It is well known that, in general, large problems of this type can be solved by the use of tree search procedures. Such procedures depend for their effectiveness on the use of bounds to limit the search (Branch and Bound). In the next section, we discuss how an upper bound on the optimal objective value of the NGC problem can be derived from formulation (5.1) to (5.7).

5.3 A Linear Programming (LP) Relaxation of MIP-1

A problem (P_R) is said to be a relaxation of a maximising problem (P) if the feasible region of (P) is included in the feasible region of (P_R) and the optimal value of (P_R) is larger than or equal to the optimal value of (P). In selecting between alternative types of relaxation for a given problem, there are two main criteria to be considered. On the one hand, it is desirable for the relaxed problem to be significantly easier to solve than the original. On the other hand, one would like (P_R) to yield an optimal solution of (P) or, failing that, the value of (P_R) should be as close as possible to that of (P). Usually, selecting a relaxation involves a trade-off between these two properties; sharper bounds require more time to compute.

The most popular type of relaxation for an integer or mixed integer linear

program is to drop all integrality requirements on the variables. The resulting ordinary linear program is often a good compromise between the two criteria mentioned above.

To obtain an LP relaxation of formulation (5.1) to (5.7), we simply relax the integrality conditions (5.7) to:

$$
\begin{aligned}
&0 \le \delta_i \le 1 \quad \forall\ i = 1, ..., M \\
&0 \le \pi_{ij} \le 1,\ 0 \le \mu_{ij} \le 1 \qquad \forall\ j > i,\ i = 1, ..., M \\
&0 \le z_{ij} \le 1,\ 0 \le w_{ij} \le 1 \qquad \forall\ j > i,\ i = 1, ..., M
\end{aligned} \tag{5.8}
$$

The Linear Program described by equations (5.1) to (5.6) and (5.8) has the same number of variables and constraints as MIP-1. However, we can also include the following area constraints in the above LP (Note that these constraints are redundant for the MIP-1 formulation):

$$\sum_{j=i+1}^{M} \alpha_j \beta_j z_{ij} + \sum_{j=1}^{i-1} \alpha_j \beta_j z_{ji} \le (\alpha_0 - \alpha_i)\, \beta_0\, \delta_i \quad \forall\ i = 1, ..., M \tag{5.9}$$

$$\sum_{j=i+1}^{M} \alpha_j \beta_j w_{ij} + \sum_{j=1}^{i-1} \alpha_j \beta_j w_{ji} \le (\beta_0 - \beta_i)\, \alpha_0\, \delta_i \quad \forall\ i = 1, ..., M \tag{5.10}$$

$$\sum_{j=1}^{M} \alpha_j \beta_j \delta_j \le \alpha_0 \beta_0 \tag{5.11}$$

Having generated the above LP, our task is to solve it and more significantly, to

solve the associated mixed integer program. The idea is to solve the LP and then to use this solution as a basis to obtain a solution to the MIP. Several methods are available to achieve this. Best known among them is Gomory's cutting plane algorithm for solving LP's with integer variables.

In order to solve the LP described by equations (5.1) to (5.6) and (5.8) to (5.11), we used an LP package called XMP, written by *Marsten* [1981] of the University of Arizona. XMP is a hierarchically structured library consisting of 38 subroutines for performing the various functions involved in solving LP's. It is written in FORTRAN and is capable of solving problems of reasonable size that are encountered in a research and development context. The size of the problems that XMP solves on any given computer is limited by the amount of main memory space available.

In solving the above LP, we utilised the following characteristics of XMP:

(i) Because of the size of the problem involved, we had to use condensed data structures. XMP stores the simplex matrix by columns with the zero entries removed.

(ii) We made use of the dual rather than the primal simplex algorithm in view of the possibility of adding extra constraints to the LP solution if necessary. The design of XMP enables it to be incorporated as part of a large program, where we cannot only get information, but can also add new information into the LP and perform more simplex iterations each time a new constraint is added.

Examination of early computational results showed that the optimal solutions to the integer variables of the LP were fractional. Consequently, we implemented the approach described in the next section based on cutting planes.

5.4 Cutting Planes

Historically, the cutting plane method was the first general approach used to solve integer and mixed integer programs. The foundations were laid by *Gomory* in a series of well-known papers through the 1960's ([1958, 1963]). A culmination of these efforts is found in *Trauth and Woosley* [1969] that compared the five leading codes in use at that time, all of which used cutting planes. Several shortcomings in these codes were pointed out, as even small problems had cases where the method failed to converge in a reasonable time.

" Cutting planes " are linear constraints. The general idea is that if the linear program does not produce an integer answer to the integer variables, perhaps by adding valid additional constraints, we can eventually reach an integer solution. To be useful, however, these extra constraints must have certain desirable properties:

(i) Each new cut must properly tighten the previous relaxation, i.e. eliminate the previously found LP solution, and yet still yield a valid relaxation of the original problem. In other words, each new cut must cut off some of the feasible region of the current linear program without also cutting off any feasible integer solutions of the original integer program.

(ii) At most, a finite number of cuts should be necessary in order to find an optimal solution of the given problem or discover that none exists (*Garfinkel and Nemhauser* [1972]). Quite often, the earlier cuts have more impact on the value of the objective function than do later cuts, since there is a "flattening out" of the feasible space around the optimal point as successively thinner slices are cut off.

Almost all of the computational experience with cutting algorithms reported in the literature has been confined to relatively small problems. One of the recognised difficulties is their poor track record on the rate of convergence, or the number of iterations required to reach an optimal solution. Thus, in our attempts to use cutting planes in solving the NGC problem, we developed an algorithm tailored to the structure of the problem and with a facility for obtaining intermediate results if it seemed that the problem was not going to converge quickly enough. This algorithm can provide us with either the optimum mixed integer solution or a much improved bound on the value of the solution than the LP could give with the greatest rate of progress being made during the early iterations. This bound can then be incorporated into a branch and bound algorithm, hopefully having a better performance than if just the LP relaxation were used, but with little extra cost in terms of time if the cutting plane procedure were to be terminated early.

5.4.1 The Cutting Plane Algorithm

The algorithm we developed using the dual simplex method is presented below (for further details on cutting plane methods, see *Gomory* [1963]):

(1) Choose a row of the optimal LP tableau for constructing a cutting plane by picking a basic variable with fractional part in the LP solution but one that is required to have an integer value in the optimum MIP solution.

(2) Calculate the cut by determining its fractional parts in terms of the current basis and produce an expression of an original constraint corresponding to the cut.

(3) Insert the new cut in the problem data structure and make the corresponding slack variable basic.

(4) Resolve the LP using the dual simplex procedure starting from the current basis.

(5) Go to (1) unless one of the following conditions is satisfied, in which case the algorithm terminates:

(i) An optimal solution to the MIP problem has been found.

(ii) The number of cutting iterations has exceeded a constant value.

(iii) The rate of convergence is too low.

Explanation of the algorithm

Step (1) is perhaps an important part of the algorithm in terms of determining the rate of convergence towards the mixed integer solution. There are many ways of choosing cuts. One possible approach is to select the row with the greatest fractional part to the RHS, thus obtaining a reasonably large simplex pivot ratio. Another possibility is to generate a Gomory cut for the integer variable whose fractional part is closest to 0.5. In a cutting plane code written by *Wolfe* [1984] to solve a class of scheduling problems, the variety of choices of cuts were investigated.

The method we employed was to choose to cut first on the δ_i's - the subset of rectangles cut from A_0 - if their values were not integer, and then on the position variables: π_{ij}'s, μ_{ij}'s, z_{ij}'s and w_{ij}'s. We picked a δ_i with the largest fractional part and a position variable whose fractional part was closest to 0.5. This procedure of performing cuts is based on the observation that the values of any of the π, μ, z or w variables are largely dependent on the number of rectangles cut from A_0 (i.e. the δ_i's are the most important decision variables of the problem). Thus, the addition of extra linear constraints on δ_i's is likely to be more noticeable in its effect on the objective function value and the solution as a whole.

At steps (2) and (3), the algorithm calculates the coefficient of the cut chosen at (1) in terms of the current basis and gives the expression of an original constraint corresponding to the cut which is then inserted in the problem data structure in this form. This is because the only data stored in the memory of the XMP package are the original data of the problem and the current inverse of the basis matrix.

5.4.2 Results

The cutting plane algorithm was programmed in FORTRAN and run on a CYBER-855 machine using some of the subroutines of the XMP. In order to evaluate this algorithm computationally, four randomly generated problems of small size were solved (maximum number of pieces in R was 5). Table 5.1 gives the exact details of these four problems, including the sizes and the values of the pieces in R (note that only one piece of each type can be used). Table 5.2 summarises the performance of the algorithm, giving for each problem the value of the LP relaxation, together with the associated computation time in CYBER-855 seconds, and the value of the solution obtained after cutting planes are added, together with the number of cuts and time required. We also give the optimal mixed integer solution for each problem, this being found by using the exact tree search procedure described in chapter 7 which solves the NGC problem. The optimum solutions for all problems are shown diagrammatically in Figure 5.2.

The four test problems of Table 5.1 are of two types: those for which the sum of the areas of pieces in R, i.e.

$$\sum_{i=1}^{M} \alpha_i \beta_i$$

is less than the area of A_0, i.e. $\alpha_0\beta_0$ (note that not all of these pieces can necessarily be cut from A_0), and those for which

$$\sum_{i=1}^{M} \alpha_i \beta_i > \alpha_0 \beta_0.$$

It can be seen from Table 5.1 that the first three test problems are of the first type and problem 4 is of the second type.

From Table 5.2, we notice that for problems 1, 2 and 3, the value of the corresponding LP bound is equal to the sum of the values of all pieces given in R, thus this bound has the largest possible value it can ever take for each problem. As a result, in each one of these LP solutions, the values of all δ_i variables are equal to 1. With this upper bound being so far from the optimum value and all δ_i's - the most significant variable to cut on - having integer values, the cutting planes were of almost no value at all. Indeed, in our attempts to cut only on position variables with fractional parts, the results have shown that the initial Gomory cuts had no effect at all on the objective function of these problems, failing to reduce the gap between the LP bound and the optimum solution of the problem.

In the case of problem 4, the LP relaxation provided us with a bound whose value is 11% away from the optimum solution. Note that in this LP solution, the area constraints (5.9) to (5.11) were active, and some of the δ_i variables had non-integer values in the LP solution. This allowed us to add cuts(always on δ_3, since this variable had the largest fractional part in each solution.) The bound obtained after adding 30 cuts to the LP solution was improved by only 1.5% over the bound derived from the LP relaxation. Figure 5.3 shows the very slow rate of convergence in the value of this bound as the first 30 cuts were added. As a result,

Problem 1: $M = 3$, $(\alpha_0,\beta_0) = (4,4)$, $|\tilde{L}| = 2$, $|\tilde{w}| = 2$, optimal solution = 100

i	α_i	β_i	v_i
1	2	2	40
2	3	2	60
3	2	3	50

Problem 2: $M = 5$, $(\alpha_0,\beta_0) = (6,6)$, $|\tilde{L}| = 4$, $|\tilde{w}| = 6$, optimal solution = 31

i	α_i	β_i	v_i
1	2	2	4
2	3	3	9
3	6	1	6
4	4	2	8
5	2	4	8

Problem 3: $M = 5$, $(\alpha_0,\beta_0) = (10,10)$, $|\tilde{L}| = 7$, $|\tilde{w}| = 7$, optimal solution = 116

i	α_i	β_i	v_i
1	2	2	5
2	5	3	15
3	6	7	52
4	4	7	44
5	2	4	12

Problem 4: $M = 5$, $(\alpha_0,\beta_0) = (20,30)$, $|\tilde{L}| = 2$, $|\tilde{w}| = 3$, optimal solution = 680

i	α_i	β_i	v_i
1	19	10	200
2	18	10	300
3	17	10	180
4	11	10	130
5	10	10	90

Table 5.1 Details of Test Problems 1 to 4 with given Set of Pieces R.

Problem Number	LP Bound		Cutting Planes			Optimal Solution Value
	Value	Time CPsecs	Value	Number of Cuts	Time CPsecs	
1	150	0.9	150	10	1.2	100
2	35	1.2	35	10	1.5	31
3	128	1.5	128	10	1.8	116
4	757.205	0.4	747.023	30	1.3	680

Table 5.2 Performance of Algorithm using Formulation MIP - 1.

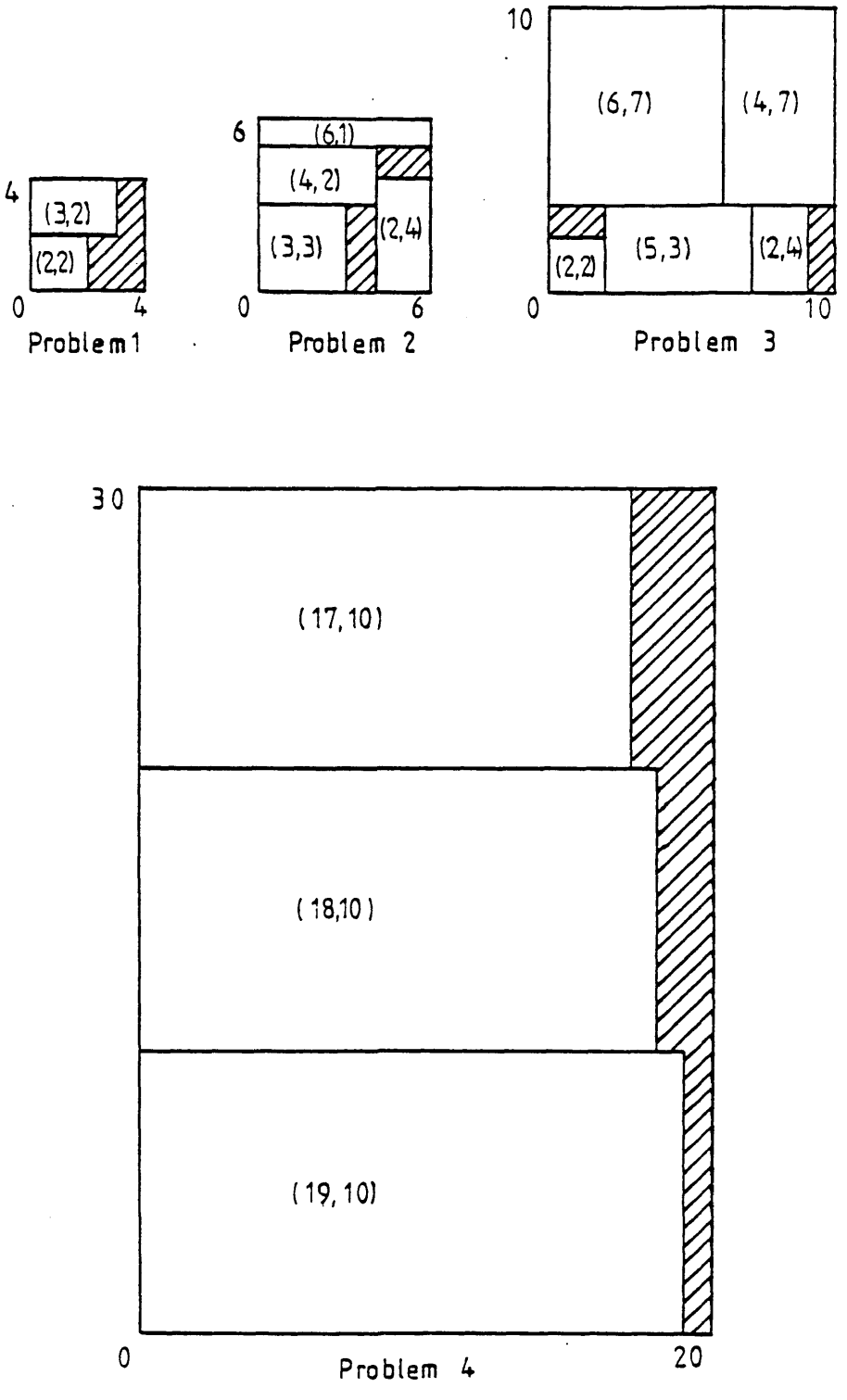


Figure 5.2 Optimal Solutions for Problems 1 to 4 of Table 5.2.

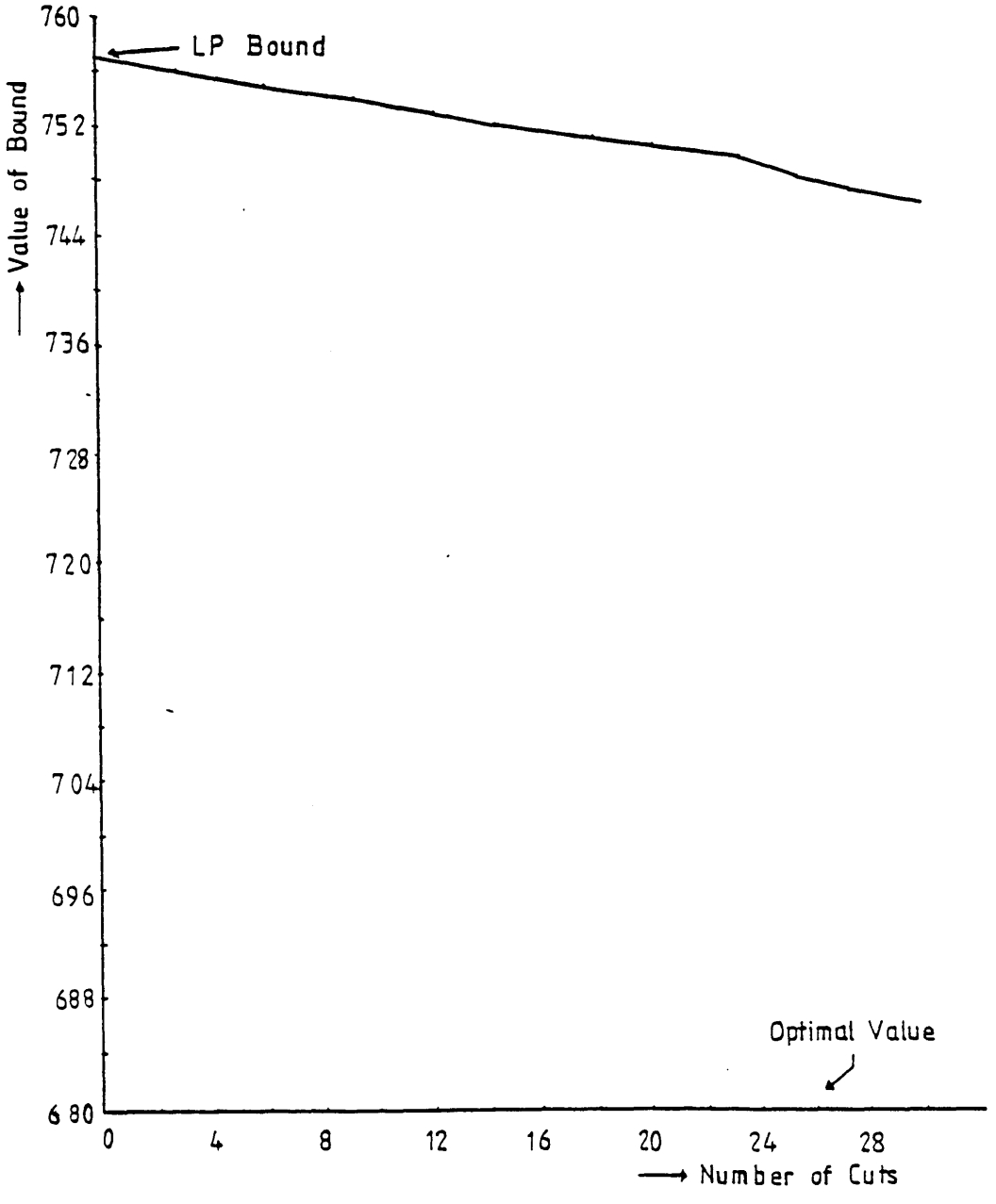


Figure 5.3 Results of the use of Cutting Planes for Problem 4.

there did not seem to be much to be gained from adding more cutting planes as the greatest rate of progress is expected to be made during the early iterations.

5.4.3 Conclusions

From the results of Table 5.2, we conclude the following:

(i) The quality of the bound derived from the LP relaxation of formulation MIP-1 is poor (the bound is about 20% away from the optimum solution on average).

(ii) The performance of the cutting plane algorithm is very poor. It failed to improve the LP bound for the first three problems or it converged very slowly in the case of Problem 4.

In the next section, we present another mixed integer programming formulation of the NGC problem using a different set of variables.

5.5 A Second Mixed Integer Programming Formulation (MIP-2)

The formulation presented here is based on similar ideas as formulation MIP-1. The continuous variables (x_i,y_i) and the 0-1 integer variables δ_i used by MIP-1 are also defined in the same way, so they are not repeated here. The position variables, however, are interpreted differently.

Let $\xi^1_{ij} = 1$ if piece j is cut totally to the right of piece i ($x_j \geq x_i + a_{ij}$)
$= 0$ otherwise

$\xi^2_{ij} = 1$ if piece j is cut totally to the left of piece i ($x_j \leq x_i - a_{ij}$)
$= 0$ otherwise

$\psi^1_{ij} = 1$ if piece j is cut totally above piece i ($y_j \geq y_i + b_{ij}$)
$= 0$ otherwise

$\psi^2_{ij} = 1$ if piece j is cut totally below piece i ($y_j \leq y_i - b_{ij}$)
$= 0$ otherwise

The NGC problem is then formulated as follows:

Problem P_2

$$\text{Max} \quad Z = \sum_{j=1}^{M} \upsilon_j \delta_j \tag{5.12}$$

subject to:

$$x_j - x_i - \alpha_0 \xi^1_{ij} \geq a_{ij} - \alpha_0 \qquad \forall\ j > i,\ i,j = 1, \ldots, M \tag{5.13}$$

$$-x_j + x_i - \alpha_0 \xi^2_{ij} \geq a_{ij} - \alpha_0 \qquad \forall\ j > i,\ i,j = 1, \ldots, M \tag{5.14}$$

$$y_j - y_i - \beta_0 \psi_{ij}^1 \geq b_{ij} - \beta_0 \qquad \forall\ j > i,\ i,j = 1, ..., M \qquad (5.15)$$

$$-y_j + y_i - \beta_0 \psi_{ij}^2 \geq b_{ij} - \beta_0 \qquad \forall\ j > i,\ i,j = 1, ..., M \qquad (5.16)$$

$$\xi_{ij}^1 + \xi_{ij}^2 + \psi_{ij}^1 + \psi_{ij}^2 \geq \delta_i + \delta_j - 1 \qquad \forall\ j > i,\ i,j = 1, ..., M \qquad (5.17)$$

$$\delta_i (\alpha_0 - \alpha_i / 2) \geq x_i \geq \delta_i \alpha_i / 2 \qquad \forall\ i = 1, ..., M \qquad (5.18)$$

$$\delta_i (\beta_0 - \beta_i / 2) \geq y_i \geq \delta_i \beta_i / 2 \qquad \forall\ i = 1, ..., M \qquad (5.19)$$

$$\delta_i \in \{ 0, 1 \} \quad \forall\ i = 1, ..., M$$

$$\xi_{ij}^1, \xi_{ij}^2, \psi_{ij}^1, \psi_{ij}^2 \in \{ 0, 1 \} \qquad \forall\ j > i,\ i,j = 1, ..., M \qquad (5.20)$$

Constraints (5.13) to (5.16) express the non-overlap conditions between any two rectangles i and j cut from A_0. Variables $\xi_{ij}^1, \xi_{ij}^2, \psi_{ij}^1$ and ψ_{ij}^2 can be considered as switches. If they take the value 0, the constraint is off, i.e. redundant. Constraint (5.17) ensures that if both pieces i and j are to be cut from A_0, then at least one of $\xi_{ij}^1, \xi_{ij}^2, \psi_{ij}^1, \psi_{ij}^2$ must be 1. Constraints (5.18) and (5.19) are similar to constraints (5.5) and (5.6) of MIP-1. (5.20) are the integrality constraints.

The above model is a mixed integer program (MIP-2) of similar size to MIP-1. To investigate this formulation computationally, we followed exactly the same approach as we did for formulaton MIP-1, namely, we computed the bound from the LP relaxation of MIP-2 and then applied the cutting plane algorithm of

section 5.4.1 to improve this bound. Using this procedure, we tried to solve the four test problems of Table 5.1. The results obtained are presented in Table 5.3. Clearly, they are almost identical to the ones presented in Table 5.2, leading us to the same conclusions mentioned in section 5.4.3 for formulation MIP-2.

5.6 Two 0-1 Integer Programming Formulations

In this section, we formulate the NGC problem as a zero-one integer programming problem. First, we make some assumptions and define a set of variables which are used by the two formulations of the problem presented below:

(i) the given set of rectangular pieces $\bar{R}$ contains m types of pieces with P_i and Q_i being the minimum and maximum number of pieces of type i that can be cut from A_0

($\bar{R}$ contains M pieces in total, where $M = \sum_{i=1}^{m} Q_i$).

(ii) the cuts on A_0 are performed at integer steps. We define $L = \{ 0, 1, 2, ..., \alpha_0-1 \}$ and $W = \{ 0, 1, 2, ..., \beta_0-1 \}$ to be the sets of points on A_0 where cuts can be made parallel to the y-axis and the x-axis, respectively. Note that we do not regard the top edge and the right hand edge of a piece as cutting points. Similarly, we define $L_i = \{ l \mid l \in L \text{ and } l \le \alpha_0 - \alpha_i \}$ and $W_i = \{ w \mid w \in W \text{ and } w \le \beta_0 - \beta_i \}$ to be the sets of lengths and widths on A_0 achieved by piece i.

Problem Number	LP Bound		Cutting Planes			Optimal Solution Value
	Value	Time CPsecs	Value	Number of Cuts	Time CPsecs	
1	150	0.9	150	10	1.2	100
2	35	1.2	35	10	1.5	31
3	128	1.4	128	10	1.7	116
4	757.2	0.4	747.0	30	1.3	680

Table 5.3 Performance of Algorithm using Formulation MIP - 2.

5.6.1 Formulation IP-1

Define x_{ipq} = 1 if a piece of type i is cut with its bottom left hand corner at (p,q) of A_0, where $0 \le p \le \alpha_0 - \alpha_i$ and $0 \le q \le \beta_0 - \beta_i$ (see Figure 5.4),

= 0 otherwise.

Then, the first formulation is as follows:

Problem P_3

$$\text{Max} \quad Z = \sum_{i=1}^{m} \upsilon_i \sum_{q \in W_i} \sum_{p \in L_i} x_{ipq} \tag{5.21}$$

subject to:

$$\sum_{i=1}^{m} \sum_{\substack{q=s-\beta_i \\ q \in W_i}}^{s-1} \sum_{\substack{p=r-\alpha_i \\ p \in L_i}}^{r-1} x_{ipq} \le 1 \qquad \forall \{ r \mid (r-1) \in L \}, \{ s \mid (s-1) \in W \} \tag{5.22}$$

$$P_i \le \sum_{q \in W_i} \sum_{p \in L_i} x_{ipq} \le Q_i \qquad \forall \; i = 1, \ldots, m \tag{5.23}$$

$$x_{ipq} \in \{ 0, 1 \} \qquad \forall \; i = 1, \ldots, m, \; p \in L_i, \; q \in W_i \tag{5.24}$$

Constraint (5.22) expresses the non-overlap conditions between any

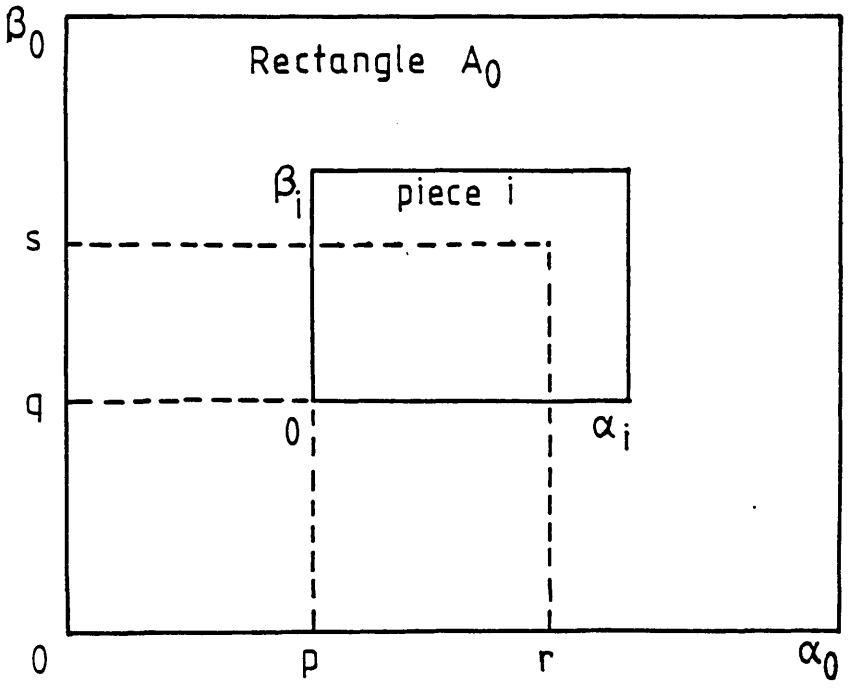


Figure 5.4 Piece i is cut with its bottom left-hand corner at position (p, q).

rectangles in $\bar{R}$ cut from A_0 by ensuring that any point (r, s) on A_0 is cut out by at most one rectangle. Constraint (5.23) expresses the fact that the number of pieces of type i cut out from A_0 lies within the required range. Constraints (5.24) are the integrality constraints.

The above is a large 0-1 integer programming formulation of the NGC problem (IP-1). It is similar to one given by *Beasley* [1985b] who developed a tree search procedure based upon a Lagrangean relaxation of his formulation to solve moderate sized non-guillotine problems.

Formulation (IP-1) involves approximately

$$\sum_{i=1}^{m} |L_i|\,|W_i| \quad \text{variables and} \quad (|L|\,|W| + m) \text{ constraints.}$$

However, the size of this program can be reduced by using the idea of "normal" cutting patterns (*Christofides and Whitlock* [1979]). As it has been explained in chapter 2, normal patterns rely on the observation that in a given cutting pattern, any cut piece can be moved to the left and/or down until its left hand edge and its bottom edge both touch other cut pieces (or the edges of A_0). Without any loss of optimality, we can replace sets L and W by:

$$\tilde{L} = \{\, l \mid l = \sum_{i=1}^{m} \eta_i \alpha_i,\ 0 \le l \le [\, \alpha_0 - \min (\alpha_i \mid i = 1, ..., m) \,],$$

$$0 \le \eta_i \le Q_i,\ \eta_i \text{ integer},\ i = 1, ..., m \,\}$$

$$\tilde{W} = \{\, w \mid w = \sum_{i=1}^{m} t_i \beta_i,\ 0 \le w \le [\, \beta_0 - \min (\beta_i \mid i = 1, ..., m) \,],$$

$$0 \le t_i \le Q_i,\ t_i \text{ integer},\ i = 1, ..., m \,\}.$$

Similarly, sets $\tilde{L}_i$ and $\tilde{W}_i$ defined by $\tilde{L}_i = \{ l \mid l \in \tilde{L}$ and $l \leq \alpha_0 - \alpha_i \}$ and $\tilde{W}_i = \{w \mid w \in \tilde{W}$ and $w \leq \beta_0 - \beta_i \}$ can be used instead of sets L_i and W_i for all $i = 1, ..., m$. It is easy to show that constraint (5.22) need be only applied for the restricted sets $\tilde{L}$, $\tilde{W}$, $\tilde{L}_i$ and $\tilde{W}_i$ ($i = 1, ..., m$), leading to a significant reduction in the size of formulation (5.21) to (5.24).

5.6.2 Formulation IP-2

Using the same definition of x_{ipq}'s given in section 5.6.1, we formulate the second 0-1 integer programming formulation of the NGC problem as follows:

Problem P_4

$$\text{Max} \quad Z = \sum_{i=1}^{m} v_i \sum_{q \in \tilde{W}_i} \sum_{p \in \tilde{L}_i} x_{ipq} \qquad (5.25)$$

subject to

$$\sum_{\substack{q = s - \beta_i + 1 \\ q \in \tilde{W}_i}}^{s + \beta_j - 1} \sum_{\substack{p = r - \alpha_i + 1 \\ p \in \tilde{L}_i}}^{r + \alpha_j - 1} x_{ipq} \leq 1 - x_{jrs} \qquad \forall j \neq i, \quad i, j = 1, ..., m \quad r \in \tilde{L}_j, s \in \tilde{W}_j \qquad (5.26)$$

$$P_i \leq \sum_{q \in \tilde{W}_i} \sum_{p \in \tilde{L}_i} x_{ipq} \leq Q_i \qquad \forall\ i = 1, \ldots, m \qquad (5.27)$$

$$x_{ipq} \in \{0, 1\} \qquad \forall\ i = 1, \ldots, m,\ p \in \tilde{L}_i,\ q \in \tilde{W}_i \qquad (5.28)$$

Constraint (5.26) expresses the non-overlap conditions between any two rectangles i and j in $\bar{R}$ cut from A_0. If $x_{jrs} = 1$, then the corresponding constraint is active, otherwise it is redundant. Constraint (5.27) ensures that the number of pieces of type i cut out from A_0 lies within the required range. Constraints (5.28) are the integrality constraints.

The restriction of the cuts on A_0 to have the property of normal patterns can also be applied to the above formulation. Thus, IP-2 involves the same number of variables as formulation IP-1 given by :

$$\sum_{i=1}^{m} |\tilde{L}_i|\,|\tilde{W}_i|.$$

The number of constraints, however, is larger and is given by $(m\,(m-1)\,|\tilde{L}|\,|\tilde{W}| + m)$.

5.7 A second Set of 0 - 1 Integer Programming Formulations

The three formulations presented below hold when in the set of rectangular pieces R we have at most one piece of type i available to cut from A_0. However, if

at most Q_i pieces of type i (i = 1, ..., m) are required to be cut, then these pieces are included in R with different labels but the same dimensions (the total number of pieces in R is given by $M = \sum_{i=1}^{m} Q_i$).

Normal cutting patterns are also used to reduce the size of the problem. We define the following variables used by the three formulations:

Let x_{ip} = 1 if piece i is cut with its bottom left hand corner at position x = p, where $p \in \tilde{L}_i$,

= 0 otherwise.

y_{iq} = 1 if piece i is cut with its bottom left hand corner at position y = q, where $q \in \tilde{W}_i$,

= 0 otherwise.

5.7.1 Formulation IP-3

Formulation IP-3 also involves the following variables:

Let z_{rs} = 1 if point (r, s) on A_0 is free so that any piece in R can be cut with its bottom left hand corner at (r,s), where $r \in L$ and $s \in W$,

= 0 otherwise.

Problem P_5

$$\text{Max} \quad Z = \sum_{i=1}^{M} v_i \sum_{p \in \tilde{L}_i} x_{ip} \qquad (5.29)$$

subject to:

$$\sum_{s=q}^{q+\beta_i-1} \sum_{r=p}^{p+\alpha_i-1} z_{rs} \le (2 - x_{ip} - y_{iq})\,\alpha_i \beta_i$$

$$\forall\ i = 1, \ldots, M,\ p \in \tilde{L}_i,\ q \in \tilde{W}_i \qquad (5.30)$$

$$\sum_{p \in L_i} x_{ip} \le 1 \qquad \forall\ i = 1, \ldots, M \qquad (5.31)$$

$$\sum_{p \in L_i} x_{ip} = \sum_{q \in W_i} y_{iq} \qquad \forall\ i = 1, \ldots, M \qquad (5.32)$$

$$\begin{aligned} x_{ip} &\in \{0, 1\} \quad \forall\ i = 1, \ldots, M,\ p \in \tilde{L}_i \\ y_{iq} &\in \{0, 1\} \quad \forall\ i = 1, \ldots, M,\ q \in \tilde{W}_i \\ z_{rs} &\in \{0, 1\} \quad \forall\ r \in L,\ s \in W \end{aligned} \qquad (5.33)$$

(5.30) ensures that any point (p, q) on A_0 is cut out by at most one piece. (5.31) and (5.32) express the fact that each piece is cut at most once from A_0. (5.33) are the integrality conditions.

The above is a complete formulation involving approximately $(2M + \sum_{i=1}^{M} |\tilde{L}_i|\,|\tilde{W}_i|)$ constraints and $(\alpha_0 \beta_0 + \sum_{i=1}^{M} (|\tilde{L}_i| + |\tilde{W}_i|))$ integer variables. Clearly, the size of IP-3 depends on the total number of pieces (M) given in R, whereas the sizes of IP-1 and IP-2 depend on the number of types of pieces (m) given in $\bar{R}$. This difference in size between formulations IP-1 and IP-3 can be illustrated by the use of an example.

Suppose we are given a test problem in which 20 types of pieces (one of each type, i.e. $Q_i = 1$ for all i = 1, ..., 20) can be cut from a stock rectangle A_0 of size (50,50). Formulating this problem as IP-1, involves about 50,000 variables whereas formulating it as IP-3, we would use only 4,500 variables. It is clear that unless Q_i's are large (the size of IP-1 is independent of Q_i's), formulation IP-3 is much smaller than IP-1.

IP-1 might be more suitable for formulating problems in which the dimensions of the pieces to be cut have small absolute values. Note that the expression giving the number of variables for IP-3 involves the extra term $\alpha_0 \beta_0$, as it is shown in Table 5.4 which compares the sizes of all 0-1 integer programming formulations of the NGC problem presented in this chapter.

5.7.2 Formulation IP-4

This formulation uses z_{rs} variables defined in the following way:

Let z_{rs} = 1 if point (r, s) on A_0 is covered by any piece i in R which is cut with its bottom left hand corner at location (p, q), where $r - \alpha_i \leq p < r$ and $s - \beta_i \leq q < s$ ($r \in L$ and $s \in W$)

= 0 otherwise.

Then, IP-4 is given by:

Formulation	Number of Variables	Number of Constraints
IP-1	$\sum_{i=1}^{m} \lvert\tilde{L}_i\rvert \lvert\tilde{\omega}_i\rvert$	$\lvert\tilde{L}\rvert \lvert\tilde{\omega}\rvert + m$
IP-2	$\sum_{i=1}^{m} \lvert\tilde{L}_i\rvert \lvert\tilde{\omega}_i\rvert$	$m\ (m-1)\ \lvert\tilde{L}\rvert \lvert\tilde{\omega}\rvert + m$
IP-3	$\alpha_0\ \beta_0 + \sum_{i=1}^{M} (\lvert\tilde{L}_i\rvert + \lvert\tilde{\omega}_i\rvert)$	$2M + \sum_{i=1}^{M} \lvert\tilde{L}_i\rvert \lvert\tilde{\omega}_i\rvert$
IP-4	$\alpha_0\ \beta_0 + \sum_{i=1}^{M} (\lvert\tilde{L}_i\rvert + \lvert\tilde{\omega}_i\rvert)$	$M\ (\alpha_0\ \beta_0 + 2)$
IP-5	$\lvert\tilde{L}\rvert \lvert\tilde{\omega}\rvert + \sum_{i=1}^{M} (\lvert\tilde{L}_i\rvert + \lvert\tilde{\omega}_i\rvert)$	$\lvert\tilde{L}\rvert + \lvert\tilde{\omega}\rvert + 2M + \sum_{i=1}^{M} \lvert\tilde{L}_i\rvert \lvert\tilde{\omega}_i\rvert$

Table 5.4 Sizes of the five IP Formulations for the NGC Problem presented in sections 5.6 and 5.7.

Problem P_6

$$\text{Max} \quad Z = \sum_{i=1}^{M} v_i \sum_{p \in \tilde{L}_i} x_{ip} \tag{5.34}$$

subject to

$$z_{rs} \geq \sum_{\substack{p = r - \alpha_i \\ p \in \tilde{L}_i}}^{r-1} x_{ip} + \sum_{\substack{q = s - \beta_i \\ q \in \tilde{W}_i}}^{s-1} y_{iq} - 1 \quad \forall\, i = 1, \ldots, M, \; r \in L, \; s \in W \tag{5.35}$$

$$\sum_{p \in \tilde{L}_i} x_{ip} \leq 1 \quad \forall\, i = 1, \ldots, M \tag{5.36}$$

$$\sum_{p \in \tilde{L}_i} x_{ip} = \sum_{q \in \tilde{W}_i} y_{iq} \quad \forall\, i = 1, \ldots, M \tag{5.37}$$

$$\begin{aligned} &x_{ip} \in \{0, 1\} \quad \forall\, i = 1, \ldots, M, \; p \in \tilde{L}_i \\ &y_{iq} \in \{0, 1\} \quad \forall\, i = 1, \ldots, M, \; q \in \tilde{W}_i \\ &z_{rs} \in \{0, 1\} \quad \forall\, r \in L, \; s \in W \end{aligned} \tag{5.38}$$

Constraint (5.35) expresses the fact that any point (r,s) on A_0 can be cut out by at most one piece. If a piece i is cut with its bottom left hand corner at a location (p,q) on A_0 such that (r,s) is cut out by piece i, then the corresponding constraint is active; otherwise it is redundant.

Constraints (5.36) and (5.37) ensure that each piece is cut at most once from A_0.

Constraints (5.38) express the integrality conditions.

The above formulation involves the same number of variables as IP-3 but a larger number of constraints given by M ($\alpha_0 \beta_0 + 2$). It is clear that applying the restriction of the cuts on A_0 to have the property of normal patterns does not reduce the size of IP-4. This means that formulating even a small-sized NGC problem as IP-4, we have a problem with a very large number of constraints to solve.

5.7.3 Formulation IP-5

Formulation IP-5 uses z_{rs} variables in a different way. These are now defined as follows:

Let z_{rs} = 1 if any piece i is cut with its bottom left hand corner at (r,s), where $r \in \tilde{L}$ and $s \in \tilde{W}$,

= 0 otherwise.

Then the problem is expressed as follows:

Problem P_7

$$\text{Max} \quad Z = \sum_{i=1}^{M} v_i \sum_{p \in \tilde{L}_i} x_{ip} \qquad (5.39)$$

subject to:

$$\sum_{\substack{s=q-\min_{j\neq i}\beta_j+1 \\ s\,\varepsilon\,\tilde{W}}}^{q+\beta_i+1} \sum_{\substack{r=p-\min_{j\neq i}\alpha_j+1 \\ r\,\varepsilon\,\tilde{L}}}^{p+\alpha_i-1} z_{rs} - z_{pq} \leq (2 - x_{ip} - y_{iq}) \min \{ M,$$

$$[(\alpha_i + \min_{j\neq i}\alpha_j - 1)(\beta_i + \min_{j\neq i}\beta_j - 1) - 1] \}$$

$$\forall\ i = 1, \ldots, M,\ p\,\varepsilon\,\tilde{L}_i,\ q\,\varepsilon\,\tilde{W}_i \qquad (5.40)$$

$$\sum_{p\,\varepsilon\,\tilde{L}_i} x_{ip} \leq 1 \qquad \forall\ i = 1, \ldots, M \qquad (5.41)$$

$$\sum_{p\,\varepsilon\,\tilde{L}_i} x_{ip} = \sum_{q\,\varepsilon\,\tilde{W}_i} y_{iq} \qquad \forall\ i = 1, \ldots, M \qquad (5.42)$$

$$\sum_{i\,\varepsilon\,\{i \mid r\,\varepsilon\,\tilde{L}_i\}} x_{ir} = \sum_{s\,\varepsilon\,\tilde{W}} z_{rs} \qquad \forall\ r\,\varepsilon\,\tilde{L} \qquad (5.43)$$

$$\sum_{i\,\varepsilon\,\{i \mid s\,\varepsilon\,\tilde{W}_i\}} y_{is} = \sum_{r\,\varepsilon\,\tilde{L}} z_{rs} \qquad \forall\ s\,\varepsilon\,\tilde{W} \qquad (5.44)$$

$$\begin{aligned} &x_{ip}\,\varepsilon\,\{0, 1\} \quad \forall\ i = 1, \ldots, M,\ p\,\varepsilon\,\tilde{L}_i \\ &y_{iq}\,\varepsilon\,\{0, 1\} \quad \forall\ i = 1, \ldots, M,\ q\,\varepsilon\,\tilde{W}_i \\ &z_{rs}\,\varepsilon\,\{0, 1\} \quad \forall\ r\,\varepsilon\,\tilde{L},\ s\,\varepsilon\,\tilde{W} \end{aligned} \qquad (5.45)$$

(5.40) are the non-overlap conditions. (5.43) and (5.44) express the number of rectangles cut with their bottom left hand corners at a particular $x = r$ ($r \in \tilde{L}$) and $y = s$ ($s \in \tilde{W}$), respectively. (5.45) are the integrality conditions.

The above formulation (IP-4) involves $(|\tilde{L}|\,|\tilde{W}| + \sum_{i=1}^{M} (|\tilde{L}_i| + |\tilde{W}_i|))$ variables and $(|\tilde{L}| + |\tilde{W}| + 2M + \sum_{i=1}^{M} |\tilde{L}_i|\,|\tilde{W}_i|)$ constraints.

5.8 Computational Aspects of Bound Calculations for the 0 - 1 IP Formulations

In Sections 5.6 and 5.7, five different 0-1 integer programming formulations for the NGC problem were presented, namely, IP-1, IP-2, IP-3, IP-4 and IP-5. In order to obtain upper bounds on the optimum solution of the problem, we relax these formulations by dropping the integrality conditions. Let UB1 and UB2 be the upper bounds derived from the linear relaxation of formulations IP-1 and IP-2, respectively. In the LP relaxations of IP-3, IP-4 and IP-5, we add the following constraints in order to obtain tighter bounds (note that these constraints are redundant for IP-3, IP-4 and IP-5).

Additional Constraints to Formulation IP-3

$$\sum_{i=1}^{M} \beta_i \sum_{\substack{p=r-\alpha_i+1 \\ p \in \tilde{L}_i}}^{r} x_{ip} + \sum_{s \in W} z_{rs} = \beta_0 \quad \forall\, r \in L \qquad (5.46)$$

$$\sum_{i=1}^{M} \alpha_i \sum_{\substack{q=s-\beta_i+1 \\ q \in \tilde{W}_i}}^{s} y_{iq} + \sum_{r \in L} z_{rs} = \alpha_0 \qquad \forall\, s \in W \qquad (5.47)$$

Additional Constraints to Formulation IP-4

$$\sum_{i=1}^{M} \beta_i \sum_{\substack{p=r-\alpha_i+1 \\ p \in \tilde{L}_i}}^{r} x_{ip} = \sum_{s \in W} z_{rs} \qquad \forall\, r \in L \qquad (5.48)$$

$$\sum_{i=1}^{M} \alpha_i \sum_{\substack{q=s-\beta_i+1 \\ q \in \tilde{W}_i}}^{s} y_{iq} = \sum_{r \in L} z_{rs} \qquad \forall\, s \in W \qquad (5.49)$$

Additional Constraints to Formulaton IP-5

$$\sum_{i=1}^{M} \beta_i \sum_{\substack{p=r-\alpha_i+1 \\ p \in \tilde{L}_i}}^{r} x_{ip} \le \beta_0 \qquad \forall\, r \in \tilde{L} \qquad (5.50)$$

$$\sum_{i=1}^{M} \alpha_i \sum_{\substack{q=s-\beta_i+1 \\ q \in \tilde{W}_i}}^{s} y_{iq} \le \alpha_0 \qquad \forall\, s \in \tilde{W} \qquad (5.51)$$

Constraints (5.46), (5.48) and (5.50) ensure that if k pieces overlap such that $\sum_{i=1}^{k} \beta_i > \beta_0$, they cannot all be cut with their bottom left hand corners at the same length. Similarly, (5.47), (5.49) and (5.51) ensure that if l pieces overlap such that

$\sum_{i=1}^{\downarrow} \alpha_i > \alpha_0$, they cannot all be cut with their bottom left hand corners at the same width.

Let UB3, UB4 and UB5 be the upper bounds derived from the linear relaxation of IP-3, IP-4 and IP-5, respectively, including the above additional constraints. Bounds UB1 to UB5 were obtained by using the XMP package described in Section 5.3. Because of the large number of variables and constraints involved in the formulations, we were forced by the memory limitations of the XMP package to solve only test problems of small size. Thus, eight problems involving up to seven pieces in R and three problems involving up to seven types of pieces in $\bar{R}$ have been randomly generated and run on a CYBER-855 machine. Details of these problems shown in Tables 5.1, 5.5 and 5.6 include, for each problem, the size of A_0, the sizes (α_i,β_i) and values (υ_i) of the given pieces, the maximum number of pieces of each type (Q_i) in $\bar{R}$ and the size of sets of normal cuts ($|\tilde{L}|$ and $|\tilde{W}|$). The value of the integer optimal solution is also given for each problem, this being found by the exact tree search procedure described in Chapter 7 which solves the NGC problem. The optimal solutions for the first four problems are shown in Figure 5.2 and for the other seven problems in Figure 5.5.

Problems 1 to 4 have been described in Section 5.4.2 and bounds derived from formulation MIP-1 have been presented for these problems in Table 5.2. In problems 5 to 8, which have been randomly generated, at most one piece of each type in R is available to be cut from A_0. Problems B1, B4 and B5 have been taken from *Beasley* [1985b]; they correspond to problems 1, 4 and 5 in the table of computational results presented in the paper by Beasley. These problems are also randomly generated with the data being drawn from uniform distributions and Q_i have integer values between 1 and 3.

Problem 5: $M = 4$, $(\alpha_0,\beta_0) = (7,9)$, $|\tilde{L}| = 3$, $|\tilde{W}| = 9$, optimal solution = 54

i	α_i	β_i	v_i
1	7	4	28
2	6	3	18
3	5	5	25
4	4	1	8

Problem 6: $M = 5$, $(\alpha_0,\beta_0) = (8,6)$, $|\tilde{L}| = 6$, $|\tilde{W}| = 4$, optimal solution = 85

i	α_i	β_i	v_i
I	3	3	15
2	5	2	20
3	7	3	40
4	2	6	30
5	4	4	35

Problem 7: $M = 7$, $(\alpha_0,\beta_0) = (10,10)$, $|\tilde{L}| = 9$, $|\tilde{W}| = 10$, optimal solution = 198

i	α_i	β_i	v_i
1	1	10	28
2	5	3	40
3	9	3	63
4	6	1	13
5	3	8	31
6	4	1	10
7	7	3	44

Problem 8: $M = 7$, $(\alpha_0,\beta_0) = (15,10)$, $|\tilde{L}| = 7$, $|\tilde{W}| = 10$, optimal solution = 262

i	α_i	β_i	v_i
1	10	3	34
2	9	3	48
3	12	2	72
4	11	3	91
5	12	3	37
6	11	1	15
7	2	10	36

Table 5.5 Details of Test Problems 5 to 8 with given set of pieces **R**.

Problem B1: $m = 5$, $(\alpha_0,\beta_0) = (10,10)$, $|\tilde{L}| = 8$, $|\tilde{W}| = 6$, optimal solution = 164

i	α_i	β_i	v_i	Q_i
1	8	2	40	2
2	2	10	43	2
3	3	7	35	2
4	10	2	27	1
5	5	4	23	3

Problem B4: $m = 5$, $(\alpha_0,\beta_0) = (15,10)$, $|\tilde{L}| = 3$, $|\tilde{W}| = 10$, optimal solution = 268

i	α_i	β_i	v_i	Q_i
1	8	3	71	1
2	15	2	61	2
3	15	1	14	1
4	7	3	27	1
5	15	2	34	2

Problem B5: $m = 7$, $(\alpha_0,\beta_0) = (15,10)$, $|\tilde{L}| = 7$, $|\tilde{W}| = 10$, optimal solution = 358

i	α_i	β_i	v_i	Q_i
1	12	2	72	3
2	11	3	91	1
3	2	10	36	1
4	9	3	48	1
5	11	1	15	3
6	10	3	34	2
7	12	3	37	3

Table 5.6 Details of Test Problems B1, B4 and B5 with given set of pieces $\bar{R}$.

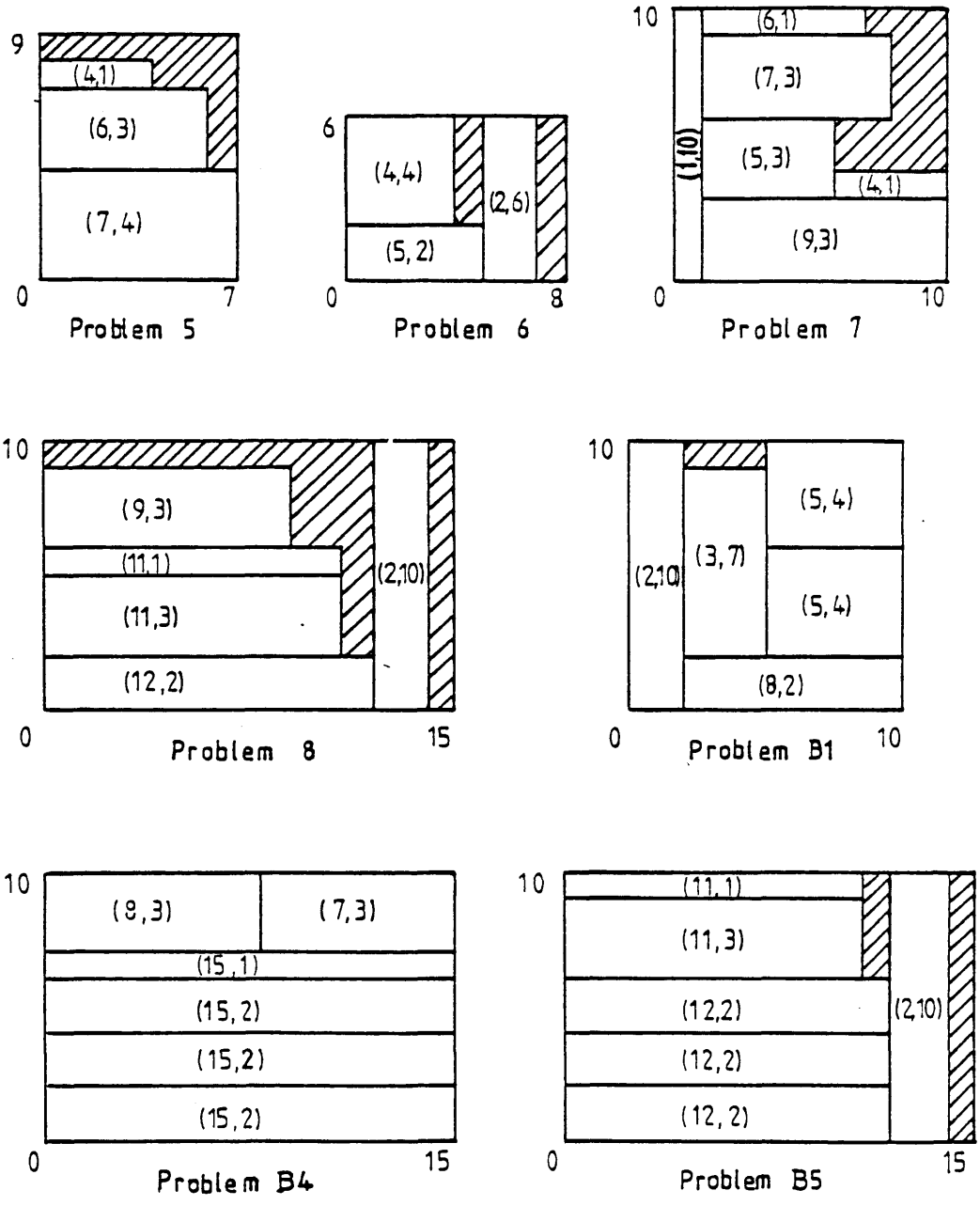


Figure 5.5 Optimal Solutions for Problems 5 to 8 of Table 5.5 and Problems B1, B4 and B5 of Table 5.6.

Tables 5.7 to 5.11 describe the performance of the five bounds UB1 to UB5 obtained for the above eleven test problems, respectively. For each problem, we give the value of the corresponding bound, together with the associated computation time in CYBER-855 CP seconds. We also give the number of variables and constraints (including the redundant constraints for formulations IP-3, IP-4 and IP-5 given in this section) involved in all formulations for each problem. (Because of the large number of non-zero entries per column in the matrix representation of the simplex tableau - greater than 100 - the XMP package could not solve problems 7, B1 and B5 using formulation IP-2 and problems 4, 7, 8, B1, B4 and B5 using formulation IP-4.)

In order to compare the quality of the various bounds, the following performance ratio is used:

$$r = \frac{\text{Upper Bound - Optimal Value}}{\text{Optimal Value}} \ (\%)$$

5.8.1 Computational Comparison

From the tables of computational results, we can see that the value of bound UB2 differs significantly from the values of the other bounds for problems 3, 4, 5 and 8 (the quality of UB2 for these problems is poorer by 8%, 11%, 15% and 5.8%, respectively). In particular, UB2 has the same poor performance as the bounds derived from the two mixed integer formulations of Sections 5.2 and 5.5 for problems 2, 3, and 4 (Table 5.2). Furthermore, because of the large number of constraints and non-zero entries in the simplex tableau involved in formulation IP-2, we are limited to solve NGC problems of very small size. Hence, UB2 has been

Problem Number	Value	Time in CP secs	$\pi\%$	Number of Variables	Number of Constraints
1	125	0.06	25%	8	7
2	35	0.3	13%	60	29
3	119	0.9	2%	143	54
4	680	0.1	-	18	11
5	59	0.1	9%	23	12
6	100	0.4	17%	44	29
7	218	8.4	10%	210	97
8	274	0.9	4%	109	77
B1	205	0.8	25%	60	53
B4	271	0.4	1%	68	35
B5	359	2.0	0.3%	233	84

Table 5.7 Performance of Bound UB1.

Problem Number	Value	Time in CP secs	$\pi\%$	Number of Variables	Number of Constraints
1	113	0.1	13%	8	20
2	35	2.6	13%	60	246
3	128	4.4	10%	143	578
4	757	0.3	11%	18	78
5	67	0.4	24%	23	74
6	101	1.4	18%	44	182
8	301	5.3	9.8%	109	662
B1	271	1.9	1%	68	278

Table 5.8 Performance of Bound UB2.

Problem Number	Value	Time in CP secs	$r\%$	Number of Variables	Number of Constraints
1	131	0.1	31%	26	22
2	35	0.7	13%	73	82
3	119	1.9	2%	152	173
4	680	0.5	-	621	78
5	59	0.7	9%	90	47
6	101	0.9	18%	80	68
7	218	3.7	10%	184	244
8	274	3.4	4%	221	148
B1	206	3.7	25%	186	174
B4	271	2.4	1%	222	125
B5	359	6.9	0.3%	297	286

Table 5.9 Performance of Bound UB3.

Problem Number	Value	Time in CP secs	$r\%$	Number of Variables	Number of Constraints
1	129	0.5	29%	26	62
2	35	3.2	13%	73	202
3	119	19.9	2%	152	530
5	59	6.1	9%	90	276
6	101	6.6	18%	80	264

Table 5.10 Performance of Bound UB4.

Problem Number	Value	Time in CP secs	π%	Number of Variables	Number of Constraints
1	134	0.1	34%	14	23
2	35	0.9	13%	61	91
3	119	1.8	2%	101	182
4	680	0.2	-	27	39
5	59	0.4	9%	35	50
6	101	0.8	18%	56	75
7	218	4.4	10%	174	263
8	274	2.0	4%	141	158
B1	206	2.7	25%	134	183
B4	271	1.8	1%	102	127
B5	359	3.5	0.3%	217	296

Table 5.11 Performance of Bound UB5.

excluded from further investigation.

The value of bounds UB3 and UB4 has basically the same performance ratio for problems 1, 2, 3, 5 and 6. However, the results for this set of test problems show that UB3 requires a considerably lower computational cost since the size of the problems formulated as IP-3 is much smaller than the size of the same problems being formulated as IP-4 (the number of variables involved in both formulations for a particular problem is the same). Hence, UB4 has been excluded from further investigation.

The three bounds UB1, UB3 and UB5 are basically compared with respect to their performance ratio r and their computational time. There is not any clear indication for better quality of bound, since the corresponding duality gaps computed for all eleven problems are basically the same (note that for problem 4, all three bounds find the optimal solution). However, the results for this set of test problems show that UB1 is computationally less expensive with the exception of problem 7. In this case, the use of normal patterns had very poor effect on reducing the problem being formulated as IP-1 (for problems in which $|\tilde{L}|$ and $|\tilde{W}|$ are relatively large compared to α_0 and β_0, respectively, UB3 is expected to have better performance than UB1). On the other hand, for problem 4, UB3 requires a large number of variables (621) compared to only 18 and 27 variables required by UB1 and UB5, respectively. This happens because of the reduction in the number of variables resulting from the small size of the normal sets obtained in problem 4 for formulation IP-3 ($|\tilde{L}|$ and $|\tilde{W}|$ have values of 2 and 3 compared to α_0 and β_0, being 20 and 30, respectively).

5.8.2 Conclusions

Based on the computational results presented in Tables 5.7 to 5.11, it is not very obvious which is the best bound for the NGC problem. Overall formulation IP-3 seems to be slightly better and hence we choose this formulation in order to develop a method to solve optimally NGC problems. In the next chapter, we present bounds derived from a Lagrangian relaxation of IP-3 with computational results obtained for medium-sized problems. These bounds can then be embedded in a tree search procedure that solves NGC problems exactly (Chapter 7).

CHAPTER 6

A LAGRANGEAN RELAXATION BOUND FOR THE NGC PROBLEM IMPROVED BY SUBGRADIENT OPTIMISATION AND PROBLEM REDUCTION TESTS

6.1 Introduction

In this chapter we consider the NGC problem of section 5.1, which was defined as follows: we are given a rectangular sheet of stock material A_0 having dimensions (α_0,β_0) and a number (m) of types of pieces in R having dimensions (α_i,β_i) and values v_i for all i=1, ... , m. Then using no more than Q_i pieces of each type i in R, we require to find an orthogonal non-guillotine cutting pattern of pieces on A_0 that has the highest possible total value. No pair of cut rectangles are allowed to overlap and no cut rectangle may overlap the edges of A_0. Each piece in R is considered to have a fixed orientation (Total number of pieces in R is given by

$$M = \sum_{i=1}^{m} Q_i \ ).$$

In our attempt to solve the above problem optimally, we use a 0-1 integer programming formulation for this problem, presented in Chapter 5, namely formulation IP-3. In this chapter, we develop a Lagrangean Relaxation of this formulation to provide us with an upper bound on the problem. Subgradient optimisation is used to optimise the bound derived from the Lagrangean Relaxation. Problem reduction tests derived from both the original problem and from the Lagrangean Relaxation are given. Using the subgradient method together with the reduction tests we present a general procedure used to obtain the best bound and the greatest reduction of the problem. If the opimal solution is not found using this procedure, then the bound obtained can be incorporated into a tree-search procedure that solves the problem optimally (Chapter 7).

Computational experience with the general procedure is presented in the last section of this chapter.

6.2 Lagrangean Relaxation

In the last decade, Lagrangean Relaxation has grown from a successful but largely theoretical concept to a tool that is the backbone of a number of large-scale applications. It is based on the observation that many difficult integer programming problems can be modelled as a relatively easy problem complicated by a set of side constraints. Replacing the complicating constraints with a penalty term in the objective function that involves the amount of violation of the constraints, we create a Lagrangean problem that is easy to solve and whose optimal value is an upper bound (for a maximisation problem) on the optimal value of the original problem. The Lagrangean problem can thus be used in place of a linear programming

relaxation to provide bounds in a branch-and-bound algorithm.

We first explain the Lagrangean Relaxation concept in general terms and then apply it to the NGC problem.

Let (P) be the following integer linear problem:

$$Z = \underset{x}{\text{maximise}}\ cx \qquad (6.1)$$

subject to

$$A x \leq b \qquad (6.2)$$

$$D x \leq d \qquad (6.3)$$

$$x \geq 0 \ \text{ and integer} \qquad (6.4)$$

where b,c and d are vectors and A and D are matrices. We assume that the constraints of (P) have been partitioned into the two sets (6.2) and (6.3) so that (P) is relatively easy to solve if the constraint set (6.2) is removed. The Lagrangean relaxation of (P) relative to (6.2) and with a non-negative multiplier vector λ ($\lambda \geq 0$) is defined as problem (PR_λ) given by:

$$Z_D(\lambda) = \underset{x}{\text{Maximise}}\ cx + \lambda (b - Ax) \qquad (6.5)$$

subject to

$$Dx \leq d \qquad (6.6)$$

$$x \geq 0 \qquad (6.7)$$

It is clear that the optimal value of problem (PR_λ) for λ fixed at a non-negative value is an upper bound on Z because we have merely added a non-negative term to the objective function (6.1) and dropped some constraints. *Geoffrion* [1974] has shown that the potential usefulness of relaxation (PR_λ) is largely determined by the gap between the value of its optimal solution and that of (P). This is known as the "duality gap" and it gives a criterion by which to measure the "quality" of a particular choice of λ. In particular, the program

$$(D) \qquad \underset{\lambda \geq 0}{\text{minimise}} \quad \upsilon(PR_\lambda)$$

gives the best possible upper bound to the original problem (P) where $\upsilon(PR_\lambda)$ is the optimal value of problem (PR_λ).

Relaxing a problem (P) using the Lagrangean approach offers a number of important advantages:

(i) With careful choice of which constraints to relax, the relaxation can make the problem significantly easier to solve. Typically, one will construct several alternative relaxations and evaluate them, both empirically and analytically based on the quality of bounds obtained. Lagrangean Relaxation, having the ability to exploit special problem structure, often is the only hope for coping with large scale real problems.

(ii) Clearly, the effectiveness of a bound is the most important parameter that determines the efficiency of a branch-and-bound algorithm. Choice of good

Lagrangean multipliers can provide us with the tightest bound on the problem, so that results can be significantly superior to LP based branch-and-bound.

(iii) Practical experience with the method has indicated that the resulting duality gap is often very small, so that a very good bound can be obtained for use in a tree-search procedure.

The use of Lagrangean Relaxation has led to dramatically improved algorithms for a number of important problems, namely the Travelling Salesman Problem (*Held and Karp* [1971]), scheduling problems (*Fisher* [1973]), location problems and set covering problems. A recent survey of Lagrangean Relaxation and its applications has been produced by *Shapiro* [1979].

6.2.1 A Lagrangean Relaxation for the NGC Problem

In section 5.7.1 we have given the 0-1 integer programming formulation (IP-3) of the NGC problem as presented below:

Problem P

$$\text{Max} \quad Z = \sum_{i=1}^{m} \upsilon_i \sum_{p \in \tilde{L}_i} x_{ip} \tag{6.8}$$

subject to

$$\sum_{s=q}^{q+\beta_i-1} \sum_{r=p}^{p+\alpha_i-1} z_{rs} \le (2 - x_{ip} - y_{iq})\,\alpha_i \beta_i \quad \forall\ i = 1, \ldots, M,\ p \in \tilde{L}_i,\ q \in \tilde{W}_i \tag{6.9}$$

$$\sum_{p \in \tilde{L}_i} x_{ip} \leq 1 \quad \forall\ i = 1, \ldots, M \tag{6.10}$$

$$\sum_{q \in \tilde{W}_i} y_{iq} \leq 1 \quad \forall\ i = 1, \ldots, M \tag{6.11}$$

$$\begin{aligned} &x_{ip} \in \{0, 1\} \quad \forall\ i = 1, \ldots, M,\ p \in \tilde{L}_i \\ &y_{iq} \in \{0, 1\} \quad \forall\ i = 1, \ldots, M,\ q \in \tilde{W}_i \\ &z_{rs} \in \{0, 1\} \quad \forall\ r \in L,\ s \in W \end{aligned} \tag{6.12}$$

The following set of constraints which were redundant in the original formulation but helped in generating tighter bounds in its Linear Programming Relaxation (section 5.8) can be added:

$$\sum_{i=1}^{M} \beta_i \sum_{\substack{p=r-\alpha_i+1 \\ p \in \tilde{L}_i}}^{r} x_{ip} + \sum_{s \in W} z_{rs} = \beta_0 \quad \forall\ r \in L \tag{6.13}$$

$$\sum_{i=1}^{M} \alpha_i \sum_{\substack{q=s-\beta_i+1 \\ q \in \tilde{W}_i}}^{s} y_{iq} + \sum_{r \in L} z_{rs} = \alpha_0 \quad \forall\ s \in W \tag{6.14}$$

We relax the above program by introducing Lagrange multipliers u_{ipq} (≥ 0) $\forall\ i = 1, \ldots, M, p \in \tilde{L}_i$ and $q \in \tilde{W}_i$ relative to the non-overlapping constraints (6.9), multipliers e_r $\forall\ r \in L$ for constraints (6.13) and multipliers f_s $\forall\ s \in W$ for constraints (6.14) to give us the following Lagrangean problem (LR):

$$Z_D(u,e,f) = \max \{ \sum_{i=1}^{M} \sum_{p \in \tilde{L}_i} \upsilon_i x_{ip} + \sum_{i=1}^{M} \sum_{p \in \tilde{L}_i} \sum_{q \in \tilde{W}_i} u_{ipq}$$

$$(2\alpha_i\beta_i - \alpha_i\beta_i x_{ip} - \alpha_i\beta_i y_{iq} - \sum_{s=q}^{q+\beta_i-1} \sum_{r=p}^{p+\alpha_i-1} z_{rs}) +$$

$$\sum_{r \in L} e_r (\beta_0 - \sum_{i=1}^{M} \beta_i \sum_{\substack{p=r-\alpha_i+1 \\ p \in \tilde{L}_i}}^{r} x_{ip} - \sum_{s \in W} z_{rs}) +$$

$$\sum_{s \in W} f_s (\alpha_0 - \sum_{i=1}^{M} \alpha_i \sum_{\substack{q=s-\beta_i+1 \\ q \in \tilde{W}_i}}^{s} y_{iq} - \sum_{r \in L} z_{rs}) \}$$

subject to (6.10), (6.11) and (6.12).

After rearrangement we get:

Problem LR

$$Z_D(u,e,f) = \max [\sum_{i=1}^{M} \sum_{p \in \tilde{L}_i} G_{ip} x_{ip} - \sum_{i=1}^{M} \sum_{q \in \tilde{W}_i} H_{iq} y_{iq} -$$

$$\sum_{r \in L} \sum_{s \in W} I_{rs} z_{rs} + 2 \sum_{i=1}^{M} \alpha_i\beta_i \sum_{p \in \tilde{L}_i} \sum_{q \in \tilde{W}_i} u_{ipq}$$

$$+ \beta_0 \sum_{r \varepsilon L} e_r + \alpha_0 \sum_{s \varepsilon W} f_s \;] \qquad (6.15)$$

where

$$G_{ip} = \upsilon_i - \alpha_i\beta_i \sum_{q \varepsilon \tilde{W}_i} u_{ipq} - \beta_i \sum_{r=p}^{p+\alpha_i-1} e_r \;,$$

$$H_{iq} = \alpha_i\beta_i \sum_{p \varepsilon \tilde{L}_i} u_{ipq} + \alpha_i \sum_{s=q}^{q+\beta_i-1} f_s \qquad \text{and}$$

$$I_{rs} = e_r + f_s + \sum_{i=1}^{M} \sum_{\substack{p=r-\alpha_i+1 \\ p \varepsilon \tilde{L}_i}}^{r} \sum_{\substack{q=s-\beta_i+1 \\ q \varepsilon \tilde{W}_i}}^{s} u_{ipq} \qquad (6.16)$$

subject to

$$\sum_{p \varepsilon \tilde{L}_i} x_{ip} \leq 1 \qquad \forall \; i = 1, \ldots, M \qquad (6.17)$$

$$\sum_{p \varepsilon \tilde{L}_i} x_{ip} = \sum_{q \varepsilon \tilde{W}_i} y_{iq} \qquad \forall \; i = 1, \ldots, M \qquad (6.18)$$

$$x_{ip} \in \{0, 1\} \ \forall \ i = 1, \ldots, M, \ p \in \tilde{L}_i$$
$$y_{iq} \in \{0, 1\} \ \forall \ i = 1, \ldots, M, \ q \in \tilde{W}_i \qquad (6.19)$$
$$z_{rs} \in \{0, 1\} \ \forall \ r \in L, \ s \in W$$

The optimal value of problem (LR) for any set of non-negative u_{ipq} multipliers is an upper bound on the optimal value of the original NGC problem (P). The Lagrangean problem (LR) can be solved optimally using the following observation. Considering a piece i in R, we can extract the terms in the above program associated with it to form a corresponding subproblem. Thus we can obtain m separate subproblems of the following type:

Subproblem LR_p

Find variables x_{ip} and y_{iq} that satisfy

$$Z_p = \max \left[\sum_{p \in \tilde{L}_i} G_{ip} x_{ip} - \sum_{q \in \tilde{W}_i} H_{iq} y_{iq} \right]$$

subject to

$$\sum_{p \in \tilde{L}_i} x_{ip} \leq 1$$

$$\sum_{q \in \tilde{W}_i} y_{iq} \leq 1$$

$$x_{ip}, \ y_{ip} \in \{0, 1\} \quad \forall \ p \in \tilde{L}_i, \ q \in \tilde{W}_i$$

Subproblem (LR_p) picks the best position for cutting out piece i from A_0 removing from the original NGC problem any restriction that the pieces cut should not overlap.

It is easy to solve subproblem (LR_p) if the Lagrangean multipliers u_{ipq} are fixed at some nonnegative values. All $(G_{ip} - H_{iq})$ values are computed for all $p \in \tilde{L}_i$ and $q \in \tilde{W}_i$. Note that if any of these values is not positive, we can set the corresponding pair of variables to 0. Otherwise, we set a pair of variables (x_{ip}, y_{iq}) with the largest $(G_{ip} - H_{iq})$ value to 1 provided that this value is non-negative ($[G_{ir} - H_{is}]$ say) and set the remaining x_{ip} and y_{iq} ($p \neq r, q \neq s$) variables to zero. Let (X_{ip}, Y_{iq}) represent the optimal values of (x_{ip}, y_{iq}) in the solution of the Lagrangean problem (LR).

Solution values to variables z_{rs} can be obtained by inspection from problem (LR). Its solution sets the z_{rs} with positive objective coefficients I_{rs} to 1 and the remaining z_{rs} to zero. Let (Z_{rs}) represent the optimal values of (z_{rs}) in the solution of problem (LR). Then the optimal value of the Lagrangean objective function $Z_D(u, e, f)$ is an upper bound Z_{UB} on the optimal objective value of the original NGC problem. This bound is given by:

$$Z_{UB} = \sum_{i=1}^{M} \sum_{p \in \tilde{L}_i} G_{ip}X_{ip} - \sum_{i=1}^{M} \sum_{q \in \tilde{W}_i} H_{iq}Y_{iq} - \sum_{r \in L} \sum_{s \in W} I_{rs}Z_{rs}$$

$$+ 2\sum_{i=1}^{M} \alpha_i\beta_i \sum_{p \in \tilde{L}_i} \sum_{q \in \tilde{W}_i} u_{ipq} + \beta_0 \sum_{r \in L} e_r + \alpha_0 \sum_{s \in W} f_s \quad (6.20)$$

Now we consider the relative sharpness of this bound. Ideally, the

Lagrangean multipliers should solve the following dual problem:

$$Z_D = \min\ Z_D(u, e, f)\,, \quad u \geq 0$$

Let Z_{LP} denote the upper bound obtained from the solution of the LP relaxation of problem (P) (Z_{LP} is another notation for bound UB3 of chapter 5). An analytic result given by *Geoffrion* [1974] allows us to compare Z_D with Z_{LP}. The result states that in general $Z_D \leq Z_{LP}$.

In the lagrangean problem (LR) we observe that the optimal values of the variables will be integer whether we require it or not. Thus $Z_D(u,e,f)$ is not decreased by removing the integrality restrictions on the variables from the constraints of problem (LR). Geoffrion calls this the " integrality property ". The implication of the property is fairly immediate and can be given by the following:

THEOREM 1: Let the LP relaxation be feasible and let the Lagrangean relaxation have the integrality property. Then the maximum value of the Lagrangean relaxation is equal to the value of the LP relaxation.

Since (LR) possesses the integrality property, by Theorem 1 its maximum value equals the value of the LP relaxation of (P), i.e $Z_D = Z_{LP}$. The best choice of approximating the Lagrangean multiplier values u, e and f for (LR) is then the optimal values of the dual multipliers from the LP relaxation of (P). However, a nice feature of the Lagrangean method is that it is not necessary to solve the dual problem which is generally a large LP even for small size NGC problem (chapter 5). The method used to optimise $Z_D(u, e, f)$, called subgradient optimisation, is more powerful than methods available for solving the large scale LP relaxation of (P).

Our purpose in section 6.4, is to show how subgradient optimisation used in conjunction with Lagrangean relaxation is successfully used to provide us with bounds for small to medium sized NGC problems which can then be embedded in a tree-search procedure used to solve these problems. First, some reduction tests derived from both problems (P) and (LR) are presented in the next section.

6.3 Problem reduction

As the optimal value Z_{UB} to the Lagrangean problem (LR) is always (for any set of u_{ipq} (≥ 0), e_r and f_s) an upper bound to the optimal solution of the original problem (P), a reduction in the problem can be achieved, if the enforcement of a set of conditions in (LR) results in an optimal value Z_{UB} below some (previously determined) lower bound Z_{LB} to (P). We then know that this set of conditions can never be satisfied at the optimal solution to the original problem e.g if forcing Q_i pieces of type i in R to be cut from A_0 takes the corresponding solution value below Z_{LB} then we know that Q_i pieces of this type can never be produced by the optimal solution, leading us to consider at most Q_i-1 pieces of type i as candidates for cutting.

Let P_i and Q_i be the minimum and maximum number of pieces of type i that can be cut from A_0 ($0 \leq P_i \leq Q_i$ for all $i = 1, ..., m$).

The first three reduction tests presented below are derived from the original problem (P).

(1) Overlapping pieces

We can update Q_i, the maximum number of pieces of any type i in R that can be cut out of A_0 (i = 1, ..., m), in the following way:

Consider any two types of pieces in R, (i* and j* say), that overlap such that $\alpha_{i*} + \alpha_{j*} > \alpha_0$; i.e a piece of type i* and a piece of type j* cannot both be cut from A_0 with their bottom left-hand corners at the same width. Let $b_{j*} = \lceil P_{j*} / \lfloor \alpha_0 / \alpha_{j*} \rfloor \rceil \beta_{j*}$ denote the amount of β_0 that is taken up with cutting out P_{j*} pieces of type j*, where $\lfloor ** \rfloor$ denotes the largest integer less than or equal to ** and $\lceil ** \rceil$ denotes the smallest integer greater than or equal to **. Then the maximum number of pieces of type i* that can be cut out using the remainder of A_0 is given by

$$\overline{Q}_{i*} = \lfloor \alpha_0 / \alpha_{i*} \rfloor \lfloor (\beta_0 - b_{j*}) / \beta_{i*} \rfloor \qquad (6.21)$$

If $\overline{Q}_{i*} < Q_{i*}$ then Q_{i*} can be updated and set equal to $\overline{Q}_{i*}$. Applying this reduction test to the example shown in figure 6.1, we find that at most two pieces of type i* can be cut out of A_0 (note that three pieces of the same type are given in R), since at least one piece of type j* has to be cut (types i* and j* overlap).

Using a similar argument, the maximum number of pieces of type j* can be updated by

$$Q_{j*} = \min (Q_{j*} , \lfloor \alpha_0 / \alpha_{j*} \rfloor \lfloor \beta_0 - \lceil P_{i*} / \lfloor \alpha_0 / \alpha_{i*} \rfloor \rceil \beta_{i*}) / \beta_{j*} \rfloor) \qquad (6.22)$$

Equation (6.22) is derived by considering how much of A_0 is taken up with cutting out P_{i*} pieces of type i* and using the remainder of A_0 to cut out pieces of type j*.

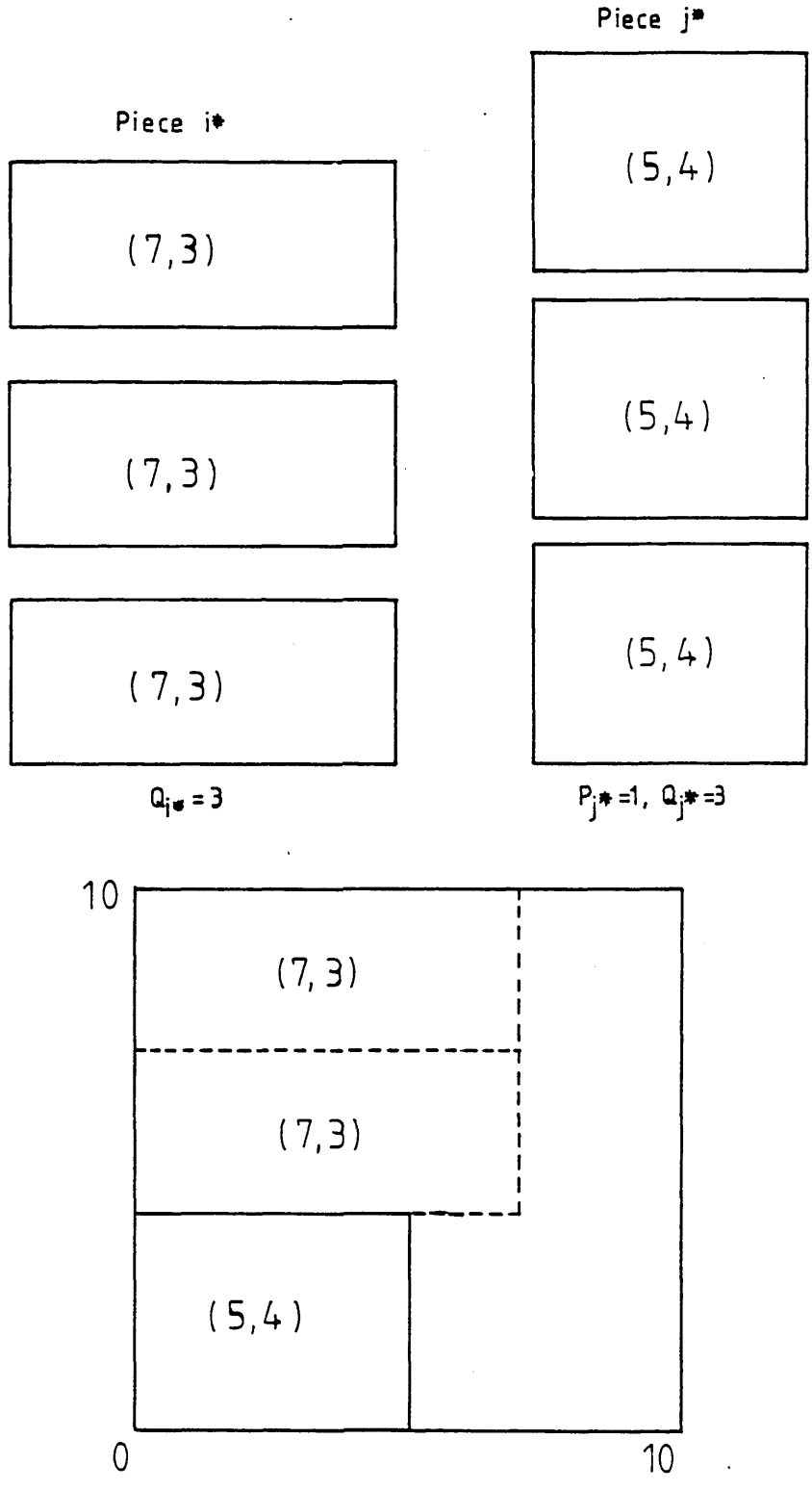


Figure 6.1 Reduction Test 1 : Overlapping Pieces.

Expressions like (6.21) and (6.22) also hold for types of pieces for which $\beta_{i*} + \beta_{j*} > \beta_0$.

(2) Free Area

Q_i can also be updated for any type in R (i = 1, ..., m) as follows:

Consider a certain type in R (i* say), with minimum number of pieces required to be cut from A_0 given by P_{i*}. Let

$$A_{i*} = \sum_{\substack{j=1 \\ j \neq i*}}^{m} P_j \alpha_j \beta_j$$

represent the area of A_0 that is cut out by P_j pieces of each type j ($j \neq i*$) in R. Then the maximum number of pieces of type i* that can be cut out using the remainder of A_0 is given by

$$\overline{Q_{i*}} = P_{i*} + \lfloor (\alpha_0 \beta_0 - A_{i*}) / (\alpha_{i*} \beta_{i*}) \rfloor .$$

If $\overline{Q_{i*}} < Q_{i*}$ then Q_{i*} can be updated and set equal to $\overline{Q_{i*}}$.

(3) Knapsack area program

Define $t_{ij} = 1$ if the jth piece of type i is cut from A_0 (j = 1, ..., Q_i)

$= 0$ otherwise

Then the best set of rectangles in R that are cut from A_0 limited by its area is given by:

Problem KNAP1

$$Z_{KN} = \max \sum_{i=1}^{m} \sum_{j=1}^{Q_i} t_{ij} v_i \tag{6.23}$$

subject to

$$P_i \leq \sum_{j=1}^{Q_i} t_{ij} \qquad \forall \ i = 1, ..., m \tag{6.24}$$

$$\sum_{i=1}^{m} \sum_{j=1}^{Q_i} \alpha_i \beta_i t_{ij} \leq \alpha_0 \beta_0 \tag{6.25}$$

$$t_{ij} \in \{ 0, 1 \} \quad \forall \ i = 1, ..., m, \ j = 1, ..., Q_i \tag{6.26}$$

This program can be viewed as a knapsack problem which is easily solved by the standard dynamic programming algorithm for the Knapsack problem (*Sahni and Horowitz* [1979]). Its solution value is clearly an upper bound on the optimal solution to problem (P). A reduction test is derived from this program which is used to update both P_i and Q_i of type i for all $i = 1, ..., m$. We can estimate a penalty value that results from forcing exactly r_{i*} pieces of a particular type i* to be cut from A_0 where $P_{i*} \leq r_{i*} \leq Q_{i*}$. The method outlined below is used to obtain penalty values for a particular type i* in R.

Let $Z_{KN}(r_{i*})$ be the upper bound obtained from (KNAP1) by forcing r_{i*} pieces of type i* to be in the solution of problem (KNAP1) and Z_{LB} be some (previously determined) lower bound to problem (P). Let

$$A_{i*} = \sum_{\substack{j=1 \\ j\neq i*}}^{m} P_j \alpha_j \beta_j \qquad \text{and} \qquad V_{i*} = \sum_{\substack{j=1 \\ j\neq i*}}^{m} P_j \upsilon_j$$

represent the area of A_0 that is taken up with cutting out P_j pieces of each type j ($j\neq i*$) in R and the associated total value of the pieces cut, respectively. The procedure is then described as follows:

(a) Set $r_{i*} = P_{i*}$

(b) Solve the following Knapsack problem:

$$Z_B = \max \sum_{\substack{j=1 \\ j\neq i*}}^{m} \sum_{k=1}^{Q_j-P_j} \upsilon_j t_{jk}$$

subject to

$$\sum_{\substack{j=1 \\ j\neq i*}}^{m} \sum_{k=1}^{Q_j-P_j} \alpha_j \beta_j t_{jk} \leq \alpha_0 \beta_0 - \alpha_{i*} \beta_{i*} r_{i*} - A_{i*}$$

$$t_{jk} \in \{0, 1\} \quad \forall \; j = 1, ..., m \; (j \neq i*), \quad k = 1, ..., Q_j$$

Then $Z_{KN}(r_{i*}) = Z_B + V_{i*} + v_{i*} r_{i*}$

(c) Set $r_{i*} = r_{i*} + 1$. If $r_{i*} \leq Q_{i*}$ go to step (b); otherwise continue.

(d) By investigating all values of r_{i*} where $P_{i*} \leq r_{i*} \leq Q_{i*}$, we obtain a set of upper bounds $Z_{KN}(r_{i*})$ which are then compared with Z_{LB} in order to update P_{i*} and Q_{i*} accordingly.

Updating of P_{i*}:

If we find that cutting r_{i*} ($P_{i*} \leq r_{i*} \leq Q_{i*}$) pieces of type i* from A_0 produces an upper bound below Z_{LB}, then P_{i*} can be set equal to $(r_{i*} + 1)$ in the optimal solution of problem (P); otherwise P_{i*} is set equal to r_{i*}. Thus P_{i*} can be updated by

$$P_{i*} = \max(P_{i*}, 1 + \max\{ r_{i*} \mid P_{i*} \leq r_{i*} \leq Q_{i*}, Z_{KN}(r_{i*}) \leq Z_{LB} \}).$$

(Note that if $P_{i*} > Q_{i*}$ then the current solution to (P) is infeasible.)

Updating of Q_{i*}:

If the upper bound $Z_{KN}(Q_{i*})$ obtained when forcing Q_{i*} pieces of type i* to be cut from A_0 is less than Z_{LB}, then we can reduce Q_{i*} by one. Thus Q_{i*} can be updated by

$$Q_{i*} = \min(Q_{i*}, \min\{ r_{i*} \mid r_{i*} = Q_{i*}, (Q_{i*}-1), \ldots, (Q_{i*}-P_{i*}) \text{ and } Z_{KN}(r_{i*}) \leq Z_{LB} \} - 1)$$

(Note that if $Q_{i*} < P_{i*}$ then the current solution to (P) is infeasible.)

The following four tests are derived from the Lagrangean problem (LR) by

estimating the decrease in the upper bound Z_{UB} that may result from forcing a set of variables to be in the solution of (LR).

(4) Free Value

A reduction test similar to Test (2), can be used to update Q_i for any type i in R in the following way:

Consider a certain type in R (i* say) with minimum number of pieces required to be cut from A_0 given by P_{i*}. Let

$$V_{i*} = \sum_{\substack{j=1 \\ j \neq i*}}^{m} P_j \upsilon_j$$

represent the total value associated with P_j pieces of each type j ($j \neq i*$) in R being cut out of A_0. Then the maximum number of pieces of type i* that can be considered for cutting is given by

$$\overline{Q}_{i*} = P_{i*} + \lfloor (Z_{UB} - V_{i*}) / \upsilon_{i*} \rfloor$$

where Z_{UB} is an upper bound on the optimal solution to problem (P). If $\overline{Q}_{i*} < Q_{i*}$ then Q_{i*} can be updated and set equal to $\overline{Q}_{i*}$.

(5) Penalties on the Number of Cut Pieces

It is clear from the structure of the Lagrangean program (LR), that we can

calculate an upper bound on the solution obtained with exactly r_i pieces of type i cut from A_0 (i=1, ..., m). Thus, a reduction test is derived from (LR) in order to update both P_i and Q_i of type i for all i = 1, ..., m using the method outlined below (Note that the Knapsack area program of Test 3 is also used to update P_i and Q_i).

Consider a particular type in R (i* say). Let Z_{LR} (r_{i*}) denote the penalty value obtained from (LR) by forcing r_{i*} pieces of type i* to be in the solution of problem (LR) and Z_{LB} denote (some previously determined) lower bound to problem (P). Let S_{i*j*} represent the largest ($G_{j*p} - H_{j*q}$) value associated with the j* th piece of type i* ($p \in \bar{L}_{i*}$ and $q \in \bar{W}_{i*}$) in the Lagrangean solution; then the upper bound Z_{LR} (r_{i*}) is given by

$$Z_{UB} - \sum_{j*=1}^{Q_{i*}} \left(\sum_{p \in \bar{L}_{i*}} G_{j*p} X_{j*p} - \sum_{q \in \bar{W}_{i*}} H_{j*q} Y_{j*q} \right) + \sum_{j*=1}^{r_{i*}} S_{i*j*} \quad (6.27)$$

where Z_{UB} is an upper bound on the optimal solution to problem (P). The second term, in the above expression, represents the cost of removing all pieces of type i* obtained by the Lagrangean solution and the third term, the cost of forcing r_{i*} pieces of this type to be in the solution.

By investigating all values of r_{i*} where $P_{i*} \le r_{i*} \le Q_{i*}$ and using (6.27), we obtain a set of upper bounds Z_{LR} (r_{i*}). These values are then compared with Z_{LB} so that P_{i*} and Q_{i*} can be updated as follows:

Updating of P_{i*}:

$P_{i*} = \max (P_{i*}, 1 + \max \{ r_{i*} \mid P_{i*} \le r_{i*} \le Q_{i*}$ and $Z_{LR} (r_{i*}) \le Z_{LB} \})$

e.g if the upper bound obtained when $r_{i*} = P_{i*}$ is less than Z_{LB}, then we can increase P_{i*} by one.

Updating of Q_{i*}:

$Q_{i*} = \min (Q_{i*}, \min \{ r_{i*} | \; r_{i*} = Q_{i*} , (Q_{i*}-1), ..., (Q_{i*}-P_{i*}) \text{ and } Z_{LR} (r_{i*}) \leq Z_{LB} \} - 1)$

e.g if the upper bound obtained when $r_{i*} = Q_{i*}$ is less than Z_{LB} then we reduce Q_{i*} by one.

(Note that if updating of P_{i*} and Q_{i*} results in $P_{i*} > Q_{i*}$ then the current solution to problem (P) is infeasible).

(6) Penalties on Cut Positions

This reduction test is derived from the Lagrangean program (LR) and is used to calculate penalties for setting variables x_{jp} and y_{jq} to one (or zero) in the Lagrangean solution (i.e forcing piece j to be cut with its bottom left-hand corner at position (p,q) on A_0 (or not)) for all pieces in R (j = 1, ..., M).

Consider a particular piece in R (j* say). First, we distinguish two separate cases for cutting this piece with its bottom left-hand corner at a specified location on A_0.

(a) Setting $x_{j*p} = 1$ (where the corresponding Lagrangean value is zero i.e $X_{j*p} = 0$); $p \varepsilon \tilde{L}_{j*}$. The penalty in setting x_{j*p} to one in the solution of problem (LR) , denoted by $Z_1(j*, p)$, is given by:

$$Z_{UB} - (G_{j^*r} - H_{j^*s})_{X_{j^*r}=1,\, Y_{j^*s}=1,\, r \in \tilde{L}_{j^*},\, s \in \tilde{W}_{j^*}}^{+}$$

$$\max_{q \in \tilde{W}_{j^*},\, q \neq s} (G_{j^*p} - H_{j^*q}) \quad \text{if } \sum_{k \in \tilde{L}_{j^*}} X_{j^*k} = 1 \text{ and } \sum_{l \in \tilde{W}_{j^*}} Y_{j^*l} = 1 \qquad (6.28)$$

$$Z_{UB} + \max_{q \in \tilde{W}_{j^*}} (G_{j^*p} - H_{j^*q}) \quad \text{otherwise} \qquad (6.29)$$

Equations (6.28) and (6.29) preserve the condition that piece j* may be included in the Lagrangean solution at most once. Location (r,s) on A_0 [equation (6.28)] is picked by the Lagrangean solution as the best possible position such that piece j* can be cut with its bottom left-hand corner at (r,s). Then, the second term of equation (6.28) represents the cost of replacing location (r,s) in the Lagrangean solution by another location (p,q) on A_0 which is determined by the maximisation term in both equations. This term represents the cost of cutting piece j* with its bottom left-hand corner at some location (p,q) in the modified solution of problem (LR) (i.e with (r,s) replaced by (p,q)). If the penalty value $Z_1(j^*, p)$ is less than Z_{LB} , a lower bound on problem (P) corresponding to a feasible solution, then we cannot set x_{j^*p} to one in the optimal solution and so x_{j^*p} can be deleted from problem (LR).

(b) Setting $y_{j^*q} = 1$ (where the corresponding Lagrangean solution $Y_{j^*q} = 0$); $q \in \tilde{W}_{j^*}$. The penalty in setting y_{j^*q} to one in the solution of problem (LR), denoted by $Z_1(j^*, q)$, is given by:

$$Z_{UB} - (G_{j^*r} - H_{j^*s})_{X_{j^*r}=1,\, Y_{j^*s}=1,\, r \in \tilde{L}_{j^*},\, s \in \tilde{W}_{j^*}}^{+}$$

$$\max_{p \in \tilde{L}_{j^*},\, p \neq r} (G_{j^*p} - H_{j^*q}) \quad \text{if} \sum_{k \in \tilde{L}_{j^*}} X_{j^*k} = 1 \quad \text{and} \sum_{l \in \tilde{W}_{j^*}} Y_{j^*l} = 1 \qquad (6.30)$$

$$Z_{UB} + \max_{p \in \tilde{L}_{j^*}} (G_{j^*p} - H_{j^*q}) \quad \text{otherwise} \qquad (6.31)$$

These penalties are derived in the same way as equations (6.28) and (6.29). If $Z_1(j^*,q)$ is less than Z_{LB}, then y_{j^*q} can be deleted from problem (LR).

It is clear from the Lagrangean problem that a penalty value can also be calculated for not cutting piece j* with its bottom left-hand corner at a specified location ((p, q) say) on A_0. Let Z_0 (j*, p, q) denote the penalty in setting both x_{j^*p} and y_{j^*q} to zero (where $X_{j^*p} = 1$ and $Y_{j^*q} = 1$); $p \in \tilde{L}_{j^*}$ and $q \in \tilde{W}_{j^*}$. Let (r*,s*) represent the best location (on A_0) in the Lagrangean solution picked for cutting piece j* (if piece j* has to be in the Lagrangean solution). Then the cost of removing location (p,q) from the solution is represented by ($G_{j^*p} - H_{j^*q}$) and the cost of bringing location (r,s) into the solution by ($G_{j^*r^*} - H_{j^*s^*}$) where

$$G_{j^*r^*} - H_{j^*s^*} = \max_{r \in \tilde{L}_{j^*},\, s \in \tilde{W}_{j^*},\, r \neq p,\, s \neq q} \{ G_{j^*r} - H_{j^*s} \}.$$

Thus Z_0 (j*, p, q) is given by:

$$Z_{UB} - (G_{j^*p} - H_{j^*q}) + (G_{j^*r^*} - H_{j^*s^*})$$

$$\text{if} \sum_{k \in \tilde{L}_{j^*}} X_{j^*k} = 1 \quad \text{and} \quad \sum_{l \in \tilde{W}_{j^*}} Y_{j^*l} = 1 \qquad (6.32)$$

$$Z_{UB} - (G_{j^*p} - H_{j^*q}) + \max \{ (G_{j^*r^*} - H_{j^*s^*}), 0 \} \quad \text{otherwise} \tag{6.33}$$

Equations (6.32) and (6.33) preserve the condition that piece j* may be included in the Lagrangean solution at most once. If $Z_0 (j^*, p, q)$ is less than Z_{LB}, then we can cut piece j* with its bottom left-hand corner at location (p,q) in the optimal solution and so both x_{j^*p} and y_{j^*q} can be set to one in problem (LR).

(7) Knapsack problem based on the Lagrangean solution

A knapsack area program can be written similar to problem (KNAP1) [equations (6.23) to (6.26)], with the difference that the objective function coefficients are values obtained from the Lagrangean problem (LR). Thus we define

$t_{ij} = 1$ if the jth piece of a type i in R is cut from A_0 ($j = 1, ..., Q_i$)

$= 0$ otherwise.

Also we define S_{ij} to be the largest ($G_{jp} - H_{jq}$) value associated with the jth piece of type i ($p \in \tilde{L}_i$ and $q \in \tilde{W}_i$) in the Lagrangean solution; then the best set of rectangles in R that are cut from A_0 limited by its area is given by:

Problem KNAP2

$$Z_{KN} = \max \sum_{i=1}^{m} \sum_{j=1}^{Q_i} S_{ij} t_{ij}$$

subject to

$$P_i \leq \sum_{j=1}^{Q_i} t_{ij} \quad \forall \; i = 1, ..., m$$

$$\sum_{i=1}^{m} \sum_{j=1}^{Q_i} \alpha_i \beta_i t_{ij} \leq \alpha_0 \beta_0$$

$$t_{ij} \in \{ 0, 1 \} \quad \forall \; i = 1, ..., m \quad j = 1, ..., Q_i$$

This program can be viewed as a knapsack problem which is easily solved. Its solution value is clearly an upper bound on the optimal solution to problem (P). A reduction test is derived from this program which is used to update both P_i and Q_i of type i for all i = 1, ..., m. We can estimate a penalty value that results from forcing exactly r_{i*} pieces of a particular type i* to be cut from A_0 where $P_{i*} \leq r_{i*} \leq Q_{i*}$. The method outlined below is used to obtain penalty values for a particular type i* in R.

Let Z_{KN} (r_{i*}) be the upper bound obtained from (KNAP2) by forcing r_{i*} pieces of type i* to be in the solution of problem (KNAP2) and Z_{LB} be some (previously determined) lower bound to problem (P). Let

$$A_{i*} = \sum_{\substack{j=1 \\ j \neq i*}}^{m} P_j \alpha_j \beta_j \quad \text{and} \quad V_{i*} = \sum_{\substack{j=1 \\ j \neq i*}}^{m} \sum_{k=1}^{P_j} S_{jk}$$

represent the area of A_0 that is taken up with cutting out P_j pieces of each type j ($j \neq i*$) in R and the associated total value of the pieces cut, respectively. Z_{KN} (r_{i*})

is then given by

$$Z_B + V_{i*} + \sum_{j=1}^{r_{i*}} S_{i*j} + 2 \sum_{j=1}^{m} \alpha_j \beta_j \sum_{p \in \tilde{L}_j} \sum_{q \in \tilde{W}_j} u_{ipq} +$$

$$\beta_0 \sum_{r \in L} e_r + \alpha_0 \sum_{s \in W} f_s - \sum_{r \in L} \sum_{s \in W} I_{rs} Z_{rs}$$

where Z_B is the solution to the following knapsack problem:

$$\max \sum_{\substack{j=1 \\ j \neq i*}}^{m} \sum_{k=1}^{Q_j - P_j} S_{ij} t_{jk}$$

subject to

$$\sum_{\substack{j=1 \\ j \neq i*}}^{m} \sum_{k=1}^{Q_j - P_j} \alpha_j \beta_j t_{jk} \leq \alpha_0 \beta_0 - \alpha_{i*} \beta_{i*} r_{i*} - A_{i*}$$

$$t_{jk} \in \{ 0, 1 \} \quad \forall \; j = 1, ..., m \; (j \neq i*), \; k = 1, ..., Q_j$$

By investigating all values of r_{i*} where $P_{i*} \leq r_{i*} \leq Q_{i*}$ and comparing the corresponding upper bounds $Z_{KN}(r_{i*})$ obtained with Z_{LB}, we can update P_{i*} and Q_{i*} accordingly (see Reduction Test 3).

6.4 Subgradient Optimisation

As mentioned in section 6.2, once Lagrangean Relaxation has been applied to a general integer linear problem [equations (6.1) to (6.4)] we must then determine the value of the Lagrange multipliers λ^* that will optimise the upper bound on the problem. We are then interested in finding the vector λ^* which provides the tightest bound by solving the dual problem

$$(D): \qquad Z_D(\lambda^*) = \min_{\lambda} Z_D(\lambda^*)$$

where $Z_D(\lambda)$ is the upper bound obtained by the Lagrangean problem (PR_λ) [equations (6.5) to (6.7)].

Finding λ^* is not a simple task; however λ^* can be approximated by using a subgradient optimisation procedure. This is an approach for approximating the minimum of certain piecewise linear convex functions. It has been effective in handling some difficult large scale combinatorial problems and has already been applied to the generalized assignment problem, travelling salesman problem and multicommodity maximum flow problem with successful results. Computational performance and theoretical convergence properties of the subgradient method are discussed in *Held, Wolfe and Crowder* [1974].

The method basically involves the application of a gradient method to minimisation of $Z_D(\lambda)$ with some adaptation at the points where this linear function is nondifferentiable. In general, the gradient of $Z_D(\lambda)$ at differentiable points is given by $Ax - b$. At nondifferentiable points, the subgradient method chooses arbitrarily from the set of alternative optimal Lagrangean solutions to (PR_λ)

and uses the vector $Ax - b$ for this solution as though it were the gradient of $Z_D(\lambda)$. The result is a procedure that determines a sequence of values for λ by applying the formula

$$\lambda^{k+1} = \max \{ 0, \lambda^k - t_k (b - Ax^k) \}. \tag{6.35}$$

In this formula, t_k is a positive scalar step size and x^k is an optimal solution to $(PR_{\lambda}{}^k)$, the Lagrangean problem with dual variables set to λ^k.

The above multiplier updating procedure requires an initial vector λ^0 to start with; $\lambda^0 = 0$ is a natural choice. It is possible, however, to generate a better initial vector using an observation which is applicable to a particular type of problem.

Equation (6.35) also uses a stepsize t_k in a different way than it is normally set in a gradient method. A fundamental theoretical result given in *Held, Wolfe and Crowder* [1974] states that

$$\text{as } k \to \infty, \; t_k \to \infty \quad \text{and} \quad \sum_{i=1}^{k} t_i \to \infty$$

then $Z_D (\lambda^k)$ converges to its optimal value $Z_D (\lambda^*)$. A formula for t_k that has been proved effective in practice is given by:

$$t_k = \frac{\pi_k (Z_D (\lambda^k) - Z_{LB})}{\| S \|^2} \tag{6.36}$$

In this formula, Z_{LB} is the objective value of the best known feasible solution to (P), π_k is a scalar satisfying $0 \leq \pi_k \leq 2$ and $\| S \|$ is any norm of the subgradient vextor (b - Ax) e.g

$$\sum_{i=1}^{m} (b_i - \sum_{j=1}^{n} a_{ij} x_j^k)^2 .$$

The initial lower bound Z_{LB} can be obtained by applying a heuristic to problem (P). Frequently, the sequence $\{ \pi_k \}$ is determined by starting with $\pi_k = 2$ and reducing π_k by a factor of two whenever $Z_D (\lambda^k)$ has failed to improve in a specified number of iterations.

Justification of formula (6.36) as well as many other interesting results on the subgradient method is given in *Held, Wolfe and Crowder* [1974]. Unless we obtain a λ^k for which $Z_D (\lambda^k) = Z_{LB}$, there is no way of proving optimality in the subgradient method. To resolve this difficulty, the method is usually terminated upon reaching a specified iteration limit.

6.4.1 Implementation of Subgradient Optimisation for the NGC Problem

In this section we present a subgradient optimisation procedure used in an attempt to minimise the upper bound Z_{UB} (6.20) obtained from the Lagrangean Relaxation of the NGC problem (section 6.2.1). The procedure incorporating some of the reduction tests mentioned in section 6.3 is as follows:

(1) Choose initial values for the multipliers . No good indication exists on how

to determine good starting values - we used

$$u_{ipq} = 0 \quad \forall \; i=1, ..., M \;\; , \;\; p \varepsilon \tilde{L}_i \;\; \text{and} \;\; q \varepsilon \tilde{W}_i \qquad (6.37)$$

$$e_r = 0 \quad \forall \; r \varepsilon L \quad \text{and} \quad w_s = 0 \; \forall \; s \varepsilon W \qquad (6.38)$$

Determine an initial value for Z_{LB} - the lower bound on the problem. This can be done using any heuristic for the NGC problem (Chapter 7).

(2) Solve the Lagrangean program (LR) [equations (6.15) to (6.19)] with the current set of multipliers obtaining the optimal objective value Z_{UB} [equation (6.20)] and the associated variable values X_{ip}, Y_{iq}, Z_{rs}.

(3) Check if the Lagrangean solution (X_{ip}, Y_{iq}) and (Z_{rs}) is a feasible solution to the original problem (P) [equations (6.8) to (6.12)]. If feasible, then if $Z_{UB} > Z_{LB}$, update Z_{LB} with $Z_{LB} = Z_{UB}$ and STOP; else STOP. If not feasible, then if $Z_{UB} < Z_{min}$ (the minimum bound obtained so far), update Z_{min} with $Z_{min} = Z_{UB}$ and continue; else continue.

(4) Stop if $Z_{min} = Z_{LB}$ i.e the best Lagrangean upper bound and the lower bound (corresponding to a feasible solution) coincide; else go to (5).

(5) Perform the reduction tests of section 6.3 based on overlapping pieces, free area, free value, the number of cut pieces and the cut positions.

(6) Define the subgradient vectors U, E and F by:

$$U_{ipq} = 2\alpha_i\beta_i - \alpha_i\beta_i X_{ip} - \alpha_i\beta_i Y_{iq} - \sum_{s=q}^{q+\beta_i-1} \sum_{r=p}^{p+\alpha_i-1} Z_{rs}$$

$$\forall\ i = 1, ..., M,\ p \in \tilde{L}_i,\ q \in \tilde{W}_i \qquad (6.39)$$

$$E_r = \beta_0 - \sum_{i=1}^{M} \beta_i \sum_{\substack{p=r-\alpha_i+1 \\ p \in \tilde{L}_i}}^{r} X_{ip} - \sum_{s \in W} Z_{rs} \qquad \forall\ r \in L \qquad (6.40)$$

$$F_s = \alpha_0 - \sum_{i=1}^{M} \alpha_i \sum_{\substack{q=s-\beta_i+1 \\ q \in \tilde{W}_i}}^{s} Y_{iq} - \sum_{r \in L} Z_{rs} \qquad \forall\ s \in W \qquad (6.41)$$

(7) Stop if $U_{ipq} = 0$ (for all i, p and q), $E_r = 0$ (for all r) and $F_s = 0$ (for all s); else goto step (8).

(8) Calculate the step size t for use in updating the Lagrange multipliers by

$$t = \frac{\pi (Z_{UB} - Z_{LB})}{\| S \|^2} \qquad (6.42)$$

where $0 \leq \pi \leq 2$ and

$$\| S \|^2 = \sum_{i=1}^{M} \sum_{p \in \tilde{L}_i} \sum_{q \in \tilde{W}_i} (U_{ipq})^2_{u_{ipq} > 0} + \sum_{r \in L} E_r^2 + \sum_{s \in W} F_s^2 \qquad (6.43)$$

(9) Update the multipliers by :

$$u_{ipq} = \max (0, u_{ipq} - t\, U_{ipq}) \text{ for all } i = 1, ..., M,\ p \in \tilde{L}_i,\ q \in \tilde{W}_i$$

$$e_r = e_r - t\, E_r \text{ for all } r \in L$$

$$f_s = f_s - t\, F_s \text{ for all } s \in W \qquad (6.44)$$

(10) Go to (2) to resolve the Lagrangean program with this new set of multipliers unless one of the following conditions is satisfied:

(i) a sufficient number of subgradient iterations has been peformed

(ii) π falls below a very small positive value

in which case STOP.

At the end of the subgradient procedure, the optimal solution to the original NGC problem (P) may have been found (step 4), but if not, the best Lagrangean bound on the problem has been obtained which can then be used in a tree-search procedure to solve the problem (Chapter 7).

Solving the Lagrangean problem (LR) (step 2) for a given set of multiplier values $u_{ipq} \geq 0$, e_r and f_s, we may obtain a solution (X_{ip}, Y_{iq}) and (Z_{rs}) which is feasible to the original problem (P). In general, this occurs rarely in practice. However, it is not uncommon that the Lagrangean solution will be nearly feasible and can be made feasible with some modifications. For example, we can often obtain feasibility by solving different Lagrangean relaxations of (P) in which certain variables are preset to fixed values or a large number of variables are eliminated from the optimal solution, thus enabling smaller sized problems to be

solved. Such relaxations can be obtained at various nodes of a tree-search used to solve optimally the NGC problem (Chapter 7), providing us with good lower bounds on (P).

6.4.2 Computational considerations on the choice of step size

No good indication exists on how to determine a good sequence $\{ \pi_k \}$ which is used in the computation of the scalar step size t_k. For certain choices of π_k, the number of iterations needed to reach optimality or the proof of infeasibility of (P) will probably be higher. To gain insight into a more sensible procedure, we have tried a number of ways for setting such a sequence. We present two rules used in our computations.

In Rule 1, the sequence $\{ \pi_k \}$ was determined by setting $\pi_0 = 2$ and halving π_k whenever $Z_{UB}{}^k$ has failed to decrease in some fixed number of iterations (3 in our case). The subgradient procedure was terminated in a finite number of iterations (400) unless the scalar t_k dropped below 5 E-6 at an earlier stage.

In Rule 2, we followed the approach of *Held et al* [1974] in setting $\pi = 2$ for 2n iterations (where n is a measure of the problem size). In our case, n is taken to be equal to the sum of the sizes of the normal sets which are relative to a particular NGC problem i.e $n = |L| + |W|$. Then, we were successively halving both the value of π and the number of iterations until the number of iterations reached a threshold value of five; π was then halved every five iterations until the resulting π_k fell below 0.005 .

Rules 1 and 2 have been tested on twelve problems (B1-B12) drawn from the literature (*Beasley* [1985b]). We solved them using the subgradient method as described in section 6.4.1 without performing any reductions tests (i.e excluding step 5 of the subgradient procedure). The results of this experiment are found in Table 6.1 . In this Table we give, for each problem, a description of the data, the value of the integer optimal solution, the best lower bound Z_{LB} (corresponding to a feasible solution), this being obtained from the table of computational results given in *Beasley* [1985b]and the number of variables involved.

To form an idea of the convergence of each Rule for π_k , we also give for each problem, the best upper bound (Z_{min}) obtained from the subgradient procedure, the number of iterations required and the time taken to reach this value. In both cases, π_k converges to 0, with each successive value equal to half the value on the previous iteration. The results show that the quality of the bounds obtained using both rules is generally the same. They both converge to a value which is on average 5% away from the optimal solution (note that only in Problem B1, the duality gap is round 27% and in Problem B7, the optimal solution is found at the first iteration). However, the rate of convergence differs drastically. When Rule 2 is applied, the subgradient method converges in a considerably smaller number of iterations. Only in Problems B4 and B11, Rule 2 requires 5 and 17 iterations more to reach bounds which are nearer to the optimal solution by 1.5% and 0.9% respectively. As a result of better convergence, Rule 2 generally has a much lower computational cost. Figure 6.2 plots every twenty iterations, the least value found during the previous twenty iterations of the upper bound Z_{min} for Problem B9 using both Rules 1 and 2 (curves $Z_{min}{}^1$ and $Z_{min}{}^2$ respectively). This plot clearly shows the linear convergence of $Z_{UB}{}^k$ to $Z_D(\lambda^*)$ which is virtually assured by our step - size choices. It is also clear that using Rule 1, π_k starts converging at a much later stage. Since computational results showed that Rule 2

Details of Test Problems							
Problem Number	(α_0, β_0)	m	$\lvert\tilde{L}\rvert$	$\lvert\tilde{W}\rvert$	Optimal Solution (Z_{opt})	Lower Bound (Z_{LB})	Number of Variables
B1	(10,10)	5	7	6	164	164	186
B2	(10,10)	7	10	10	230	230	310
B3	(10,10)	10	9	10	247	246	369
B4	(15,10)	5	3	10	268	268	222
B5	(15,10)	7	6	10	358	358	297
B6	(15,10)	10	13	10	289	289	386
B7	(20,20)	5	14	20	430	430	606
B8	(20,20)	7	6	20	834	834	655
B9	(20,20)	10	18	17	924	924	817
B10	(30,30)	5	7	7	1452	1452	1127
B11	(30,30)	7	18	27	1688	1688	1325
B12	(30,30)	10	27	30	1865	1770	1659

Problem Number	Results of Rule 1			Results of Rule 2		
	Upper Bound (Z^1_{min})	Number of Iterations	Time to Obtain Z^1_{min}	Upper Bound (Z^2_{min})	Number of Iterations	Time to Obtain Z^2_{min}
B1	208	123	1.3	209	79	1.2
B2	257	218	4.5	258	100	2.6
B3	262	166	3.6	261	96	2.7
B4	275	70	0.7	271	75	0.7
B5	362	128	5.0	365	89	3.6
B6	317	178	5.2	317	110	4.5
B7	430	1	2.8	430	1	2.8
B8	919	400	18.6	922	122	8.9
B9	947	400	47.3	946	160	16.7
B10	1523	200	22.2	1533	133	23.3
B11	1818	176	39.2	1803	193	40.8
B12	1972	400	104.3	1964	240	120.7

Table 6.1 The Subgradient Method using two different Rules for setting π_K.

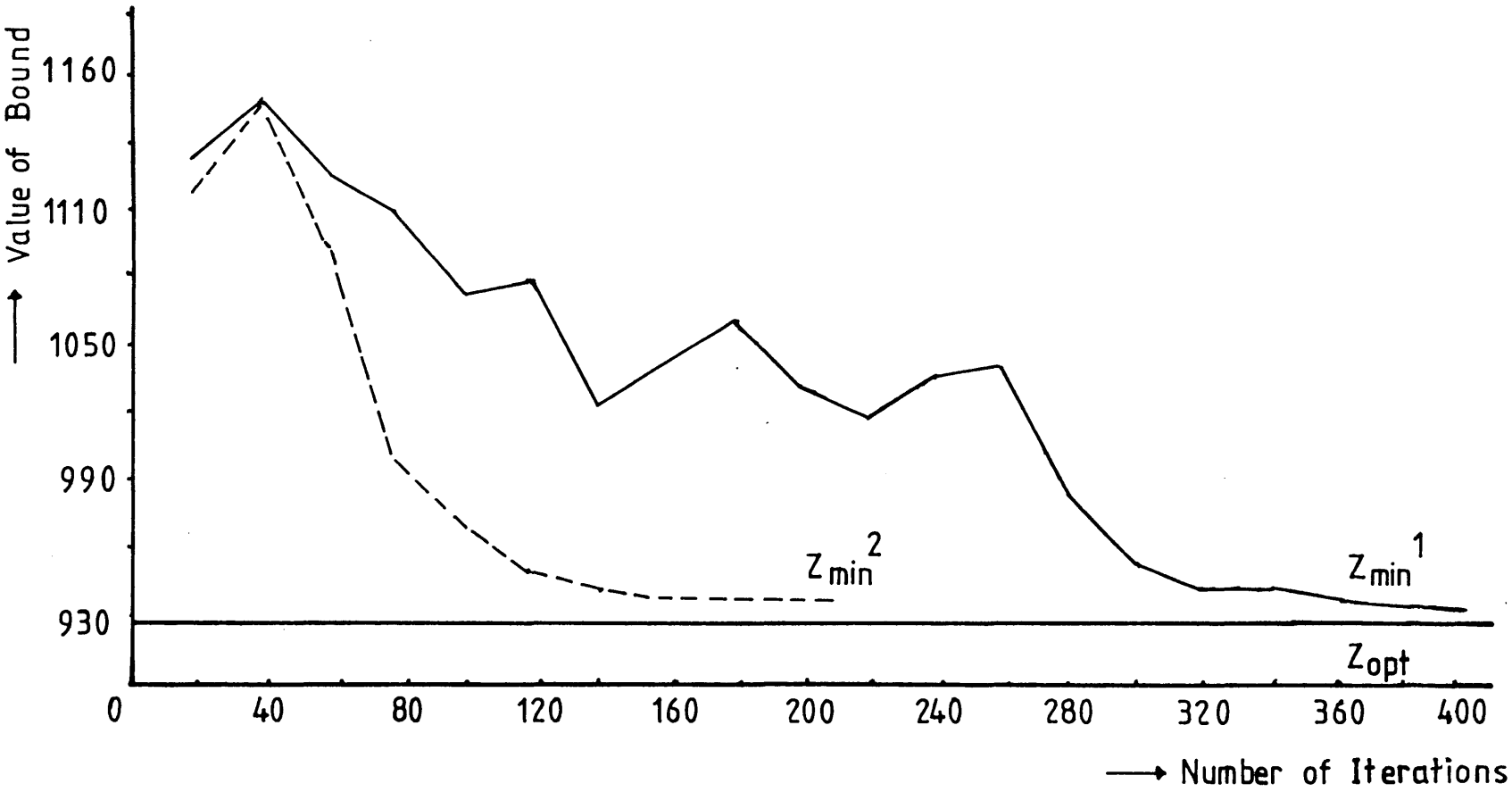


Figure 6.2 Descent of the Lagrangean Bound for Problem 9 using Rules 1 and 2.

performed better than Rule 1, we applied the former to compute the sequence $\{ t_k \}$.

The denominator of formula (6.42) includes basically any norm of the subgradient vectors U, E and F. In our computations, we used an expression which involved the sum of squares of these vectors [equation (6.43)]. Note that in the sum of squares of vector U, we included only those U_{ipq} for which the corresponding multipliers u_{ipq} were strictly greater than zero. This choice was based on the observation that only very few u_{ipq} multipliers ~ at most as many as the pieces in R i.e

$$\sum_{i=1}^{m} Q_i$$

~ are nonzero among all u_{ipq} ' s obtained at any iteration

$$\sim (= \sum_{i=1}^{m} | \tilde{L}_i | | \tilde{W}_i |) \sim$$

allowing us to compute a total sum of squares of vector U of much smaller value. Having obtained a small value for the denominator of formula (6.42), a sequence $\{ t_k \}$ was generated that did not converge to zero very quickly. On the other hand, summing over all U_{ipq} 's, would lead us to a choice of step size t_k that would still converge to zero, but more quickly. In general, if the step size converges to zero too quickly, then the subgradient method may converge to a point other than the optimal solution, provided convergence happens at all. This observation has been confirmed in a result given in *Held, Wolfe and Crowder* [1974] (also mensioned in section 6.4). Therefore, we used formulae (6.42) and (6.43) to compute $\{ t_k \}$ which performed well on a variety of problems. The results are shown in the section dealing with computational results.

6.5 A General Procedure based on Subgradient Optimisation and Reduction Tests

Before carrying out the subgradient optimisation procedure of section 6.4.1, for any NGC problem, we use some of the problem reduction tests described in section 6.3. Thus we develop a more general approach to solve a problem, at the end of which either the optimal solution is found or a good upper bound on the problem is obtained. In the latter case, a reduction in the size of the problem may be achieved, because the reduction tests may exclude some pieces from cutting, identify others as neccessary to be cut or eliminate a number of variables. A tree-search procedure for the NGC probelm can then be used to obtain the optimal solution (Chapter 7). The general procedure is as follows:

(1) Reduction: Carry out the first three reduction tests of section 6.3, namely the tests based on overlapping pieces and free area to reduce Q_i and then the Knapsack area program reduction test to update both P_i and Q_i for all i=1, ..., m .

(2) Normal Patterns: The sets of normal cuts L and W, $\tilde{L}_i$ and $\tilde{W}_i$ for all i=1, ..., m are calculated for the reduced problem of step (1).

(3) Subgradient Procedure: Carry out the subgradient procedure as described in section 6.4.1. If at any iteration, one of the following conditions is satisfied, then the problem is solved optimally and the general procedure is terminated with Z_{LB} being the value of the optimal solution:

$$(i)\ \min\left(\sum_{i=1}^{m} Q_i \upsilon_i,\ Z_{UB}\right) = Z_{LB}$$

$$(ii)\ \sum_{i=1}^{m} P_i \alpha_i \beta_i > \alpha_0 \beta_0$$

$$(iii)\ \sum_{i=1}^{m} P_i \upsilon_i > Z_{UB}$$

If no optimal solution is found at the end of the subgradient procedure, the set of Lagrange multipliers that gave the minimum upper bound (Z_{min}) are recalled. Let u*, e* and f* denote this set of multipliers.

(4) Reduction: The reduction tests of step 5 of the subgradient procedure are performed using u*, e* and f* . In addition, we carry out the two Knapsack area program reduction tests 1 and 7. Typically, these reductions remove a large number of the possible cutting patterns. If no reduction is made at this step, we terminate the general procedure; otherwise a further 30 iterations (arbitrarily chosen) of the subgradient procedure are performed, using the above three termination criteria (step 4), in order to see if any advantage can be taken from the problem reduction.

6.6 Computational Results with the given Procedure

The general procedure described in the previous section was coded in FORTRAN and was tested on a Cyber-855 computer. It was investigated computationally on twelve problems which were randomly generated by *Beasley* [1985b] (B1-B12). The method of data generation used is the following: m real numbers r_i for i = 1, ..., m are randomly generated from the uniform distribution

$U[0, \alpha_0\beta_0/4]$. The dimension α_i of each piece is generated from the integer $U[1, \alpha_0]$ and the dimension β_i is obtained by setting $\beta_i = \lceil r_i / \alpha_i \rceil$. An integer value υ_i for each piece, is set equal to $\alpha_i\beta_i$ multiplied by a real random number drawn from $U[1,3]$ and rounded down. A maximum number Q_i of pieces of type i that can be cut from A_0 is generated from the integer range $U[1,3]$ (note that $P_i = 0$ for all i=1, ..., m).

The twelve NGC test problems, included five, seven or ten types of pieces in R to be cut from stock rectangles A_0 of sizes (10,10), (15,10), (20,20) or (30,30). Table 6.2 gives details of the problems solved.

To obtain a measure of the effectiveness of the reduction tests, we use a reduction percentage $100 (1 - D_2/D_1)$, where D_2 and D_1 represent the value of

$$\sum_{i=1}^{m} (Q_i - P_i)$$

at the start and at the end of the general procedure respectively. In problems where the optimal solution is found by the general procedure, the reduction percentage is set to 100%. Otherwise, the larger this value the greater the reduction that has been achieved. The amount of reduction produced by the procedure in problem size is shown in Table 6.2 .

In that table we give, for each problem, the size of the sets $\tilde{L}$ and $\tilde{W}$ obtained for the reduced problem and the number of variables left after reduction. We also give the number of Lagrangean multipliers involved in the relaxed problem, together with the associated computation time required to obtain Z_{min}. Times are given in CYBER-855 seconds excluding time for input-output and the first reduction

step of the general procedure. The value of the integer optimal solution (Z_{opt}) obtained for each problem by the algorithm of Chapter 7, used to solve optimally NGC problems, is presented in Table 6.2. Z_{opt} allows us to evaluate the quality of Z_{min} by calculating the following percentage ratio:

$$r = 100 (Z_{min} - Z_{opt}) / Z_{opt}$$

The values of the initial lower bounds for each problem, shown in Table 6.2, were taken from *Beasley* [1985b], who developed a heuristic procedure capable of finding good feasible solutions from any Lagrangean solution. In many cases the lower bounds obtained were optimal.

From Table 6.2, we can see that for five out of the twelve test problems, our general procedure found the optimal solution. The initial values of Z_{LB}, used by the procedure for Problems B4 and B7 , were verified to be optimal by solving the associated Knapsack area program of reduction test 3 (section 6.3). In the case of Problems B5, B9 and B10, the general procedure found the optimal solutions by performing 53, 43 and 1 subgradient iterations respectively. This does not necessarily imlpy that the linear programming relaxation of problem (P) gives an integer solution for these problems (since we do not use the simplex method , we do not necessarily discover integrality when a problem is solved). However, it can be seen from Table 6.1, that for these problems, the gap between the best upper bound (Z_{min}^2) obtained by applying the subgradient procedure of section 6.4.1 (Rule 2 is used for the choice of step size) to the original problem and the integer optimum is very small. In particular, it is estimated to be 1.1, 1.9, 2.4, 5.6 and 0 % for Problems B4, B5, B9, B10 and B7 respectively.

The computational results of the general procedure shown in Table 6.2,

Problem Data						Reduction (%) $(1-D_2/D_1)$	Upper Bound (Z_{min})	Lower Bound (Z_{LB})	Duality Gap (η) (%)	Number of Subgradient Iterations	Time in CDC CYBER-855 seconds	Number of Variables Left After Reduction	Number of Multipliers
Problem Number	(α_0,β_0)	m	$\lvert\tilde{L}\rvert$	$\lvert\tilde{W}\rvert$	Optimal Solution (Z_{opt})								
B1	(10,10)	5	7	6	164	30	194	164	18.2	75	1.6	163	113
B2	(10,10)	7	10	10	230	11	257	230	11.7	100	2.8	288	502
B3	(10,10)	10	9	10	247	38	261	246	5.6	96	2.5	280	538
B4	(15,10)	5	-		268	100	268	268	-	-	0.04	-	-
B5	(15,10)	7	6	10	358	100	358	358	-	53	1.2	234	157
B6	(15,10)	10	13	10	289	20	317	289	9.6	110	4.5	370	769
B7	(20,20)	5	-		430	100	430	430	-	-	0.04	-	-
B8	(20,20)	7	6	20	834	15	921	834	10.4	122	7.13	638	675
B9	(20,20)	10	18	17	924	100	924	924	-	43	5.2	714	1833
B10	(30,30)	5	7	7	1452	100	1452	1452	-	1	1.5	962	160
B11	(30,30)	7	18	27	1688	13	1798	1688	6.5	198	33.5	1301	2110
B12	(30,30)	10	27	30	1865	9	1963	1770	5.2	240	96.9	1630	5281

Table 6.2 Performance of the general procedure on 12 problems from the Literature.

demonstrate that a large reduction in problem size (small number of variables left after reduction) is achieved by the reduction tests leading to a better performance in the subgradient optimisation (fewer iterations, less computational cost, tighter upper bound). An observation drawn from the computational experience is that time increases with problem size. For the two largest problems that we solved, B11 and B12, each involving 1301 and 1630 variables - after being reduced by 13 and 9 % respectively - the computing time reached the values of 33.5 and 96.9 seconds respectively.

Bounds on problems B1 - B12 have also been obtained by *Beasley* [1985b]. They are derived from a Lagrangean relaxation of a 0-1 integer programming formulation of the NGC problem. Comparing with his results, we notice that our procedure has a better performance on the larger problems, e.g the bounds we obtained for Problems B11 and B12 are nearer to the integer optimum by 3.7 and 1 % respectively, with no additional computational cost.

6.7 Conclusions

We applied Lagrangean relaxation to a 0-1 integer programming formulation of the NGC problem. Subgradient optimisation was used to optimise the bounds derived from it. Tests for problem reduction, both before the subgradient procedure and at each subgradient iteration, were given and shown to produce a large reduction in problem size.

Computational experience of the method on a number of problems of differing size was presented.

For problems, in which the optimal solution has not been found by the procedure described in this chapter, the bounds obtained can be embedded in a tree-search procedure used to solve these problems exaclty. Such a procedure is developed in the following chapter.

CHAPTER 7

A TREE - SEARCH ALGORITHM FOR THE NGC PROBLEM

7.1 Introduction

In this chapter, we apply a tree-search procedure to the NGC problem as it has been defined in Section 6.1. First we describe how a finite number of orthogonal cutting patterns of the cut rectangles on A_0 can be generated. From amongst these we choose the highest value pattern of demanded rectangles. We then show how the process of obtaining such a most valuable orthogonal pattern of rectangles can be considered as a tree - search. Certain conditions are derived and imposed in order to limit the size of this tree - search. A bounding procedure is then incorporated into the above tree so as to reduce the amount of search necessary before the optimum solution is obtained. The Lagrangean bound on the solution of the problem obtained in the way described in Chapter 6 is used during the search.

The computational performance of the algorithm is illustrated by tests

performed on randomly generated problems. Results are given in the last section of this chapter.

7.2 Description of the problem

We are given a rectangular stock-plate A_0 of length α_0 and width β_0 and a demand for P_j rectangles of dimensions (α_j, β_j) and value υ_j for each $j = 1, ..., m$. Suppose that the total number of given rectangles is equal to M, so that

$$M = \sum_{j=1}^{m} Q_j.$$

We denote this set of pieces by $R = \{ r_1, r_2, ... , r_M \}$. The problem is then to obtain an orthogonal layout of all of the demanded rectangles in the stock plate if one exists. Otherwise, a set of rectangles is to be cut from A_0 that has the highest possible total value, subject to the constraints on the number of pieces cut.

If μ_j is the number of rectangles of type j included in a particular cutting arrangement for A_0, then a "combination" is defined as any set of μ_j, $j = 1, ... , m$ such that all μ_j are non-negative integers and the set of rectangles represented by the μ_j can be fitted within the stock-plate. We define the number of rectangles of type j included in the kth combination by μ_{kj}. The problem is then to find a combination k such that

$$\underset{k}{\text{Maximise}} \quad z = \sum_{j=1}^{m} \upsilon_j \mu_{kj}$$

subject to

$$P_j \leq \mu_{kj} \leq Q_j \quad \forall \quad j = 1, \ldots, m$$

The arrangement of a combination k within the stock plate is termed a "pattern" (π_k). For most combinations there will be a number of such cutting patterns. These are produced by using a cutting process that involves restrictions upon positioning of cuts in A_0. Only orthogonal patterns are considered so that each of the edges of the cut rectangles is parallel to an edge of A_0. However, many of these orthogonal patterns are not necessarily guillotineable.

In the following section, a procedure referred to as the "Enumerative Procedure" is presented which is used to generate all possible patterns corresponding to all combinations of rectangles in R. Then we describe how these patterns can be implicitly enumerated as a general tree search.

7.3 Enumerative Procedure

We assume that each rectangle r_i in R has a fixed orientation i.e length α_i and width β_i. Then we arbitrarily choose some ordering for the set R of oriented rectangles. One way of evaluating the contribution of each rectangle in the final solution is using the ratio of each value v_i over its area $\alpha_i\beta_i$. Assume that a possible sequence $r_1, \ldots, r_M$ is chosen by placing the rectangles in decreasing order of the ratio $v_i / \alpha_i\beta_i$. Using this ordering we describe a method of generating a finite number of orthogonal patterns corresponding to all possible combinations of the demanded rectangles in A_0.

The cutting process used for generating a pattern involves a heuristic

sequential placement of rectangles in R in the large stock-plate A_0. A rectangle r_i is selected according to some criterion and then packed optimally with respect to those already placed. The logic of the placement process follows a single rule which may be described as "left-most downward placement" using the following referencing method. The lower left hand corner of the stock-plate A_0 will be referenced as the point (0, 0). The rectangle selected for packing next is placed with its bottom left hand corner (b. l. h. c.) at a location in A_0 which firstly minimises the X co-ordinate of the placement of the rectangle and secondly (if more than one such location is possible) at a location which minimises the Y co-ordinate of the placement. This process results in placing the selected rectangles as far to the left of A_0 as possible and then moved as far as possible downward. It is implicit in this procedure that, within a particular pattern a rectangle is packed optimally with respect to those already placed but without regard for those remaining i.e once a rectangle is placed it is fixed until the end of the pattern.

Short [1973] explored in detail various heuristics and interactive techniques for solving a variation of the problem stated in section 7.2. He supposed that the length α_0 of the stock-plate was effectively infinite. His aim was to obtain a layout of the demanded rectangles on the stock strip that would minimise the length of the strip. A heuristic sequential placement procedure for solving this problem was presented using the rule of "left-most downward placement" of rectangles on the stock strip. This procedure was tested in a number of test problems. The results obtained were not satisfactory because of the lack of consideration of the consequences of a particular placement on the remaining subproblem. So, it would be desirable to incorporate some facility whereby the cutting process could "back-track" and alter the selection and placement of some of the fixed rectangles enabling us to generate different patterns. In this case we would have to determine what criteria might be used when a back-track should take place, how far it should

go, what the changes to the selection and placement process should be and how the various patterns could be processed and recognised. The method we describe below, based on an extension of the left-most downward sequential placement procedure described by *Short* [1973] and *DeCani* [1979], carries out the above functions allowing us to develop an exact approach for solving the NGC problem.

It will be convenient to illustrate the concepts involved using an example. We consider the problem of cutting five rectangles from a stock-plate A_0 of size (8, 6) using the dimensions and values of rectangles in R as shown in Figure 7.1.

The first rectangle in R, r_1, is placed in the b.l.h.c. of the stock-plate A_0. This placement is illustrated for the example in Figure 7.2 by the second layout. Fig. 7.2 shows all possible layouts of rectangles of set R in A_0. A layout can be characterised by ξ(k, l), where ξ is the identification number (the order in which the layout is produced), k is the identification number of the completed cutting pattern corresponding to the layout ξ and l is the number of rectangles cut by the kth pattern. When a layout is identified by ξ only, this means that the current layout does not correspond to any completed pattern e.g. -17- represents the 17th layout produced by the method in the process of generating a possible pattern corresponding to a combination of rectangles including rectangles r_1 and r_2 from set R. One such possible pattern is described by 18 (9, 3) representing the 9th pattern generated by the method resulting in three rectangles being cut from A_0.

Once we placed rectangle r_1 at position (0, 0) in A_0, we then consider allowable positionings for the rest of rectangles in R. According to a selection process which will be described later in the chapter, we first pick rectangle r_3. The only allowable positionings for r_3 are those in which it has one of its edges collinear with an edge of r_1 and where no further vertical-downwards or

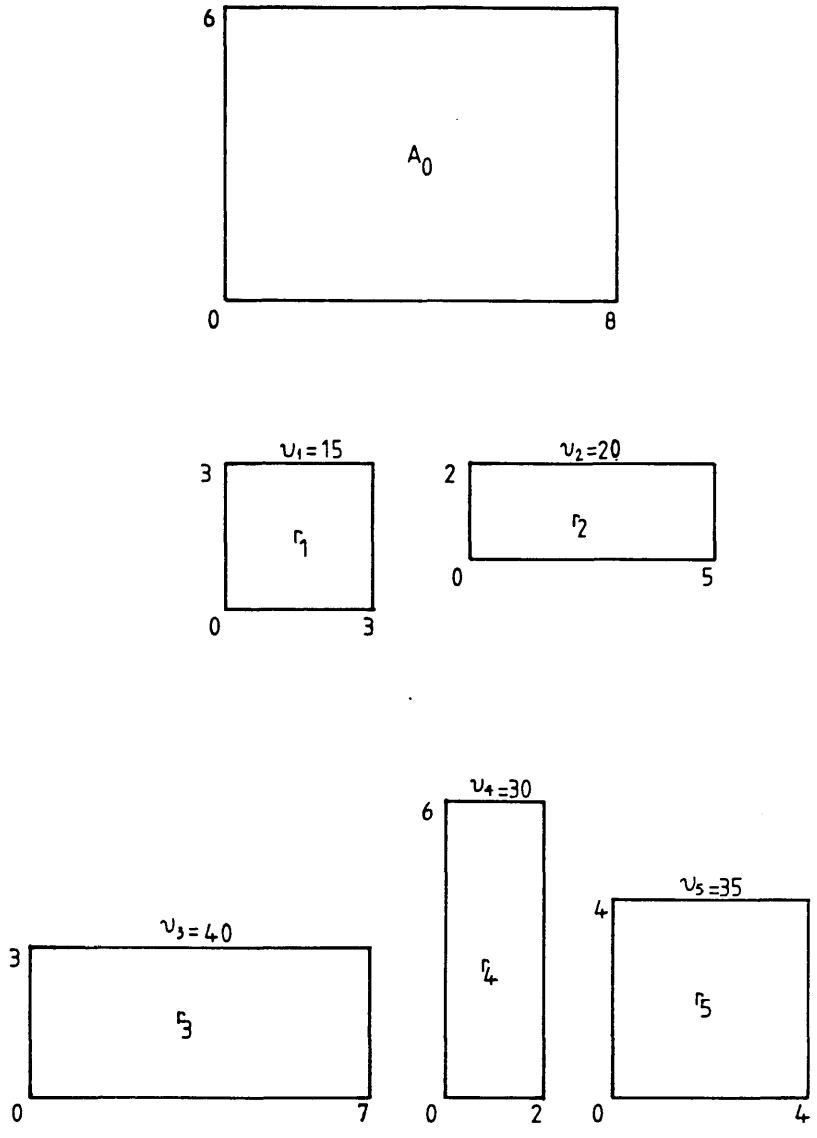


Figure 7.1 Stock - plate and pieces in R used for Illustrative Example.

horizontal-leftwards movement of r_3 is possible. The resulting feasible layout is the only one allowable for the rectangles r_1 and r_3 and it is illustrated in Fig. 7.2 by layout 3. Similarly, layouts 5 and 7 represent the two allowable layouts of rectangles r_1 and r_2 with r_2 being the second possible rectangle (according to the selection process) to consider for placement once r_1 has already been placed. Thus, the set of all allowable combinations of r_1 with only one other rectangle in R, when the former is placed at the b. l. h. c. of A_0, is given by {3, 5, 7, 9 (4, 2), 10}. This set represents all allowable combinations resulting from layout 2 in the way described above and is denoted by T (2).

To each layout ξ of the above set ($\xi \varepsilon T$), corresponds another set T (ξ) of allowable layouts obtained by placing each one rectangle in turn, of those in R not included in the current layout, at all allowable positions in ξ. For example, we consider an allowable layout of rectangles r_1 and r_2 represented by 5. The only allowable positionings for r_4 are those obtained by applying the left-most downward placement rule to layout 5. The resulting feasible layout is the only one allowable for the rectangles r_1, r_2 and r_4 corresponding to layout 5, and it is illustrated in Fig. 7.2 by layout 6 (2, 3). Neither of rectangles r_3 and r_5 can fit in layout 5, so no further layouts are produced corresponding to layout 5. As a result, T'(5) = { 6 (2, 3) } represents all allowable layouts resulting from layout 5, with each rectangle in R being tested in turn for placement in A_0.

In general then, for each of the allowable layouts (ξ say) of (i-1) rectangles in A_0, the only allowable positionings for the ith rectangle are those in which it has at least one of its edges collinear with an edge of at least one of the (i-1) rectangles and where no further vertical-downwards or horizontal-leftwards movement of the ith rectangle is possible. The resulting feasible layouts are the only ones produced from layout ξ.

We carry out the above procedure to generate all possible layouts of rectangles in A_0 starting with placing each rectangle of R, in turn at location (0,0) of A_0 i.e. initially, we obtain sets T (12), T (24), T (29) and T (37) by placing r_2, r_3, r_4 and r_5 at the b.l.h.c. of A_0, respectively. All resulting layouts in our example are shown in Fig. 7.2.

For certain layouts (ξ say) of (i - 1) rectangles, there is no allowable position into which any ith rectangle can be feasibly fitted. Such layout ξ of (i - 1) rectangles is then referred to as a " terminal layout ". This means that a complete cutting pattern (k say) is generated corresponding to the layout ξ of (i-1) rectangles in A_0, which is thus identified as layout ξ (k, i-1). A layout of all M rectangles in A_0 is also conventionally taken to be a terminal layout. All terminal layouts of rectangles in A_0 should be determined by the enumerative procedure, producing a set of cutting patterns for A_0 which is referred to as the " Basic Pattern Set ". This set consists of all possible patterns corresponding to the various combinations of rectangles in A_0, with each pattern k satisfying the following conditions:

(i) No further rectangle can be added in A_0.

(ii) No rectangle in the layout representing the pattern can be moved in either a vertical-downwards or a horizontal-leftwards direction.

Each pattern k, contained within the Basic Pattern Set, has a total value V_k for its cut rectangles given by:

$$V_k = \sum_{i=1}^{l} v_i$$

where l represents the number of rectangles produced by the pattern. The

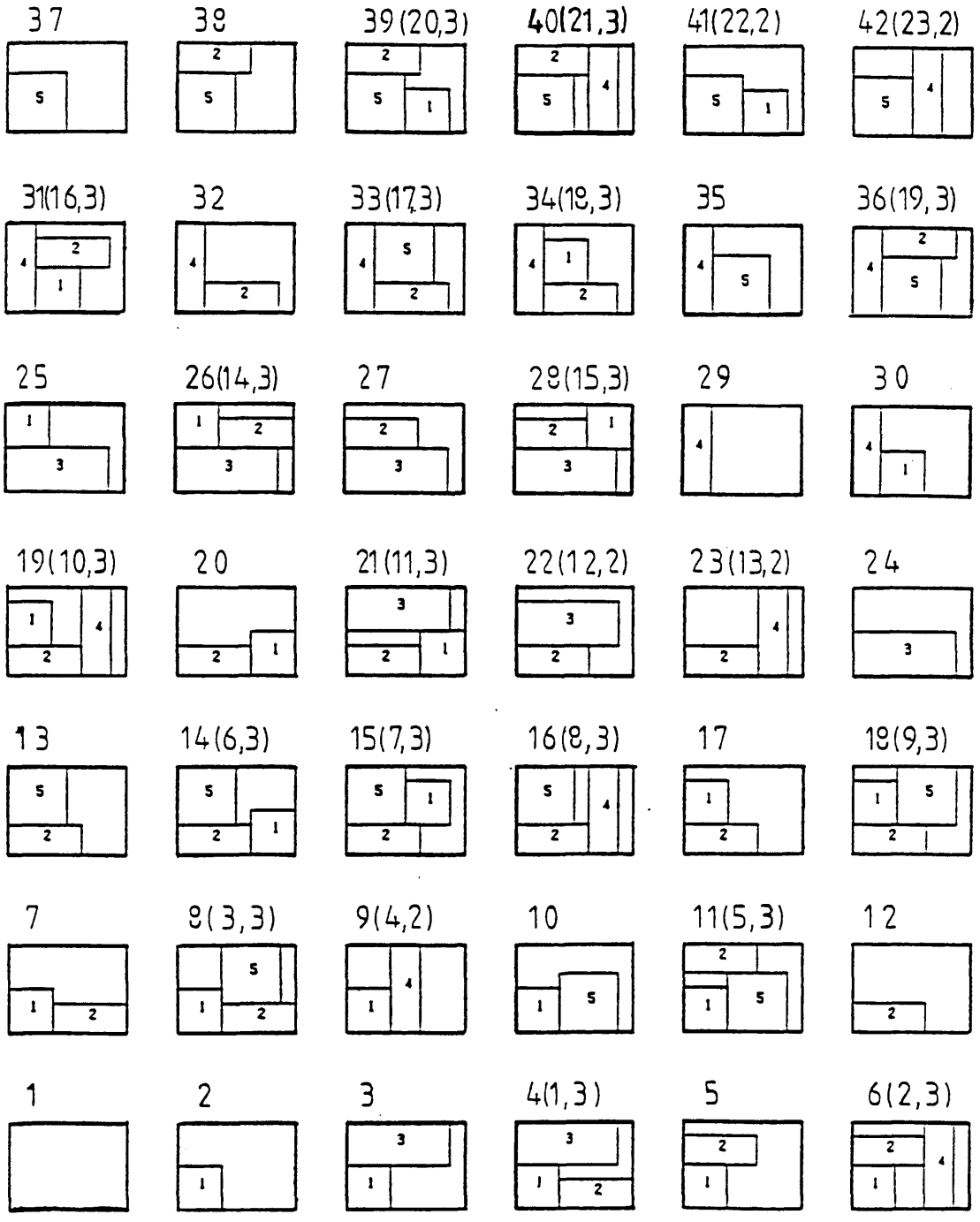


Figure 7.2 All allowable layouts of rectangles in the Illustrative Example of section 7.3.

enumerative procedure then continues by choosing from amongst all patterns in the Basic Pattern Set one having the highest total value V_k. This results in an optimum pattern of all M rectangles in A_0 if one exists. Otherwise, the described procedure determines a most valuable orthogonal pattern of rectangles in A_0.

In the Illustrative Example, the Basic Pattern Set consists of 23 patterns as shown in Fig. 7.2. The optimum value occurs at layouts 16 (8, 3), 33 (17, 3), 36 (19, 3) and 40 (21, 3) giving four patterns corresponding to the combination of rectangles r_2, r_4 and r_5 in A_0.

7.3.1 Tree representation of the cutting process

The enumerative method for generating all possible patterns in A_0, as described in section 7.3, involves two main processes: the selection and the cutting. The former will be described in the next section. The cutting process, involving the sequential placement of rectangles, can be recorded in terms of a tree which is described below.

All possible patterns in the Basic Pattern Set can be generated by developing a tree, where a branching represents placement of a rectangle in A_0, using the rule of "left-most downward placement". Thus, the branches emanating from the root node of the tree correspond to the placement of all rectangles at the b.l.h.c. of A_0 and each node at the end of a branch represents a pattern of the rectangles produced with the corresponding rectangle being placed at the b.l.h.c. of A_0.

The tree corresponding to the Example presented in section 7.2 is shown in Fig. 7.3. Each node in the tree corresponds to a layout of rectangles in A_0, shown

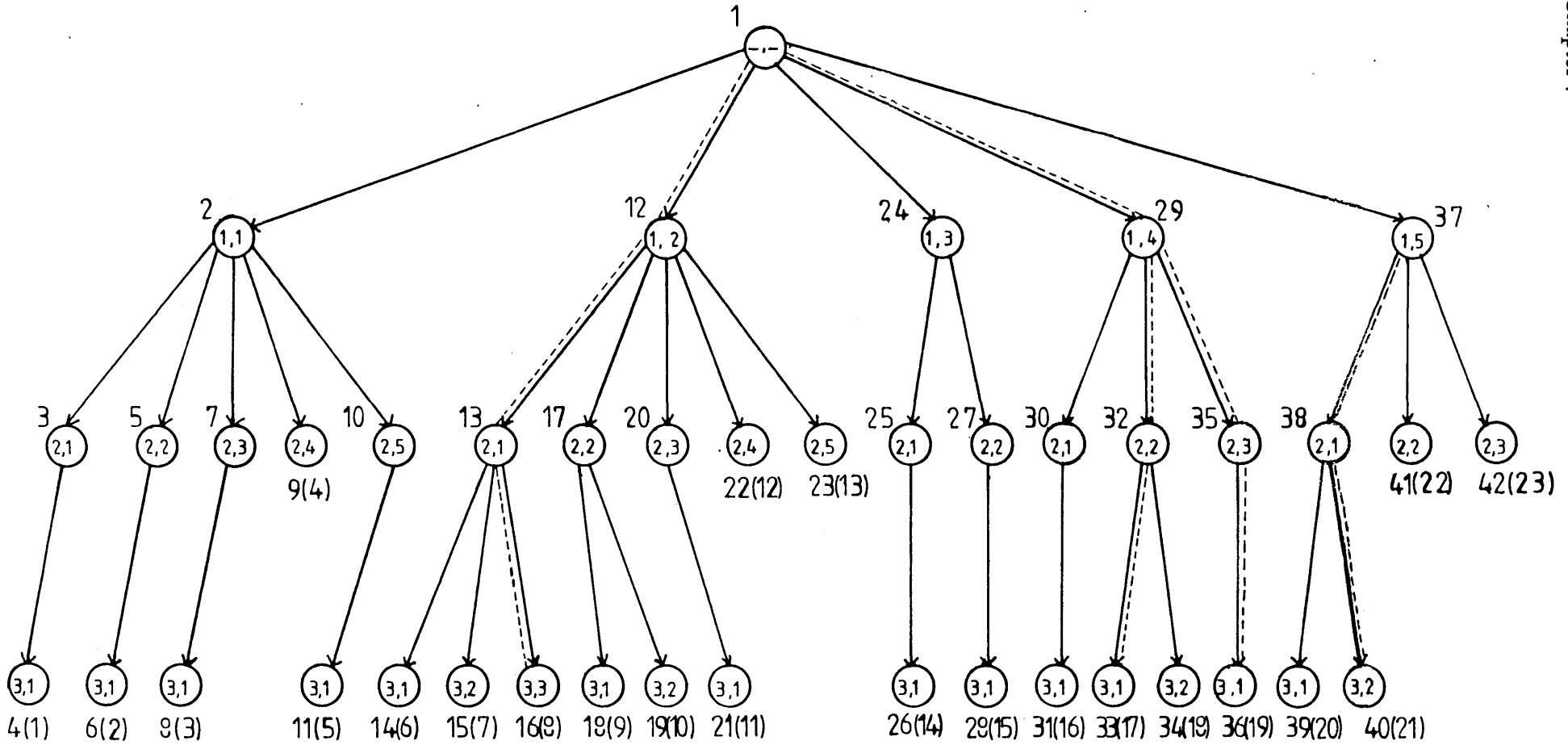


Figure 7.3 Representation of the Enumerative Procedure by a Tree for the Illustrative Example of section 7.3.

in Fig. 7.2 and is given two labels:

(i) a number n representing the identification number of the node

(ii) a pair of integers (l, ν) shown within each circle, indicating the level (l) in the tree in which node n lies and the order (ν) in which it is derived from its parent node. (Each node n in the tree has a unique parent node from which it is derived. This is shown in Figure 7.3 by the use of an arrow originating from the parent and pointing to node n).

A node n with label (l, ν) in row l of the tree corresponds to an allowable layout ξ ($n = \xi$) of l rectangles in A_0. A terminal node with label (l, ν) corresponds to a terminal layout ξ (k, l) representing the kth pattern in the Basic Pattern Set which produces l rectangles from A_0 and is denoted by n (k). Thus the Basic Pattern Set is represented by all terminal nodes of the tree.

The notation (l, ν) assigned to node n which is derived from a parent node ($l - 1$, ν') should be interpreted as meaning that the allowable layout of l rectangles corresponding to node n has been obtained by placing any rectangle at an allowable position to the layout of ($l - 1$) rectangles corresponding to node ($l - 1$, ν'). The aggregate of all resulting nodes (l, ν) taken over all ν corresponds to all allowable placements of only one rectangle in R to the layout of ($l - 1$) rectangles generated at node ($l - 1$, ν'). For example, nodes 3, 5, 7, 9 (4) and 10 of the tree shown in Figure 7.3 correspond to all allowable layouts obtained by placing in turn only one rectangle from Set R to the layout corresponding to node (1, 1).

We, therefore, see that the procedure for generating all patterns, as presented in section 7.3, is equivalent to determining all terminal nodes corresponding to the above tree. The determination of the maximum value pattern can, therefore, be thought of as searching the tree for the corresponding terminal

node.

Thus, the dotted lines, as shown in Fig. 7.3, indicate the paths leading to the four optimum patterns produced for the Illustrative Example.

7.3.2 A selection rule in the sequential placement of rectangles

The enumerative procedure, as described in section 7.3, requires the development of a selection rule to determine the next rectangle to be placed in an allowable position to a layout ξ of l rectangles in A_0 so that an allowable layout ξ' of $(l+1)$ rectangles is produced. This selection rule represents the branching rule used to generate a node n', representing layout ξ', from parent node n representing layout ξ.

Short [1973] investigated a number of selection rules in developing heuristic methods to solve the problem of Optimal Batch Execution on a Multi-processing computer (Two-Dimensional Packing Problem). The first approach he examined, was based on ranking the rectangles according to some feature; either length, area, maximum dimension, perimeter length or diagonal length (packing rectangles in pre-determined order). It is apparent from the results that no one criterion was superior to all others in every case (a number of trial problems was tested with the different selection rules). He then tried a hierarchy of criteria, including minimizing a measure of " probable waste " obtaining some improvement in his results. Our selection process is based on the rule that uses the measure of " probable waste ".

Since the problem of " value " maximisation is closely related to the problem

of " waste " minimisation, it is desirable to generate cutting patterns of rectangles in A_0 of high value that also have high utilisation factors (the utilisation factor indicates the percentage of the surface area of A_0 used by a cutting pattern). It would then appear reasonable to minimise the waste caused by selection and placement of each rectangle. The problem is to determine a measure for waste which can be attributed to a rectangle (it is the sum of areas in A_0 likely to be unfilled after placing a rectangle in a particular position to the current layout).

The measure of " probable waste " used in the sequential placement of rectangles of the enumerative procedure is a combination of two types of waste. It is either the current value for total waste (if the rectangle is placed in a position such that it projects beyond the right-hand end of the current layout in A_0), or the waste to the left of the trailing edge of the rectangle when placed. This measure effectively truncates rectangles such that for evaluation purposes they do not project beyond the right-hand end of the current layout in A_0. It represents the current waste which will be unusable later. This interpretation of " probable waste " is then incorporated in the selection process of the enumeration algorithm described in the next section.

Two possible measures of " probable waste " are illustrated in Fig. 7.4. Consider nodes 13 and 32 shown in Fig.7.3 for the Illustrative Example of section 7.3. Node 13 represents a layout ξ_1 of rectangles r_2 and r_5 in A_0 and node 32 represents a layout ξ_2 of rectangles r_2 and r_4. The waste attributed to rectangle r_1 by placing it in positions (4, 2) and (2, 2) in A_0 to layouts ξ_1 and ξ_2 respectively, is shown in Fig. 7.4.

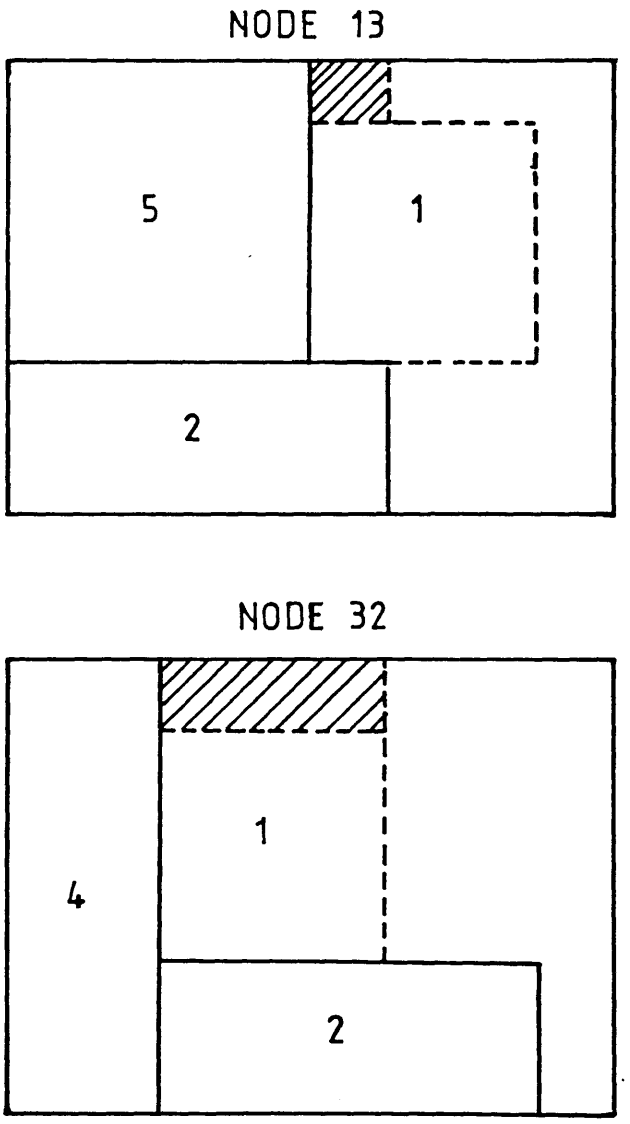


Figure 7.4 Measures of "Probable Waste"

7.3.3 Description of the Enumerative Algorithm

Whichever technique is employed to implement the enumerative procedure described in Section 7. 3, a critical feature is the data structure used to represent a cutting pattern. The structure used must be such that it is simple to test if different sequences of placements of rectangles lead to the same pattern and to rearrange the structure when transforming one pattern into another. To achieve these requirements for all types of patterns the structure described below is adopted.

The state at a node n in level l of the enumerative tree is described by four lists L_1, L_2, L_3 of rectangles and list S of locations in A_0. L_1 is the list of l rectangles placed in A_0 along the path that leads from the root of the tree to node n. It represents a layout denoted by ξ or $\xi\,(k, l)$ of l rectangles in A_0, with eight entries $(\bar{r}_i, r_i, j_i, \theta_j, p_i, q_i, \alpha_i, \beta_i)$ for each rectangle r_i $(i = 1, ..., M)$ where

$\bar{r}_i$ represents the last rectangle placed in A_0 in the current layout ξ before adding rectangle r_i

j_i is the type of rectangle r_i

θ_j represents the number of rectangles of type j included in the current layout after placement of r_i

p_i, q_i represent the X and Y co-ordinates of the b. l. h. c of rectangle r_i

α_i, β_i represent the length and width of rectangle r_i.

L_2 is the list of rectangles such that by placing any one of them in an allowable position to the allowable layout of l rectangles represented by list L_1, an allowable layout of $(l + 1)$ rectangles can be obtained representing a node derived from node n. In list L_2 each rectangle r_i is represented by a six - part label $(r_i, j_i, p_i, q_i, \alpha_i, \beta_i)$ where all five entries have the same meaning as

above.

L_3 is the list of rectangles such that none of them is ever placed in any allowable position in the layout of l rectangles corresponding to node n. Each rectangle r_i in this list is represented by a four - part label (r_i, j_i, α_i, β_i) with these notations being interpreted as above.

Let the number of rectangles included in lists L_1, L_2 and L_3 corresponding to a node n be denoted by $| L_1 |$, $| L_2 |$ and $| L_3 |$ respectively. $| L_1 |$ represents the level in which node n lies and $| L_2 |$ represents the number of nodes in the tree emanating from the current node n. Clearly there is no overlapping between the three lists. If $| L_2 | = 0$ then node n is a terminal node at which a pattern n in the Basic Pattern Set is produced.

The state at node 32 after nodes 33 and 34 have been generated, shown in Fig. 7. 3 for the Illustrative Example of Section 7. 3 would be described by the lists of rectangles shown in Table 7. 1, using i to head the column representing the number of the rectangle.

The state at a node n is not fully described by lists L_1, L_2 and L_3 of rectangles. It is necessary to maintain a full list S of allowable positions in A_0 in the layout corresponding to node n for possible placements of rectangles. Since the rule of left - most downward placement is applied by the cutting process, the task of generating possible locations (points at which the b. l. h. c. of a rectangle might be placed) is relatively simple. Each time a rectangle is placed, two additional locations are formed. One of these lies on the top edge of the rectangle projected back as far as possible to the left. The other location lies on the right - hand edge of the rectangle projected as far as possible downward. In list S, each location is represented by (p, q) where p and q are its X and Y coordinates respectively.

List L_1

i	$\tilde{r}_i$	r_i	j_i	θ_j	p_i	q_i	α_i	β_i
1	A_0	4	4	1	0	0	2	6
2	4	2	2	1	2	0	5	2

List L_2

i	r_i	j_i	p_i	q_i	α_i	β_i
1	5	5	2	2	4	4
2	1	1	2	2	3	3

List L_3

i	r_i	j_i	α_i	β_i
1	3	3	7	3

Table 7.1 Lists L1, L2 and L3 of rectangles corresponding to node 32 of Fig. 7.3.

The Set of allowable locations in A_0 at node 32 , shown in Fig. 7. 3 (before nodes 33 and 34 are generated), would be described by S = { (7, 0), (2, 2) }.

Before describing the enumerative algorithm that generates all possible patterns for A_0, important points need to be noted.

Suppose that a set of rectangles has been placed in A_0 in the ordered sequence of $r_1, r_2, ..., r_i, ..., r_l$ producing a pattern k of l rectangles, with a rectangle r_i being placed with its b. l. h. c. at location (p_i, q_i). Let this layout ξ (k, l) be described by List L_1 (ξ). Considering the same combination of l rectangles, it is possible that the enumerative procedure will produce layouts which are identical to ξ (k, l) by placing the l rectangles in any other ordered sequence in which each rectangle r_{i*} ($r_{i*} = r_i$) is placed with its b. l. h. c. at location (p_i, q_i) in A_0. This results in duplicated patterns of the same set of rectangles. An example of identical patterns is presented in Fig. 7. 5 (layouts 4 and 6).

Central to the enumerative procedure is the concept of " equivalent " patterns. This is best shown by an example illustrated in Fig. 7.6. Consider patterns 5 and 9 represented by layouts 11 (5, 3) and 18 (9, 3), respectively, of rectangles r_1, r_2 and r_5 in A_0 with reference to the illustrative Example of Fig. 7. 2. Patterns 5 and 9 corresponding to the same combination of rectangles are clearly not identical. It can be seen from Fig. 7. 6 that two closed wasted areas are produced by each pattern (these areas are denoted by W_1 and W_2 for the 5^{th} pattern and W_1' and W_2' for the 9^{th} pattern) . Each wasted area resulting from cutting A_0 by an orthogonal pattern corresponds to a closed space bounded by straight lines which are orthogonal to each other. If we detach the four wasted areas

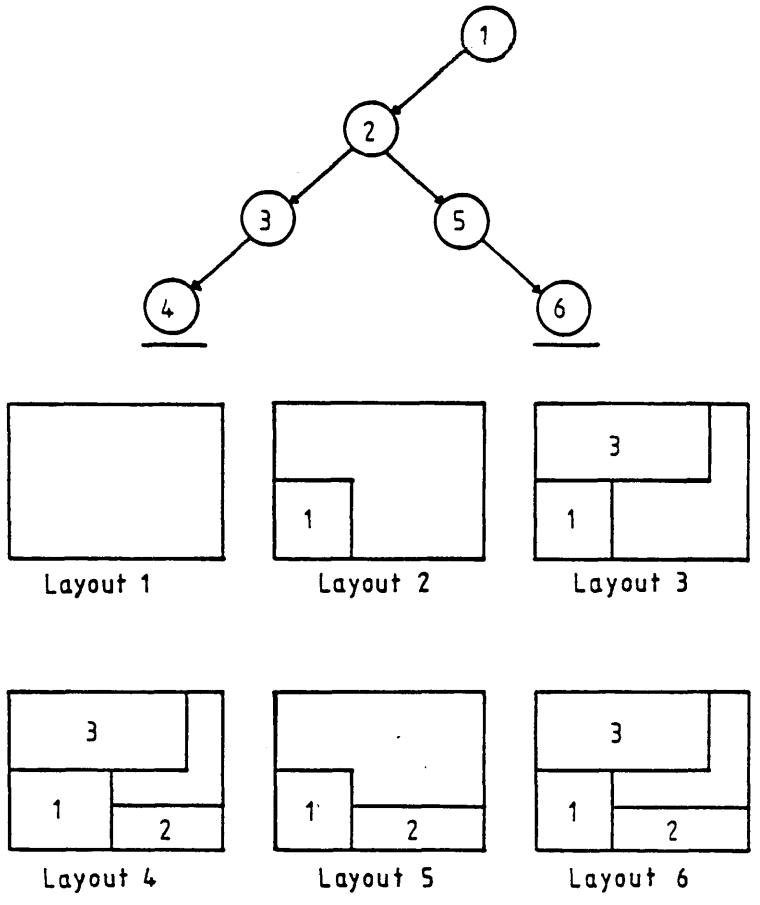


Figure 7.5 Identical Patterns.

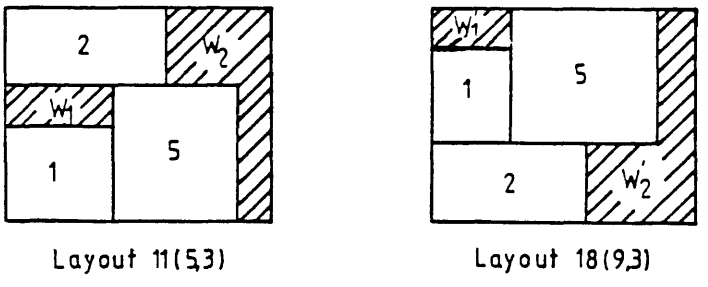


Figure 7.6 Equivalent Patterns.

from both layouts, we notice that area W_1 fits exactly in area W_1'. Similarly, by taking the image of area W_2 with respect to the horizontal axis we obtain an orthogonal shape that can be mapped exactly on area W_2'. This means that cutting a combination of rectangles from A_0 using different patterns may result in identical wasted areas which can be used for further processing in exactly the same way. The patterns of rectangles producing identical waste are referred to as " equivalent ".

Four pairs of equivalent patterns are shown in Fig. 7.2 represented by layouts { 3, 25 } , { 21 (11, 3) , 28 (15, 3) } , { 33 (17, 3) , 36 (19, 3) } and { 16 (8, 3) , 40 (21, 3) } respectively.

The number of both identical and equivalent patterns is relatively large compared to the total number of layouts produced by the enumerative procedure (Section 7.3.5 of Computational Results). It is therefore, desirable to generate patterns of rectangles which are essentially distinct by eliminating from explicit consideration layouts of rectangles when these lead to identical or equivalent patterns. These duplications can be removed as follows:

Let node n represent a layout ξ of l rectangles in A_0 and r_i be the rectangle chosen by the selection process to be provisionally placed with its b.l.h.c. at position (p_i, q_i) to the current layout ξ producing a new layout of $(l + 1)$ rectangles (ξ' say). Let the provisional layout ξ' be described by List $L_1 (\xi')$.

The algorithm checks whether by provisionally placing rectangle r_i at position (p_i, q_i), a distinct pattern is generated or not. This test is performed by comparing lists L_1 of all patterns corresponding to the same combination of $(l+1)$ rectangles with $L_1 (\xi')$. Clearly, an identical pattern can be easily identified. If $L_1 (\xi')$ represents a duplicated pattern, then rectangle r_i is never added to layout

ξ at position (p_i, q_i).

By implementing the procedure described above, it is possible to considerably reduce the size of the tree generated by the enumerative procedure.

In Chapter 6, we used the concept of " normal cuts " to enhance the formulation of the NGC problem. The same concept is used by the enumerative procedure in the following way. If a rectangle (α, β) is placed with its b.l.h.c. at some position (p, q) in A_0, then in the final cutting pattern there must be some combination of the lengths α_i of the available rectangles whose sum must be exactly p and some combination of the widths β_i of the available rectangles whose sum must be exactly q. This pattern is called " normal " and it is apparent that for any pattern there is a normal equivalent. Normality is a property of a cutting pattern that is relative to the set of pieces available for cutting.

$\tilde{L}$ and $\tilde{W}$ are the normal sets of X and Y - coordinates and have already been defined in Chapter *5*. Sets $\tilde{L}_i$ and $\tilde{W}_i$ have also been defined to include the X and Y- coordinates of locations in A_0 at which rectangle r_i can be placed. It is clear that using the rule of left - most downward placement of rectangles, the enumerative procedure always generates normalised cutting patterns.

A few comments on the mechanics of the search procedure are necessary before the enumerative algorithm is described:

The state of the search at level l in the tree is represented by:

(a) The list L_1 of eight - part labels corresponding to the current layout of l rectangles placed in A_0 so far.

(b) The list L_2 of six - part labels corresponding to the rectangles which have

been added so far to the current layout in A_0 so that an allowable layout of $(l + 1)$ rectangles is produced by placing each one of these rectangles in turn in A_0. List L_2 may include rectangles which, when provisionally placed in A_0 resulted in identical or equivalent patterns. It may also include the same rectangle placed in different allowable positions.

(c) The list L_3 of four - part labels corresponding to the rectangles which have so far been rejected by the algorithm to be placed to the current layout in A_0.

(d) The list S of locations in A_0 available for placement of rectangles in the current layout.

All above lists are updated for forward and backward branchings. Note that forward branching takes place when $|L_1| + |L_2| + |L_3| \leq M$. A diagramatic description of the algorithm is shown in Fig. 7. 7. A detailed description is given in Appendix A.

7.3.4 Computational Results

This Section presents the results obtained when the enumerative algorithm described in Section 7.3.3 was applied to twelve test problems involving up to seven types of pieces in R. The data for the first ten problems is fully presented in Tables 5.1, 5.5 and 5.6 of Chapter 5. The last two problems have been taken from *Beasley* [1985b]; they correspond to problems 7 and 10 in the table of computational results presented in his paper.

The algorithm was coded in FORTRAN and run on a CYBER - 855 computer. Table 7. 2 describes its performance on the twelve randomly generated test problems. It shows the data given for each problem, the value Z_{opt} of the

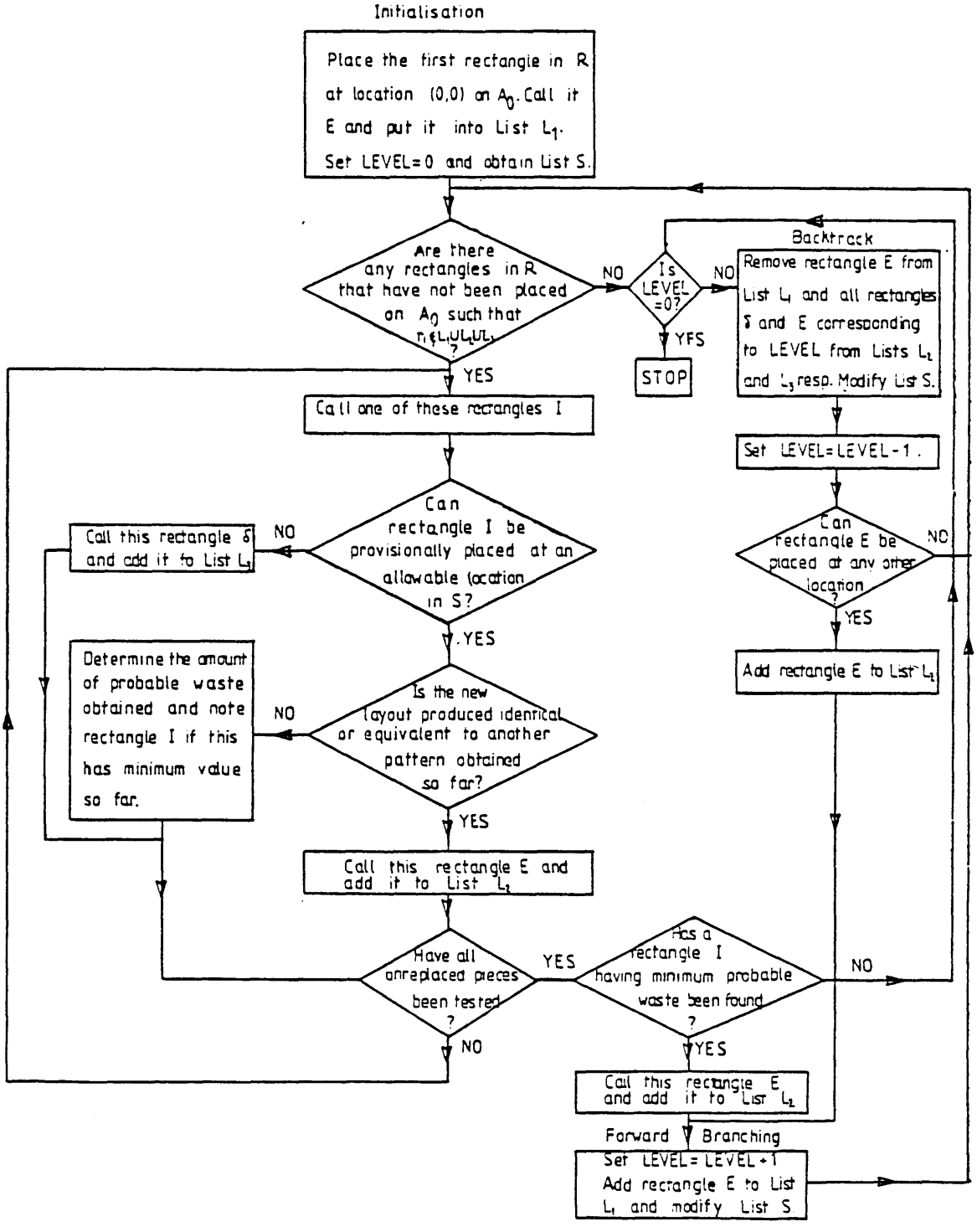


Figure 7.7 Flowchart of the Enumerative Algorithm.

optimum solution, the number of nodes in the tree as well as the number of the essentially distinct patterns generated by the algorithm and finally the total time needed to solve the problem.

The computational effort involved in solving the test problems by the algorithm is described in terms of the number of nodes in the tree - search as well as in terms of computing time. From Table 7. 2 we can see that there is a high correlation between the number of nodes in the tree search and the total time needed to solve each problem. The time needed to solve a problem strongly depends on the number of duplicated patterns obtained for the problem. The number of nodes generated by the tree search for a particular problem is obtained when (i) equivalent patterns are generated in the search and (ii) when such patterns are eliminated from explicit consideration.

The above observation is illustrated by the results obtained when problems 7 and 8 are solved by the tree - search procedure performed in both ways for each problem. In the case of problem 7, the computer program took about 24.8 seconds on the CYBER - 855 to produce a tree of 1606 nodes without testing for the existence of equivalent patterns (identical patterns are not generated) compared to 1017 nodes obtained in 64.7 seconds corresponding to clearly distinct patterns. In the case of problem 8, the same program took about 17.2 seconds on the same computer to obtain 1190 nodes compared to 861 produced, including pattern equivalence testing, in 35.4 seconds. Finally, the computer program took about 52 minutes on the same computer to generate 7679 patterns corresponding to the combination of seven rectangles in a (16, 16) stock plate as presented in Fig. 7. 8 (this problem is Problem No. 2 in Appendix B of this chapter). Almost double the time was required to generate 5678 essentially distinct patterns for this problem.

Problem Number	(α_0, β_0)	m	$\lvert\tilde{L}\rvert$	$\lvert\tilde{W}\rvert$	Optimum solution Z_{opt}	Number of Tree nodes	Number of distinct patterns	Total time in CYBER-855 seconds
1	(4,4)	3	2	2	100	6	4	0.02
2	(6,6)	5	4	6	31	106	59	0.9
3	(10,10)	5	7	7	116	208	122	6.5
4	(20,30)	5	2	3	680	56	35	0.7
5	(7,9)	4	3	9	54	24	12	0.04
6	(8,6)	5	6	4	85	37	21	0.08
7	(10,10)	7	9	10	198	1017	600	64.7
8	(15,10)	7	7	10	262	861	477	35.4
B1	(10,10)	5	8	6	164	203	93	1.7
B4	(15,10)	5	3	10	268	531	282	15.7
B7	(20,20)	5	14	20	430	9	1	0.05
B10	(30,30)	5	9	20	1452	963	470	42.0

Table 7.2 Results of the Enumerative Algorithm.

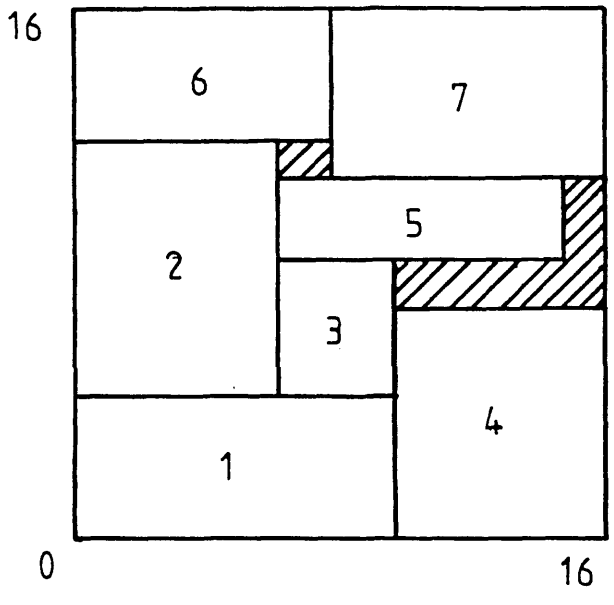


Figure 7.8 A non-guillotine cutting pattern of seven rectangles in A_o
(Problem 2 of Appendix B)

7.3.5 Conclusions

Based on the results presented in the last section we conclude the following:

(i) The procedure for recognising equivalent patterns in most cases takes up about half of the computational time needed to solve a problem. However, this extra cost is outweighed by reducing the tree length by 1/4. (Here we can emphasize the possibility of improving the pattern recognition procedure currently used).

(ii) The computational effort required to solve a problem largely depends on the number of pieces in R (It also depends on the dimensions of the available rectangles relative to the dimensions of A_0 i. e. sizes of normal sets).

It is clear that the enumerative algorithm of Section 7.3.3 can solve optimally small size NGC problems. It will be shown in the following Section how a Branch and Bound Procedure can be used to limit the tree search necessary in order to determine an optimum solution allowing larger NGC problems to be solved.

Problem 6 of Table 7. 2 optimally solved by the algorithm described in Section 7.3.3 is fully worked out in Figures 7. 2 and 7. 3, being used as the Illustrative Example throughout this Chapter.

7.4 The Tree - Search Algorithm

As discussed in Chapter 6, in the event that, the subgradient procedure

together with the problem reduction tests being used in a procedure as described in Section 6.5., does not optimally solve the problem a reduction in the size of the problem will have been achieved. The optimal solution to the reduced problem is then obtained by developing a tree - search algorithm. A description of this algorithm is presented below:

(a) Perform the procedure described in Section 6.5 at the initial tree node effecting a reduction in the size of the problem to be considered.

(b) Select a node of the tree using a specified selection rule and pick an unplaced rectangle associated with this node to branch on . Using the selection rule described in Section 7.3.2 a rectangle r_i is chosen to be placed with its bottom left - hand corner at some position (p, q) in A_0 (the left - most downward placement described in Section 7.3 is used) , such that the pattern of rectangles placed along the branch emanating from the initial tree node to the current node forms a normalized cutting pattern for A_0. We investigated placing rectangle r_i at (p, q) in A_0 by setting variables x_{ip} , y_{iq} to one and

$$\sum_{\substack{r \in \tilde{L}_i \\ r \neq p}} x_{ir} \ , \qquad \sum_{\substack{s \in \tilde{W}_i \\ s \neq q}} y_{is}$$

to zero at the current node. Note that if no tree can be found the tree search is finished.

(c) Perform the subgradient procedure at each tree node - it is clear from the Lagrangean program described in Section 6.2.1 that we can calculate an upper bound for the optimal completion of each tree node as that program is easily adapted to cope with the setting of (x_{ip}, y_{iq}) to specific values in the tree. Each node is then labelled by the minimum upper bound Z_{min} found during the subgradient

iterations for that node.

(d) If a new feasible solution is found at (c) - the conditions for feasibility are given in Section 6.4.1. - update the largest lower bound Z_{LB} (Z_{LB} corresponds to a feasible solution) obtained so far accordingly.

(e) Discard any node in the tree that has an associated upper bound Z_{min} less than Z_{LB} (i. e. backtrack) and go to step (b).

7.4.1 Node Selection Rule

We decided to develop the tree search using a depth - first rather than a breadth - first search strategy allowing us to select the tree nodes along branches extending from the first to lowest levels. Thus, we can regard the tree, starting with the leftmost branch, progressively developing the top to bottom branches and working from left to right, as slowly building up all complete normalised cutting patterns in A_0 with each pattern being generated in the same way as in the enumerative algorithm. By going to the lowest level in the tree as rapidly as possible, although at the expense of temporarily ignoring potentially more promising branches en route, feasible solutions (complete patterns) are generated at early stages in the search. Once a pattern (k) in the Basic Pattern Set (Section 7.3) is generated, its value V_k constitutes a bound on the optimal solution of the problem which can be used to prune the tree and hence reduce the area of search necessary. Furthermore, if computer time is restricted, the best feasible solution (V_k say,) generated to date may be adopted.

A depth - first strategy was chosen for our problem since it was easily implemented computationally (less computer storage and less book-keeping routines

are required than the corresponding amounts involved in the use of a strategy of always branching from the tree node with the current largest upper bound).

7.4.2 Branching Rule

At any tree node, the state of search is represented by Lists L_1, L_2, L_3 and S as they have been defined in Section 7.3.3. Let n be the tree node (excluding the initial tree node) that has been currently picked by the node selection rule to branch on. Let $L_4 = \{ r_i \varepsilon R \mid r_i \notin L_1,\ j_i \neq L_2 \cup L_3 \}$ represent the set of unplaced rectangles associated with node n from which a rectangle r_i must be chosen to branch on . Set L_4 is computed as follows. Consider any rectangle r_i of type j_i in Set R that is not included in the current layout in A_0 represented by L_1. Rectangle r_i is excluded from further consideration if any other rectangle of the same type has been already either placed in allowable positions to the current layout to produce other layouts or found impossible to be placed. Once such a rectangle r_i is found, it is then tested whether by provisionally placing it in all allowable locations $(p, q) \varepsilon S$ available in the current layout, a new layout is produced which is either identical or equivalent to a pattern generated so far. If the new layout is essentially distinct, the corresponding probable waste (Section 7.3.2) is noted and rectangle r_i belongs to Set L_4.

If Set L_4 is empty, then no branching from node n is possible; otherwise a rectangle $r_i \varepsilon L_4$ with minimum probable waste is picked to branch on.

7.4.3 Bound Calculation

Once a rectangle r_i is chosen to be placed with its b. l. h. c. at position (p, q) in A_0 at node n - level l - variables X_{ip} and Y_{iq} are set to one and both summations

$$\sum_{r \in \tilde{L}_i,\ r \neq p} X_{ir} \quad \text{and} \quad \sum_{s \in \tilde{W}_i,\ s \neq q} Y_{iq}$$

to zero in the current Lagrangean solution. The placement of rectangle r_i at (p, q) also requires the setting of

$$\sum_{r=p}^{p+\alpha_i} \sum_{s=q}^{q+\beta_i} Z_{rs}$$

- represents the area at A_0 taken up by rectangle r_i - to zero. The Lagrangean bound (Section 6.2.1.) becomes tighter by imposing necessary conditions on some variables in the Lagrangean solution derived from the amount of " essential waste " associated with the current layout of rectangles on A_0. The amount of " essential waste " is computed as follows:

Let L_1 represent the current layout of l rectangles in A_0 at node n. Since only (M - l) rectangles remain to be placed, the essential waste (W) for this layout may be taken as the sum of the areas of the waste regions corresponding to L_1 into which none of the (M - l) rectangles can fit. Let Ω denote the set of locations (r, s) that lie within these regions of essential waste. Since no rectangle can be placed with its b. l. h. c. at any location (r, s) ε Ω, we can set the corresponding variable Z_{rs} to one in the current Lagrangean solution.

Thus, in the Illustrative Example of Section 7.3, consider layout 10 of rectangles r_1 and r_5 as given in Fig. 7.2., with rectangles r_2, r_3 and r_4

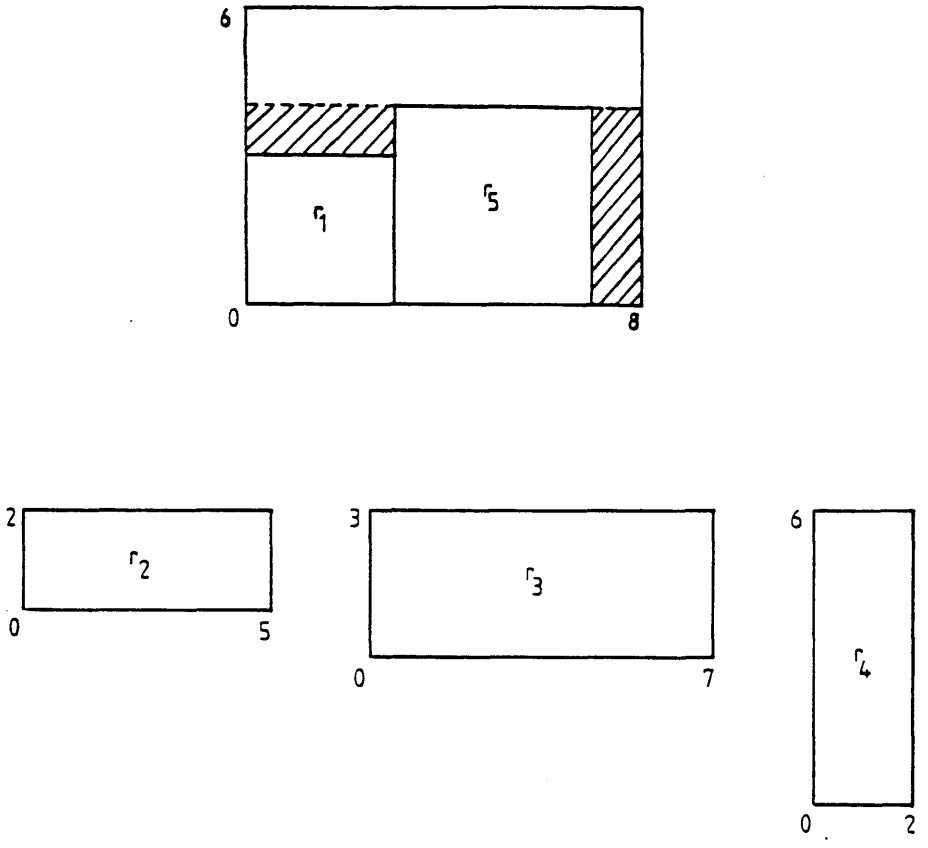


Figure 7.9 Layout of Rectangles r_1 and r_5 (see Fig. 7.2).

remaining to be placed. This layout and the remaining unplaced rectangles are reproduced in Fig. 7.9.

The shaded regions in Fig. 7.9 are essential waste regions, since it is impossible for either of rectangles r_2, r_3 or r_4 to be placed in A_0 so that they cover these regions. The essential waste corresponding to this layout is equal to 3 + 4 = 7, the sum of the areas of these essential waste region and set Ω of locations is given by { (0, 4), (1, 4), (2, 4), (7, 0), (7, 1) , (7, 2), (7, 3) }. Thus, in the Lagrangean program solved at the tree node corresponding to the layout of Fig. 7.9., variables Z_{rs} such that (r, s) ϵ Ω are set to one.

The essential waste associated with a particular node increases as we move from the top to bottom branches in the tree. The essential waste attributed to a pattern of rectangles in the Basic Pattern Set is conventionally taken to be the total waste area of the corresponding pattern i.e

$$W = \alpha_0 \beta_0 - \sum_{r_i \epsilon L_1} \alpha_i \beta_i.$$

Once a tree node is found to be terminal, all X and Y variables associated with the rectangles that are not in the current pattern should be eliminated from the corresponding Lagrangean program. In the case of a non - terminal node, certain X_{ip} and/or Y_{iq} variables associated with any yet unplaced rectangle r_i to the current layout L_1 should also be eliminated if, by placing rectangle r_i with its b. l. h. c. at a location with an X - coordinate equal to p or a Y- coordinate equal to q, it overlaps A_0 or any other rectangle in L_1.

7.4.4 Computational Considerations

In this Section we present the computational implementation of the tree - search algorithm as described in Sections 7.4.1 to 7.4.3.

At the initial tree node, a procedure based on the subgradient procedure and problem reduction tests was carried out in the way described in Section 6.5., so it will not be repeated here. An initial value for Z_{LB} used by the procedure - a value of a feasible solution to the problem - was very easily obtained - it was taken to be equal to the value V_1 of the first complete pattern in the Basic Pattern Set generated by the tree - search.

If no optimal solution was found at the initial node then the set of Lagrangean multipliers that gave the minimum upper bound Z_{min} were recalled --let u*, e* and f* denote this set of multipliers -- to be used in the calculation of the Lagrangean bounds at all nodes emanating from the initial node. Also, the minimum and maximum requirements on the number of rectangles of each type that can be cut from A_0 were noted -- let P_j and Q_j denote these requirements $\forall\ j = 1, ..., m$.

A. The tree-search nodes. Each node n of the tree has associated with it a vector of size four, namely the generation number of its parent node - π (n), the label of the rectangle placed in A_0 at the current node - r_i and the coordinates of the location where rectangle r_i is placed - (p, q).

Each level l of the tree has associated with it three vectors $\underline{u}$ (l), $\underline{e}$ (l) and $\underline{f}$ (l) that contained the best set of u, e and f Lagrangean multipliers (Section 6.2.1) for this level (i. e. the set associated with the lowest upper bound

found at the most recently visited node of level l) and two further vectors, each of size m that contained the minimum and maximum number of rectangles of type j required to be cut by the Lagrangean solution obtained at the most recently visited node of level l. The latter vectors are denoted by $\underline{P}(l)$ and $\underline{Q}(l)$ respectively.

Three vectors that contained information on which variables were set to one or zero or which ones were free at a Lagrangean solution corresponding to any tree node and a further vector that stored all allowable locations in A_0 for placement of rectangles at the current node, were used throughout the search and updated for forward and backward branching accordingly. The three former vectors were denoted by $\underline{X}$, $\underline{Y}$, and $\underline{Z}$ and the latter by $\underline{S}$.

All the above vectors were kept in RAM when a tree node was generated so that they were available if a branch were later to take place from the node .

At each node n of level l the following procedure was carried out:

(a) Reduction. Reduction tests 1 and 2 based on overlapping pieces and free area described in Section 6.3 were used in order to update $Q_j(l)$ for all $j=1, ..., m$. The knapsack area program reduction test 3 was also performed in order to update $P_j(l)$ and $Q_j(l)$ for all j types of rectangles.

(b) Bound. The subgradient procedure as described in Section 6.4.1.was carried out until either 50 iterations were performed or π fell below 0.005. (Rule 2 of Section 6.4.2. was used in the step - size calculation starting with $\pi = 2$). The initial set of Lagrange multipliers used was associated with the tree node at level $(l - 1)$ from which branching was taking place - $\underline{u}(l - 1)$, $\underline{e}(l - 1)$ and $\underline{f}(l - 1)$. Similarly, the initial $\underline{P}(l)$ and $\underline{Q}(l)$ were the final $\underline{P}(l - 1)$ and $\underline{Q}(l - 1)$ at

the predecessor tree node except that $\underline{P}(\iota)$ was updated at each forward branch to take account of the rectangles placed in A_0. The state of the variables in the Lagrangean program was described by vectors $\underline{X}$, $\underline{Y}$ and $\underline{Z}$.

(c) Backtracking. We can backtrack in the tree if any of the following conditions is satisfied:

$$(i) \quad \min \{ \sum_{j=1}^{m} Q_j(\iota)\upsilon_j,\ Z_{min} \} \leq Z_{LB}$$

$$(ii) \quad \sum_{j=1}^{m} (P_j(\iota) + \theta_j)\alpha_j\beta_j > \alpha_0\beta_0$$

where θ_j represents the number of j rectangles included in the layout L_1 corresponding to the current node (Section 7.3.3).

$$(iii) \quad \sum_{j=1}^{m} (P_j(\iota) + \theta_j)\upsilon_j > Z_{UB}$$

7.5 Computational Experience with the Algorithm

The complete tree - search algorithm described in Section 7.4 was programmed in FORTRAN and run on a CYBER - 855 machine for a number of randomly generated problems. The algorithm was tested as a whole on 3 sets of problems that we now describe. Each problem in the first and third sets is denoted

by two numbers. The first number is the set to which the problem belongs, the second one distinguishes the problems within the same set. Thus, 3.2 is the second problem in Set 3.

Sets 1 and 2, containing 8 and 12 problems respectively, have already been used as test problems in Chapters 5 and 6 respectively. The data for all problems in Set 1 is presented in Tables 5.1 and 5.5 of Chapter 5. All problems in Set 2 are from *Beasley* [1985b](each one is denoted by the letter B and a number that distinguishes it within the Set).

Finally, Set 3 contains 18 problems, each randomly generated in the following way. The dimensions α_i and β_i of each rectangle were generated by sampling two integers from the uniform distributions [1, 3 $\alpha_0/4$] and [1, $3\beta_0/4$] respectively. The integer value of each rectangle v_i was generated by multiplying $\alpha_i \beta_i$ by a real random number drawn from the uniform distribution [0. 5, 1. 5] and rounding up. $P_i = 0$ and $Q_i = 1$ for all $i = 1, ..., m$.

Tables 7.3, 7.4 and 7.5 describe the performance of the tree - search algorithm on the 38 test problems of Sets 1, 2 and 3 respectively. Each table shows the size of the stock - rectangle A_0 and the number of rectangles in R to be cut from A_0 for each problem. It also gives the size of the normal Sets $\tilde{L}$ and $\tilde{W}$ being calculated once the original problem has been reduced in size by applying to it the first three reduction tests of Section 6.3. The amount of reduction produced is shown as a reduction percentage $100 (1 - D_2/D_1)$ where D_2 and D_1 are defined in Section 6.6. Furthermore, each table shows the best upper bound Z_{min} and the best lower bound Z_{LB} corresponding to a feasible solution before the algorithm first branches; the value of the optimum solution Z_{opt} and the number of nodes generated in the search and finally, the total time and the time spent at the initial tree

Problem Number	(α_0, β_0)	m	$\lvert\tilde{L}\rvert$	$\lvert\tilde{W}\rvert$	Initial Tree node					Tree-Search		
					Reduction percentage $(1-D_2/D_1)100$	Upper bound Z_{min}	Lower bound Z_{LB}	Duality gap %	Time to obtain Z_{min} (CP secs)	Optimum solution Z_{opt}	Number of tree nodes	Total time CYBER-855 seconds
1.1	(4,4)	3	2	2	100%	100	100	-	0.02	100	-	0.02
1.2	(6,6)	5	4	6	60%	35	27	12.9%	0.4	31	18	1.3
1.3	(10,10)	5	7	7	100%	116	116	-	0.9	116	-	0.9
1.4	(20,30)	5	2	3	100%	680	680	-	0.5	680	-	0.5
1.5	(7,9)	4	1	8	25%	58	51	13.7%	0.5	54	7	0.8
1.6	(8,6)	5	6	4	-	102	85	20%	0.4	85	8	0.8
1.7	(10,10)	7	9	10	-	219	198	10.6%	3.2	198	70	10.1
1.8	(15,10)	7	7	10	-	275	205	5%	3.0	262	48	8.8

a Time limit exceeded

b Best solution found before exceeding time limit

Table 7.3 Tree - Search Algorithm on 8 problems of Chapter 5.

Problem Number	(α_0, β_0)	m	$\lvert\tilde{L}\rvert$	$\lvert\tilde{W}\rvert$	Initial Tree node					Tree-Search		
					Reduction percentage $(1-D_2/D_1)100$	Upper Bound Z_{min}	Lower Bound Z_{LB}	Duality gap %	Time to obtain Z_{min} (CP secs)	Optimum solution Z_{opt}	Number of tree nodes	Total time CYBER-855 seconds
B1	(10,10)	5	7	6	30%	194	164	18.2%	1.6	164	31	4.5
B2	(10,10)	7	10	10	11%	257	230	11.7%	2.8	230	2500	652.8
B3	(10,10)	10	9	10	38%	261	246	5.6%	2.5	247	399	40.5
B4	(15,10)	5	-	-	100%	268	268	-	0.04	268	-	0.04
B5	(15,10)	7	6	10	100%	358	358	-	1.2	358	-	1.2
B6	(15,10)	10	13	10	20%	317	289	9.6%	4.5	289	357	45.0
B7	(20,20)	5	-	-	100%	430	430	-	0.04	430	-	0.04
B8	(20,20)	7	6	20	15%	921	834	10.4%	7.13	834	2400	748.1
B9	(20,20)	10	18	17	100%	924	924	-	5.2	924	-	5.2
B10	(30,30)	5	7	7	100%	1452	1452	-	1.5	1452	-	1.5
B11[a]	(30,30)	7	18	27	13%	1798	1688	6.5%	33.5	1688[b]	343	800
B12[a]	(30,30)	10	27	30	9%	1963	1770	5.2%	96.9	1851[b]	257	800

Table 7.4 Tree - Search Algorithm on 12 problems from literature.

a Time limit exceeded

b Best solution found before exceeding time limit.

Problem Number	(α_0, β_0)	m	$\lvert\tilde{L}\rvert$	$\lvert\tilde{W}\rvert$	Initial Tree node					Tree-Search		
					Reduction percentage $(1-D_2/D_1)100$	Upper Bound Z_{min}	Lower Bound Z_{LB}	Duality gap %	Time to obtain Z_{min} (CP secs)	Optimum solution Z_{opt}	Number of tree nodes	Total time CYBER-855 seconds
3.1	(10,10)	5	7	7	100%	911	911	-	0.7	911	-	0.7
3.2	(10,10)	7	7	8	-	119	83	5.3%	1.5	113	145	12.2
3.3	(10,10)	10	10	8	-	1051	869	12.1%	3.3	937	499	58.0
3.4	(10,10)	15	10	8	100%	1307	1307	-	0.3	1307	-	0.3
3.5	(15,15)	5	7	9	80%	4115	4016	2.4%	3.9	4016	34	9.5
3.6[a]	(15,15)	10	14	13	-	4982	3953	20.6%	12.5	4128[b]	1650	800
3.7	(15,15)	15	14	13	100%	5827	5827	-	3.2	5827	-	3.2
3.8	(20,20)	5	9	10	80%	6914	6771	2.1%	10.5	6771	70	64.1
3.9[a]	(20,20)	10	19	15	-	8262	6517	19.3%	31.9	6924[b]	801	800
3.10[a]	(20,20)	15	19	15	86%	10362	10287	0.7%	31.5	10287[b]	558	800
3.11	(30,30)	5	8	10	100%	31974	31974	-	12.6	31974	-	12.6
3.12	(30,30)	7	18	13	28.5%	1237	1178	5%	41.6	1178	358	531.9
3.13[a]	(30,30)	10	26	14	-	40323	31530	24.2%	86.8	32452[b]	271	800
3.14[a]	(30,30)	15	27	15	20%	1325	1270	4.3%	94.2	1270[b]	262	800
3.15	(40,40)	5	10	10	80%	2522	2401	5%	40.1	2401	74	285.1
3.16[a]	(40,40)	7	18	13	28.5%	2720	2487	9.3%	78.5	2487[b]	181	800
3.17[a]	(40,40)	10	32	14	20%	2738	2517	8.8%	149.6	2517[b]	104	800
3.18	(40,40)	15	35	21	100%	2949	2949	-	65.2	2949	-	65.2

Table 7.5 Tree - Search Algorithm on 18 problems.

a Time limit exceeded

b Best solution found before exceeding time limit.

node. A measure of the gap between the value Z_{min} obtained before first branching and the optimum is also given for each problem.

The 38 test problems solved by the algorithm are of two types: those for which at most one rectangle of each type j is required to be cut from A_0 ($Q_j = 1$ for all $j = 1, ..., m$) and those for which there is an upper bound ($Q_j \leq 3$) on the number of pieces of type j that can be cut. Problems in Sets 1 and 3 are of the first type and problems in Set 2 are of the second type. All problems contain between 30 and 2000 variables with 20 to 5000 constraints approximately.

An overall evaluation of the computational results reveals some interesting features. Two out of the 8 problems in Set 1 , five out of the 12 problems in Set 2 and five out of the 18 problems in Set 3, did not require any branching. Note that convergence to optimality was achieved without the tree - search being required in the case of three of the largest problems being solved, involving 15 types of rectangles each, namely problems 3.4, 3.7 and 3.18 . Out of the 26 remaining problems, optimality is obtained in 17 problems; the other 9 problems could not be solved within the time limit of 13 minutes and 20 seconds. In these problems, the algorithm found good feasible solutions, with a bound on the distance from the optimum.

Thus, the best solutions found for problems B11 and B12, with values 1688 and 1851 respectively, are at most 0.23 and 0.86% worse than the optimum, since < 1692 > and < 1867 > are the best valid upper bounds obtained by the search within the allowed time limit for each problem. Similarly, the best feasible solutions found for problems 3.6, 3.9, 3.10, 3.13, 3.14, 3.15 and 3.16 with the corresponding values shown in Table 7.5 are at most 0.5, 0.6, 0.3, 2.0, 0.6, 1.8 and 5% respectively, worse than the optimum for each

problem.

Tables 7.2 and 7.3 both present the results obtained by solving all 8 problems in Set 1 with the enumerative algorithm (Section 7.3.3.) and the tree - search algorithm respectively. We can see that all these problems were solved by the second algorithm with a considerably less computational effort, in terms of the number of tree nodes, as well as in terms of computing time. In particular, problems 3 and 4 were optimally solved at the initial tree node. In both cases there was a 100% reduction in problem size produced by applying the reduction tests of the tree search algorithm on the original problems. However, the optimum solution for problem 8 was found by the same algorithm in 8.8 seconds by generating 48 tree nodes only, although no reduction on the original problem was possible, compared to 600 nodes obtained by the enumerative search in 64.7 seconds.

The computational effort needed by the tree-search algorithm to solve a problem strongly depends on the gap between the best upper bound obtained at the root node of the tree and the optimum. There is a high positive correlation between the value of this gap (expressed in percentage) and the number of nodes in the search tree. For problem B2, the best bound obtained at the root node was away from the optimum by 11.7% requiring 2500 nodes to be generated in 652 seconds compared to 399 tree nodes obtained in 40 seconds for problem B3 with an initial bound on the solution being at most 5.6% worse than the optimum.

The total time needed to solve a problem strongly depends on the number of variables left after reduction before one has to branch. For instance, for problem 3.15 involving 1666 variables before branching, the search generated only 74 nodes in 285 seconds compared to 145 nodes obtained in 12.2 seconds for problem 3.2 which had 177 variables left for branching (in both problems, the

best upper bound obtained at the initial tree node was about 5% away from the optimum).

All problems in Set 2 have also been solved by *Beasley* [1985b]. Comparing his results with the ones shown in Table 7.4. obtained by our algorithm we conclude that for problems for which m is small (Q_j is also small), Beasley' s results may involve less computational effort. On the contrary, we expect our tree - search algorithm to perform better on problems (Sets 1 and 3) in which the number of types of rectangles in R is larger (e.g. one rectangle for each type available).

Fig. 7.10 gives the complete tree for the Illustrative Example of Section 7.3 (problem 1.6 of Table 7.3) generated by our tree - search algorithm (the tree generated for the same problem by the enumerative algorithm is presented in Fig. 7. 3). In Fig. 7.10, the order in which each node is generated is denoted by the one - digit integer number written outside the node at the top left or right hand side of it. The state of search at each node is represented by three numbers written within each circle. The first of these numbers indicates the label of rectangle that is placed in A_0 at the current node with its b. l. h. c. at the location represented by the following pair of numbers. The best upper bound obtained at each node is denoted by the number written at the bottom left hand side or below each node. This problem was solved with a value of an initial feasible solution equal to 85. No better feasible solution was found by the algorithm, so the initial solution was the optimal one.

In Appendix B, the data and the optimum cutting patterns (both guillotine and non-guillotine) are presented for a number of constrained cutting problems. The computational results of these problems are given in Table 7.6.

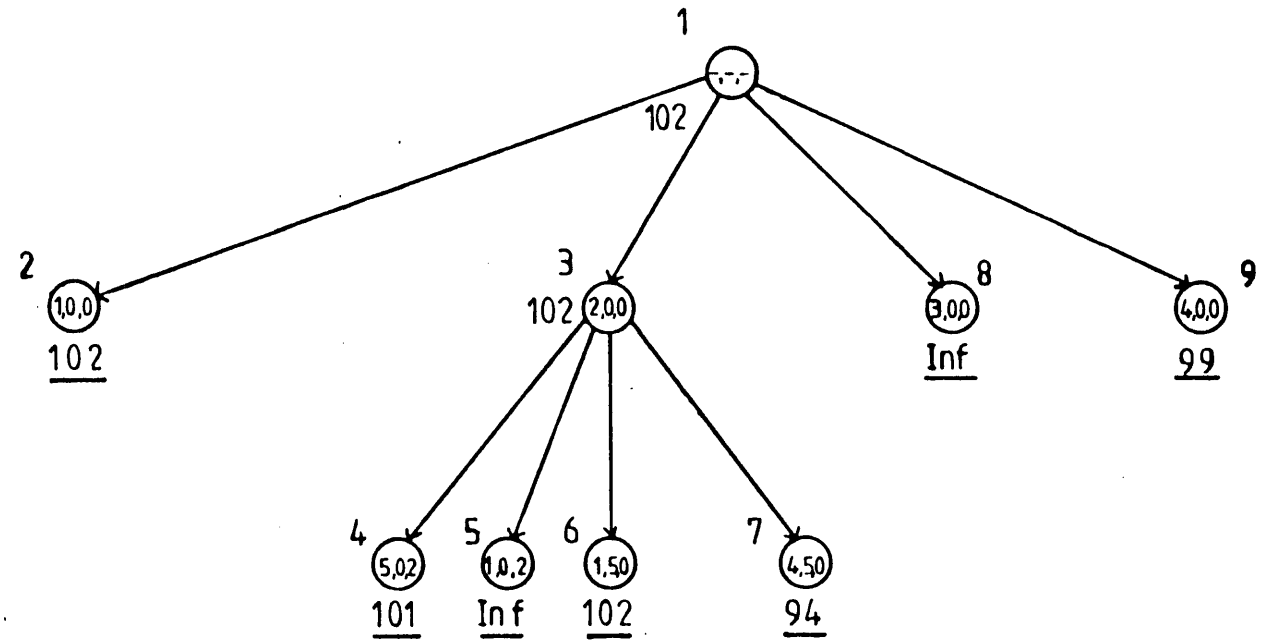


Figure 7.10 The Tree - Search obtained for the Illustrative Example of Section 7.3.

Guillotine Solution							
Problem Number	Initial Tree Node				Tree - Search		
	Upper Bound Z_{min}	Duality Gap (%)	Number of SSA Iters	Time in CYBER-855 CP seconds	Optimum Solution Z_{opt}	Number of Tree Nodes	Total Time in CP secs
1	48	17.0%	20	29.7	41	649	32.1
2	253	10.4%	20	128.3	229	749	134.3
3	370	3.0%	20	577.9	359	5855	593.5
4	675[a]	9.4%	16	1500	617	71000	1850.0
5	853[a]	8.2%	14	1500	788	63700	1911.4

Non - Guillotine Solution								
Problem Number	Initial Tree Node					Tree - Search		
	Reduction Pecentage	Upper Bound Z_{min}	Lower Bound Z_{LB}	Duality Gap (%)	Time in CP secs	Optimum Solution Z_{opt}	Number of Tree Nodes	Total Time CPsecs
1	100%	47	47	-	0.02	47	-	0.02
2	100%	245	245	-	0.03	245	-	0.03
3	100%	360	360	-	0.04	360	-	0.04
4	20%	673	642	4%	23.7	647	2671	1500
5	20%	898	856	4%	33.7	856[b]	728	1500

[a] Time limit at root node [b] Best Solution found before exceeding time limit

Table 7.6 Computational Results of the problems in Appendix B.

7.5.1 Conclusions

In this Chapter, we developed a new exact method for solving the NGC problem. In order to limit the search necessary to find the optimum solution, we embedded in the algorithm bounds, obtained from the Lagrangean relaxation of a 0 - 1 integer programming formulation of the problem (Chapter 5) with the Lagrangean problem being solved by subgradient optimisation (Chapter 6). The algorithm has been tested on a number of randomly generated test problems of small to medium size. The largest problem solved includes 15 types of rectangles (one rectangle for each type) to be cut from a stock - plate of size (40, 40).

As can be seen from the computational experience presented in Section 7.5, the algorithm we adopted is a reasonably reliable, efficient tool for solving medium size NGC - problems, as well as for finding very good approximate solutions to problems that involve a lot of computational effort to be solved exactly.

APPENDIX A

We are given a set $R = \{ (\alpha_j, \beta_j), j = 1, ..., m \}$ of rectangles to be cut from a stock rectangle $A_0 = (\alpha_0, \beta_0)$. Each rectangle j in R is associated with a value v_j and an upper bound Q_j on the number of rectangles of that type required to be cut. Let

$$M = \sum_{j=1}^{m} Q_j$$

denote the total number of rectangles available.

Let $P_j(l)$ and $Q_j(l)$ be the minimum and maximum number of rectangles of type j that can be cut by the most recent Lagrangean solution in level - l - of the tree. Let $\underline{u}(l)$, $\underline{e}(l)$ and $\underline{f}(l)$ represent the set of Lagrangean multipliers associated with the current solution.

The state of the search in the tree is described by the lists $L_1(l)$, $L_2(l)$, $L_3(l)$ and S which are updated for forward and backward branching. Each node n in the tree has a unique parent node denoted by $\pi(n)$. The total number of nodes generated by the search is denoted by " no. of nodes ". The description of the complete tree-search algorithm, including the calculation of bounds is then as follows:

Initialisation

1.1 Set $n = 1, l = 0, L_1(0) = \{\}, L_2(0) = \{\}, L_3(0) = \{\}, S = \{\}, \underline{u}(0) = 0,$ $\underline{e}(0) = 0, \underline{f}(0) = 0$, no. of nodes = 1, $P_j(0) = P_j$ and $Q_j(0) = Q_j$ for all $j = 1, ..., m$.

Calculation of bound at initial tree node

2.1 Perform problem reduction tests 1, 2 and 3 (Section 6.3).

2.2 Calculate normal sets $\tilde{L}$, $\tilde{W}$, $\tilde{L}_j$ and $\tilde{W}_j$ for all types j = 1, ..., m and an initial value for the lower bound Z_{LB}.

2.3 Perform the subgradient procedure (Section 6.4.1) to calculate the value of Z_{min}.

2.4 If $Z_{min} = Z_{LB}$ or

$$\sum_{j=1}^{m} P_j \alpha_j \beta_j > \alpha_0 \beta_0 \quad \text{or} \quad \sum_{j=1}^{m} P_j \upsilon_j > Z_{min},$$

stop; Z_{LB} is the optimal solution.

2.5 Recall the set of multipliers $\underline{u}_{min}$, $\underline{e}_{min}$, and $\underline{f}_{min}$ associated with Z_{min} and update $P_j(0)$ and $Q_j(0)$ accordingly.

2.6 Resolve the Lagrangean problem with $\underline{u}_{min}$, $\underline{e}_{min}$, $\underline{f}_{min}$ and perform problem reduction tests 1 to 7. If no reduction is achieved, go to 2.7; otherwise perform a further 30 subgradient iterations to achieve further reduction. Set $\underline{u}(0) = \underline{u}_{min}$, $\underline{e}(0) = \underline{e}_{min}$, $\underline{f}(0) = \underline{f}_{min}$. Update $P_j(0)$ and $Q_j(0)$ and continue.

2.7 Place the first rectangle in R at location (0, 0) in A_0. Call this rectangle ζ (of type η) and go to 5.1.

Calculation of bound at node -n- in level - l -.

3.1 If $P_\zeta(l) > 0$, set $P_\zeta(l) = P_\zeta(l) - 1$.

3.2 Since rectangle r_ζ is actually placed at location (p, q) in A_0, set the corresponding variables in the Lagrangean solution to fixed values i.e. set $X_{\zeta p'} = 1$, $Y_{\zeta q'} = 1$ and

$$\sum_{p \varepsilon \tilde{L}_\zeta, p \neq p'} X_{\zeta p} = 0, \quad \sum_{q \varepsilon \tilde{W}_\zeta, q \neq q'} Y_{\zeta q} = 0, \quad \sum_{r=p'}^{p'+\alpha_\zeta} \sum_{s=q'}^{q'+\beta_\zeta} Z_{rs} = 0.$$

3.3 Compute set Ω of locations corresponding to the amount of essential waste - ω - obtained at the current node and set $Z_{rs} = 1$ for all $(r, s) \varepsilon \Omega$.

3.4 If

$$w = \alpha_0 \beta_0 - \sum_{r_i \varepsilon L_1(l)} \alpha_i \beta_i$$

then the current node n becomes a terminal node, so set $Q_j(l) = Q_j(l) - \theta_j$ (see Section 7.3.3) if a rectangle r_i (of type j) $\varepsilon L_1(l)$, $Q_j(l) = 0$ otherwise, and go to 3.7.

3.5 Perform problem reduction tests 1 and 2 in order to update $Q_j(l)$ for all $j = 1, ..., m$.

3.6 Eliminate any X_{ir} and/or Y_{is} variables associated with any rectangle $r_i \notin L_1(l)$ from the Lagrangean solution if it is not possible to place r_i at any location (r,q) for all $q \varepsilon \tilde{W}_i$ or location (p, s) for all $p \varepsilon \tilde{L}_i$.

3.7 Perform problem reduction 3 in order to update $P_j(l)$ and $Q_j(l)$ for all $j = 1, ..., m$.

3.8 Set no.of iterations = 50 and obtain $\underline{u}(l) = \underline{u}(l-1)$, $\underline{e}(l) = \underline{e}(l-1)$ and $\underline{f}(l) = \underline{f}(l-1)$.

3.9 Perform the subgradient procedure to calculate the value of Z_{min}.

3.10 If $Z_{min} > Z_{LB}$ and Z_{min} is such that the Lagrangean solution is feasible to the original problem (Section 6.4.1), set $Z_{LB} = Z_{min}$ and go to 6.1; if

$$\min \{ \sum_{j=1}^{m} Q_j(\iota)\, v_j , Z_{min} \} \leq Z_{LB} \text{ or } \sum_{j=1}^{m} P_j(\iota)\, \alpha_j \beta_j > \alpha_0 \beta_0$$

$$\text{or } \sum_{j=1}^{m} P_j(\iota)\, v_j > Z_{min}$$

then go to 6.1; otherwise, recall the set of multipliers $\underline{u}_{min}$, $\underline{e}_{min}$ and $\underline{f}_{min}$ associated with Z_{min}, set $\underline{u}(\iota) = \underline{u}_{min}$, $\underline{e}(\iota) = \underline{e}_{min}$ and $\underline{f}(\iota) = \underline{f}_{min}$, update $Q_j(\iota)$ and $P_j(\iota)$ accordingly and continue.

Selection of a rectangle for placement

4.1 Choose a rectangle r_i from set R such that $r_i \notin L_1 \cup L_2 \cup L_3$ and it has not been tested for placement. Let this rectangle be r_I of type J and go to 4.2. If none exists, check whether a rectangle $r_I \notin L_1 \cup L_2 \cup L_3$ having minimum value of problable waste has been found for actual placement at (p', q') in A_0. Call this rectangle r_ζ (of type η) and go to 5.1; otherwise go to 6.1.

4.2 If

$$\alpha_I \beta_I \leq \alpha_0 \beta_0 - \sum_{r_i \in L_1(\iota)} \alpha_i \beta_i$$

then check if rectangle r_I can be provisionally placed with its b. l. h. c. at any allowable location in S; otherwise go to 4.3. If none of the following conditions is satisfied for a particular (p', q') ε S,

(i) rectangle r_I overlaps boundary of A_0

(ii) rectangle r_I overlaps a rectangle $r_i \varepsilon L_1$

then rectangle r_I can be provisionally placed at (p', q') and go to 4.4;

otherwise continue.

4.3 Set $L_3(l) = L_3(l) \cup \{ r_I, J_I, \alpha_I, \beta_I \}$ and go to 4.1.

4.4 If by provisionally placing rectangle r_I at (p', q') to the current layout of l rectangles represented by $L_1(l)$, a new layout is produced which is identical or equivalent to another pattern obtained so far, then set $L_2(l) = L_2(l) \cup \{ r_I, J_I, p'_I, q'_I, \alpha_I, \beta_I \}$ and go to 4.1;

otherwise determine the amount of " probable waste " generated, note rectangle r_I if this has minimum value so far and go to 4.1.

Forward branching

5.1 Set $L_2(l) = L_2(l) \cup (r_\zeta, \eta_\zeta, p'_\zeta, q'_\zeta, \alpha_\zeta, \beta_\zeta)$.

5.2 Set $l = l + 1$, no.of nodes = no. of nodes + 1, π (no. of nodes) = n, n = no. of nodes and $P_j(l) = P_j(l-1)$, $Q_j(l) = Q_j(l-1)$ for all $j = 1, ..., m$.

5.3 Set $L_1(l) = L_1(l-1) \cup \{ \bar{r}_\zeta, r_\zeta, \eta_\zeta, \theta_\eta, p'_\zeta, q'_\zeta, \alpha_\zeta, \beta_\zeta \}$ and modify S accordingly.

5.4 If $|L_1(l)| = M$ stop (all rectangles in R have been placed in A_0); otherwise go to 3.1.

Backtracking

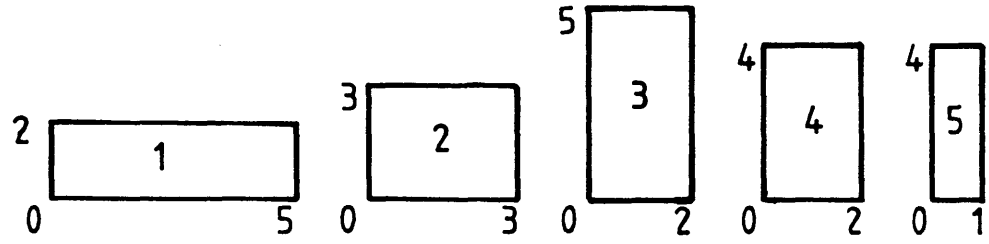
6.1 If $l = 0$, stop; all normal cutting patterns have been generated; the optimum solution is given by the current value of Z_{LB}.

6.2 To remove rectangle r_ζ (of type η) placed at (p', q') in A_0 at node n, level l, set $l = l - 1$, $n = \pi(n)$, $L_1(l) = L_1(l) - \{ \bar{r}_\zeta, r_\zeta, \eta_\zeta, \theta_\eta, p'_\zeta, q'_\zeta, \alpha_\zeta, \beta_\zeta \}$ and modify S.

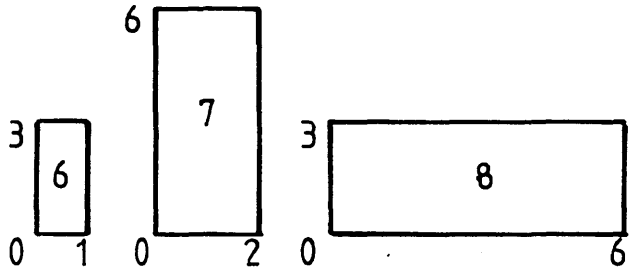
6.3 If rectangle r_ζ can be placed at an allowable location $(r, s) \in S$ such that $r \neq p'$ and $s \neq q'$, set $p' = r$, $q' = s$ and go to 5.1; otherwise go to 4.1.

APPENDIX B

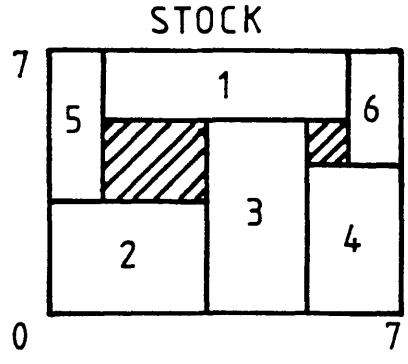
PROBLEM 1



Value:	12	8	12	6	7
Constraint:	1	1	1	1	1

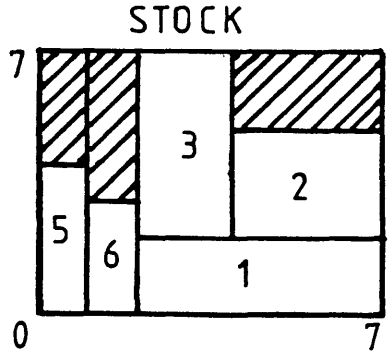


Value:	2	4	5
Constraint:	1	1	1



Optimal non-guillotine cutting pattern

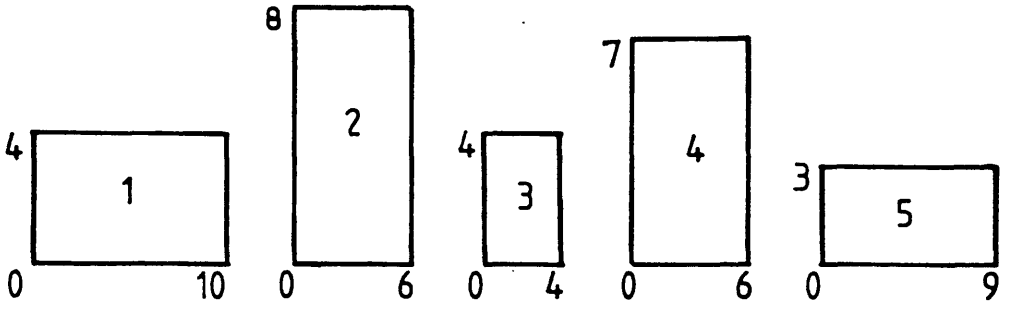
(Opt. Sol. Value=47)



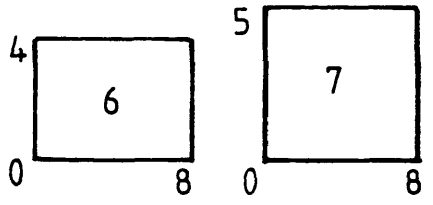
Optimal guillotine cutting pattern

(Opt. Sol. Value=41)

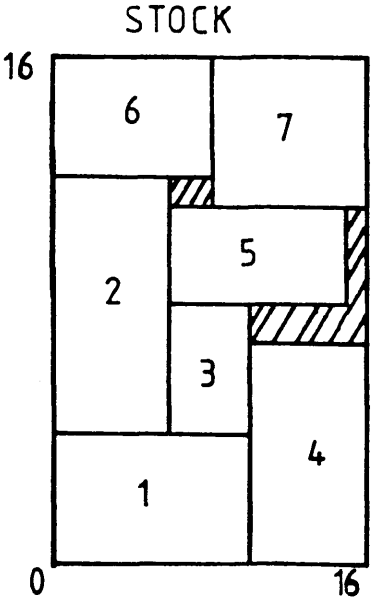
PROBLEM 2



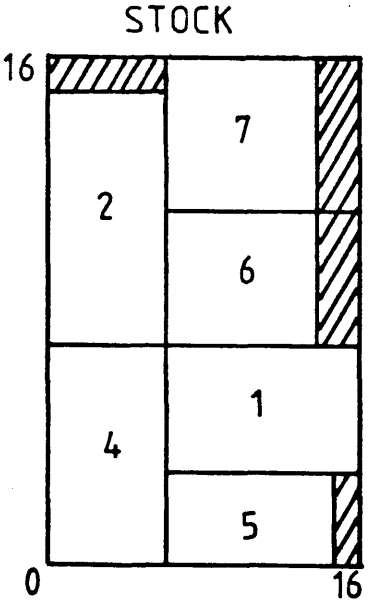
Value:	40	48	16	42	27
Constraint:	1	1	1	1	1



Value:	32	40
Constraint:	1	1

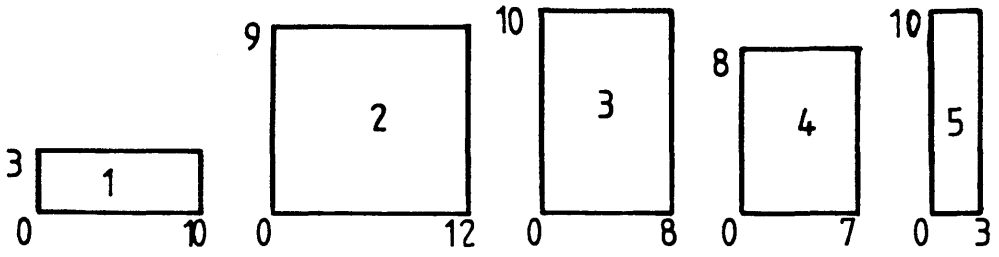


Optimal non-guillotine cutting pattern (Opt. Sol. Value=245)

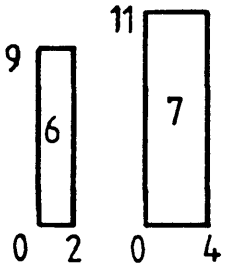


Optimal guillotine cutting pattern (Opt. Sol. Value=229)

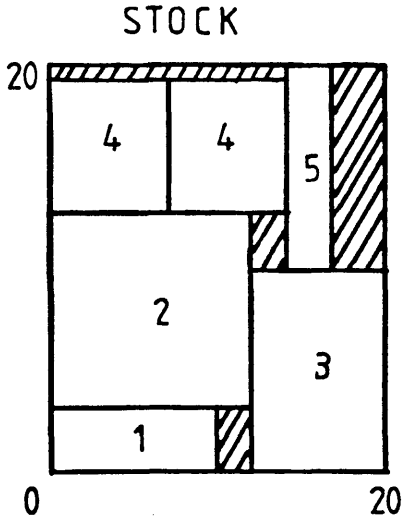
PROBLEM 3



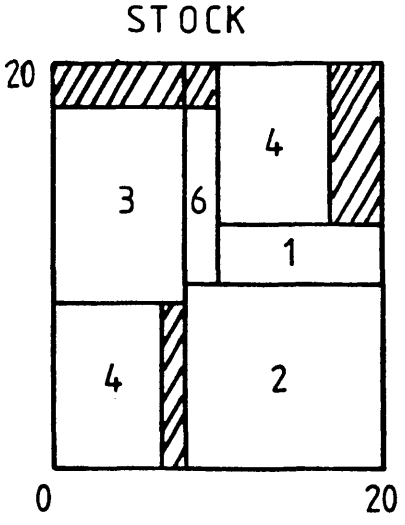
Value:	35	100	85	60	20
Constraint:	1	1	1	2	1



Value:	19	21
Constraint:	1	1

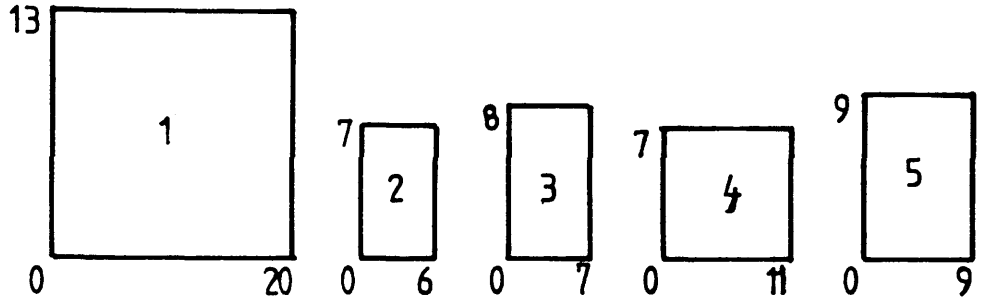


Optimal non-guillotine cutting pattern (Opt. Sol. Value = 360)

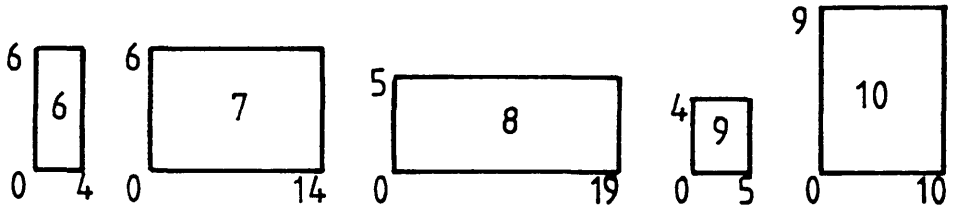


Optimal guillotine cutting pattern (Opt. Sol. Value=359)

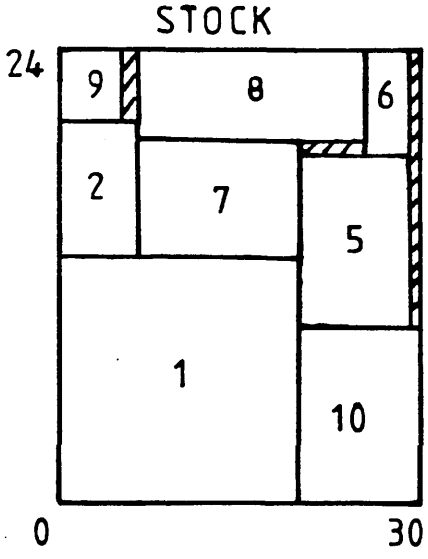
PROBLEM 4



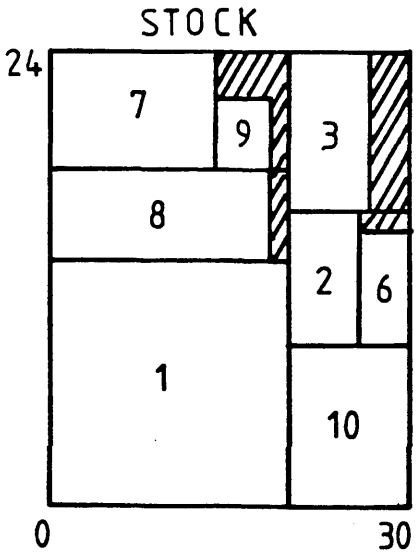
Value:	200	40	60	65	90
Constraint:	1	1	1	1	1



Value:	23	74	95	25	100
Constraint:	1	1	1	1	1

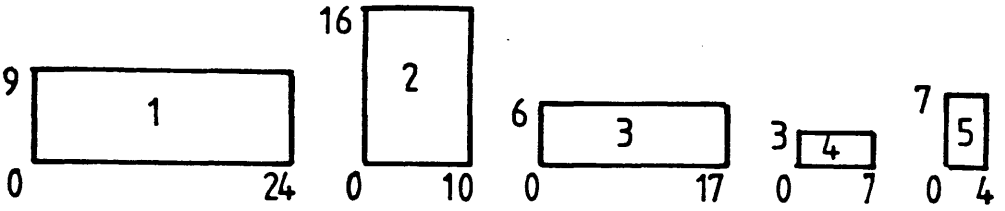


Optimal non-guillotine cutting pattern
(Opt. Sol. Value= 647)

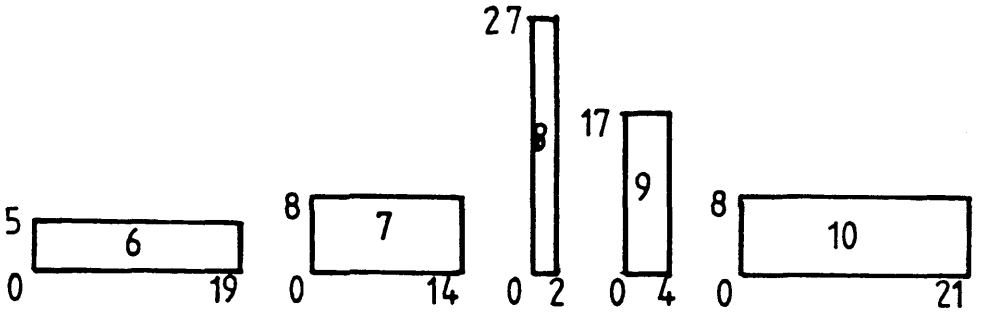


Optimal guillotine cutting pattern
(Opt. Sol. Value =617)

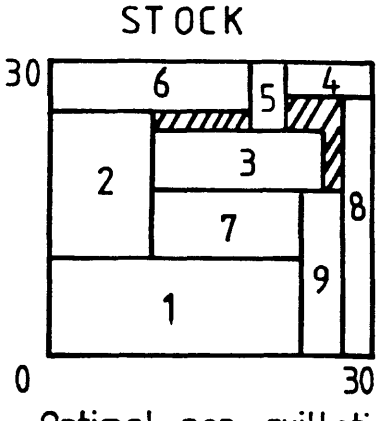
PROBLEM 5



Value:	216	160	102	21	28
Constraint:	1	1	1	1	1

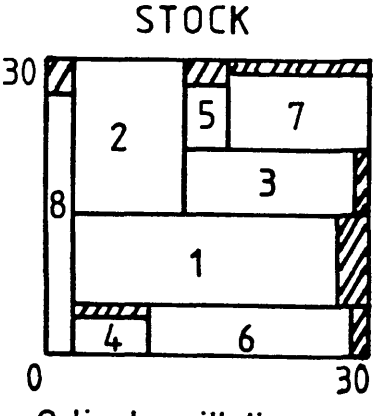


Value:	95	112	54	68	168
Constraint:	1	1	1	1	1



Optimal non-guillotine cutting pattern

(Opt. Sol. Value=856)



Optimal guillotine cutting pattern

(Opt. Sol. Value=788)

CHAPTER 8

CONCLUSION

In this thesis we have considered orthogonal two-dimensional constrained cutting stock problems that are encountered in a variety of industrial applications. Two versions of these problems have been examined, namely guillotine and general cutting problems. The restriction of guillotine cuts is made in the general two-dimensional problem because many important practical situations require this limitation. Problems of glass - cutting, cutting thin sheet metal and paper sheets fall in this category. Problems in which the optimal cutting patterns of rectangles are not restricted to those with the guillotine property e.g. cutting thick metal sheets, are much harder to solve.

New exact algorithmic procedures have been presented for solving both types of cutting problems based on the development of mathematical programming formulations and their exploitation by relaxation techniques to produce bounds for Branch - and - Bound algorithms. The formulation of the Constrained Guillotine Cutting problem as a dynamic program was followed by the examination of a

method to provide upper bounds, namely the State Space relaxation. Subsequently a State Space Ascent procedure was investigated computationally for minimising the resulting upper bounds. This method proved to be efficient in providing high quality bounds for medium - sized problems, which were embedded in a tree - search procedure used to solve these problems exactly. We have shown that CGC problems of practical size can be solved in reasonable computing time using this algorithm. The design of an effective interactive system allowed us to produce solutions to guillotine problems manually using microcomputer graphics. Comparisons were made between manually produced and exact solutions derived from the Branch - and - Bound algorithm.

For solving the Non - Guillotine Cutting problem, two mixed integer programming formulations of the problem were investigated followed by the examination of two methods to provide upper bounds, namely linear programming relaxation and the cutting plane algorithm. Subsequently, five linear problems were investigated; these problems were based on various 0 - 1 integer programming formulations of the NGC problem. Computational experience is available for cutting problems of small size only as a result of the large dimensionality of the NGC integer programs. This led to the investigation of a Lagrangean problem based on the best of the 0 - 1 integer programming formulations of the NGC problem. Subgradient optimisation was used to optimise the resulting upper bounds. Reductions derived from both the original NGC problem and the Lagrangean relaxation produced substantial computational gains in the problem and, in several instances, the optimal solution. The Lagrangean upper bound was incorporated in a depth - first tree - search algorithm used to solve the NGC problem optimally. The effectiveness of such a procedure (measured by the size of the problem it can tackle as well as the running time) depends mainly on the sharpness of the upper bound used, the branching strategy employed and the quality of the

lower bound used. The computational results obtained using the algorithm indicate that it is a reasonably reliable tool for obtaining exact solutions to problems of small to medium size. The largest problem solved includes 15 types of rectangles (one rectangle for each type) to be cut from a stock - plate of size (40, 40). In addition, this algorithm is perfectly capable of providing very good approximate solutions to large problems that involve too great a computational effort to be solved exactly.

REFERENCES

Adamowicz M. and Albano A. [1972] "A Two-Stage Solution of the Cutting Stock Problem", Inf. Processing 71, (Proc. IFIP Congress 71), Amsterdam, North-Holland Publishing Company, pp 1086-1091.

Adamowicz M. and Albano A. [1976a] "Nesting two-dimensional shapes in rectangular modules", Computer Aided Design, Vol 8, pp 27-33.

Adamowicz M. and Albano A. [1976b] " A solution of the rectangular cutting stock problem", IEEE Transactions on Systems, Man and Cybernetics, SMC6, pp 302-310.

Albano A. and Orsini R. [1979] "A heuristic solution of the rectangular stock problem", Computer Journal, Vol 23, pp 338-343.

Art R. C. [1966] "An approach to the Two-Dimensional, Irregular Cutting Stock Problem", IBM Cambridge Scientific Center, Report No. 320-2006.

Baker B., Coffman E. and Rivest R. [1980] "Orthogonal packing in two dimensions", SIAM Journal of Computing, Vol 9, No. 4, pp 846-855.

Baker B., Brown D. and Katseff H. [1981] "A 5/4 algorithm for two-dimensional packing", Journal of Algorithms, Vol 2, pp 348-368.

Baker B. and Schwartz J. [1983] "Shelf algorithms for two-dimensional packing problems", SIAM Journal of Computing, Vol 12, No. 3, pp 508-525.

Beasley J.E. [1985a] "Algorithms for Unconstrained Two-Dimensional Guillotine Cutting", Journal of the Operational Research Society, Vol 36, pp 297-306.

Beasley J.E. [1985b] "An Exact Two-Dimensional Non-Guillotine Cutting Tree-Search Procedure", Operations Research, Vol 33, pp 49-64.

Beckman M. J. [1968] "Dynamic Programming of Economic Decisions", Springer, New York.

Bellman R. E. and Dreyfus S. E. [1962] "Applied Dynamic Programming", Princeton University Press, Princeton, New Jersey.

Biro M. and Boros E. [1984] "Network flows and non-guillotine cutting patterns", European Journal of Operations Research, Vol 16, No. 2, pp 215-221.

Bischoff E. and Dowsland E. B. [1982] "An application of the Micro to Product Design and Distribution", Journal of the Operational Research Society, Vol 33, pp 271-280.

Brown A. R. [1971] "Optimum Packing and Depletion", American Elsevier, New York.

Brown D. J. [1980] "An improved BL bound", Information Processing Letters, Vol 11, No. 1, pp 37-39.

Chambers M. L. and Dyson R. G. [1976] "The Cutting Stock Problem in the flat glass industry - selection of stock sizes", Operational Research Quarterly, Vol 27, pp 949-957.

Christofides N. [1974] "Optimal cutting of two-dimensional rectangular plates", CAD 74 Proc. (Int. Conf. on computers in engineering and building design), IPC Business Press-Microfiche.

Christofides N. and Whitlock C. [1977] "An algorithm for Two-Dimensional Cutting Problems", Operations Research, Vol 25, pp 30-44.

Christofides N., Mingozzi A. and Toth P. [1981a]" State-Space Relaxation Procedure for the computation of Bounds to Routing Problems", Networks, Vol 11, pp 145-164.

Christofides N., Mingozzi A. and Toth P. [1981b]"Exact algorithms for the Vehicle Routing Problem, based on Spanning Tree and Shortest Path Relaxations", Mathematical Programming, Vol 20, pp 255-282.

Chung F. R., Garey M. and Johnson D. [1982] "On packing two-dimensional bins", SIAM Journal of Algor. and Discrete Methods, Vol 3, pp 66-76.

Coffman E. G., Garey M., Johnson D. and Tarjan R. [1980] "Performance bounds for level-oriented two-dimensional packing algorithms", SIAM Journal of Computing, Vol 9, No. 4, pp 808-826.

Coffman E. G., Garey M. R. and Johnson D. S. [1984] "Approximation Algorithms for Bin-packing - An Updated Survey", Bell Laboratories, New Jersey.

Cook S. A. [1971] "The Complexity of theorem proving procedures", Proceedings of the third ACM Symposium on theory of computing, pp 151-158.

Coverdale L. and Wharton F. [1976] "An improved heuristic procedure for a non-linear cutting stock problem", Management Science, Vol 23, pp 78-86.

Dantzig G. B. and Wolfe P. [1960] "The Decomposition Principle for Linear Programs", Operations Research, Vol 8, pp 101-111.

De Cani P. [1978] "A note on the Two-Dimensional Rectangular cutting stock problem", Journal of the Operational Research Society, Vol 29, pp 703-706.

De Cani P. [1979] " Packing Problems in Theory and Practice ", PhD Thesis, Department of Engineering Production, University of Birmingham.

Dowsland K. A. [1982] "Two-dimensional Rectangular Packing" , MSc Thesis, Department of Management Science, University of Wales, Swansea, Wales.

Dyson R. G. and Gregory A. S. [1974] " The cutting stock problem in the flat glass industry", Operational Research Quarterly, Vol 25, pp 41- 53.

Eilon S. [1960] " Optimising the shearing of steel bars", Journal of Mechanical Engineering Science, Vol 2, pp 129 -142.

Eilon S. and Christofides N. [1971] "The loading problem", Management Science, Vol 17, pp 259 - 268.

Eisemann K. [1957] " The Trim Problem", Management Science, Vol 3, No. 3, pp 279 - 284.

Erdos P. and Graham R. L. [1975] "on Packing Squares with Equal Squares",

Journal of Combinatorial Theory, Series (A), Vol 19, pp 119-123.

Farley A. [1983a] "Trim-Loss pattern arrangement and its relevance to the flat-glass industry", European Journal of Operations Research, Vol 14, pp 386-392.

Farley A. [1983b] "A note on modifying a two-dimensional trim-loss algorithm to deal with cutting restrictions", European Journal of Operations Research, Vol 14, pp 393-395.

Fisher M.L. [1973] "Optimal solution of scheduling problems using Lagrangean multipliers", Part I, Operations Research, Vol 21, pp 1114-1127.

Fisher M. L. [1981] " The Lagrangian Relaxation method for solving integer programming problems", Management Science, Vol 27, No. 1, pp 1-18.

Fox B. L., Lenstra J. K., Rinnoy K. and Schrage L. E. [1978] "Branching from the largest upper bound", European Journal of Operations Research, Vol 2, No. 3, pp 191-194.

Garey M., Graham R. L. and Ullman J. D. [1973] "An Analysis of some Packing Algorithms" in Combinatorial Algorithms, edited by R. Rustin, Algorithmic Press, pp 39-48.

Garey M. R. and Johnson D. [1979] " Computers and Intractability : A guide to the Theory of NP completeness ", W. H. Freeman, San Francisco.

Garey M., and Johnson D. [1981] " Approximation algorithms for bin-packing problems - a survey ", pp 147-172 in Ausiello G., Lucertini M., ed. Analysis and Design of Algorithms in Combinatorial Optimisation, Springer Verlag, New York.

Garfinkel R. S. and Nemhauser G. L. [1972] "Integer Programming", John Wiley & Sons.

Geoffrion A. M. [1974] "Lagrangean Relaxation and its uses in integer programming", Mathematical Programming Study, Vol 2, pp82-114.

Ghare P. M. and Walters L. E. [1968] " A Branch and Bound Algorithm for the multidimensional knapsack Problem", presented at a joint meeting of the 33rd

national meeting of the Operations Research Society of America and American meeting of the Institute of Management Science.

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

Gilmore P. C. and Gomory R. E. [1963] " A Linear Programming Approach to the Cutting Stock Problem - part II ", Operations Research, Vol 11, pp 863-888.

Gilmore P. C. and Gomory R. E. [1965] " Multistage cutting stock problems of two and more dimensions", Operations Research, Vol 13, pp 94-120.

Gilmore P. C. and Gomory R. E. [1966] " The Theory and Computation of knapsack Functions ", Operations Research, Vol 14, pp 1045-1074.

Golden B. [1976] " Approaches to the cutting stock problem", AIIE Transactions, pp 265-274.

Gomory R. E. [1963] " An Algorithm for Integer Solutions to Linear Programs ', in Graves R. L. and Wolfe P., eds., Recent Advances in Mathematical Programming, McGraw - Hill, New York.

Greenberg H. and Hegerich R. L. [1970] " A Branch Search Algorithm for the knapsack problem", Management Science, Vol 16, pp 327-332.

Haessler R. W. [1971] " A heuristic programming solution to a non-linear cutting stock problem", Management Science, Vol 17, B-p793-802.

Haessler R. W. [1975] " Controlling cutting pattern changes in one-dimensional trim problems", Operations Research, Vol 23, pp 483-493.

Hahn S. G. [1968] " On the optimal cutting of defective sheets ", Operations Research, Vol 16, pp 1100-1114.

Haims M. J. and Freeman H. [1970] " A multistage solution of the Template-Layout Problem" IEEE Transactions on Systems Science and Cybernetics ", Vol 55C-6, No. 2, pp 145-151.

Held M. and Karp M. [1971] " The Travelling Salesman problem and Minimum Spanning Trees", Part II, Mathematical Programming, Vol 1, pp 6-25.

Held M., Wolfe P. and Crowder H. P. [1974] " Validation of Subgradient Optimisation ", Mathematical Programming, Vol 6, pp 62-88.

Hertz J. [1972] " Recursive computational procedure for two-dimensional stock cutting ", IBM Journal of Research and Development, Vol 16, pp 462-469.

Hinxman A. [1980] " The Trim-loss and assortment problem - a survey ", European Journal of Operations Research, Vol 5, pp 8-18.

Hodgson T. [1982] " A combined approach to the pallet loading problem ", IIE Transactions, Vol 14, No. 3, pp 175-182.

Hodgson T., Hughes D. and Martin-Vega L. [1983] " A note on a combined approach to the pallet loading problem ", IIE Transactions, Vol 15, No. 3, pp 268-271.

Johnson D. [1973] " Near Optimal Bin Packing Algorithms", Doctoral thesis, M.I.T.

Johnson D. [1974] " Fast Algorithms for Bin packing ", Journal of Comput. Systems Sci. ", Vol 8, pp 272-314.

Johnson D., Demers A., Ullman J., Garey M. and Graham R. [1974] " Worst-Case Performance Bounds for Simple One-Dimensional Packing Algorithms ", SIAM Journal of Computing, Vol 3, pp 297-385.

Kantorovich L. V. [1960] " Mathematical methods of organising and planning production ", Management Science, Vol 6, pp 366-422.

Karp R. M. [1972] " Reducibility among combinatorial problems ", Complexity of Computer Computations, R. Miller and J. Thatcher eds, Plenum Press, N. Y., pp 85-104.

Lemke C. E. [1954] "The Dual method of solving the Linear Programming Problem", Naval Research Logistics Quarterly, Vol 1, No. 1.

Madsen O. B. G. [1979] " Glass cutting in a small firm ", Mathematical Programming, Vol 17, pp 85-90.

Marconi R. [1971] "Heuristic method for minimising trim loss in the paper industry", IBM Technical Disclosure Bulletin, Vol 14, pp 325-327.

Marsten R. E. [1981] " The Design of the XMP Linear Programming Library ", ACM Transactions on Mathematical Software ", Vol 7, No. 4, pp 481-497.

Martello S and Toth P. [1979] " The 0-1 Knapsack Problem ", Combinatorial Optimisation, Christofides N., Mingozzi A., Toth P. and Sandi C. editors, John Wiley & Sons, pp 237-279.

Metzger R. W. [1958] " Stock Slitting ", Chapter 8 of Elementary Mathematical Programming, John Wiley & Sons Inc., New York.

Paull A. E. and Walter J. R. [1955] " The Trim Problem : an application of linear programming to the manufacture of newsprint paper ", presented at Annual Meeting of Econometric Society, Montreal, Sept 10-13, 1954, Abstract in Econometrica, Vol 23, p 336.

Paull A. E. [1956] " Linear programming : a key to optimum newsprint production", Pulp Paper Mag. Can. 57, pp 145-150.

Pierce J. F. [1964] " Some Large Scale Production Scheduling Problems in the Paper Industry " (Prentice Hall, Englewood Cliffs, NJ).

Pierce J. F. [1966] " On the solution of Integer Cutting Stock Problems by Combinatorial Programming - Part I ", IBM Cambridge Scientific Center, Report No. 36, Y02.

Sahni S. and Horowitz E. [1979] " Fundamentals of Computer Algorithms ", Pitman, London.

Salkin H. M. and Dekluyver C. A. [1975] " The Knapsack Problem : a survey ", Naval Research Logistics Quarterly, Vol 20, pp 127-144.

Shapiro J. F. [1979] " A survey of Lagrangean techniques for discrete

optimisation", Annals of Discrete Mathematics, Vol 5, pp 113-138.

Short P. J. [1973] " Optimal Batch Execution on a Multi-Processing Computer (A Two-Dimensional Packing Problem) ", MSc thesis, Department of Management Science, Imperial College of Science and Technology, University of London.

Smith A. and De Cani P. [1980] " An Algorithm to Optimise the layout of Boxes in Pallets ", Journal of the Operational Research Society, Vol 31, pp 573-578.

Stainton R. S. [1977] " The Cutting Stock Problem for the stockholder of steel reinforcement bars ", Operations Research Quarterly, Vol 28, pp 139-149.

Steudel H. [1979] " Generating pallet loading patterns - a special case of the two-dimensional cutting stock problem ", Management Science, Vol 25, No. 10, pp 997-1004.

Tilanus C. B. and Gerhardt C. [1976] " An application of cutting stock in the steel industry ", in K. B. Harley (ed.), Operational Research 75 (North -Holland, Amsterdam), pp 669-675.

Trauth C. A. and Woolsey R. E. [1969] " Integer Linear Programming : A study in Computational Efficiency ", Management Science, Vol 15, No. 9, pp 481-493.

Vajda S. [1958] " Trim loss reduction ", Chapter 21 of Readings in Linear Programming, Wiley, New York.

Wagner H. M. and Whitin T. M. [1958] " Dynamic version of the economic lot size model ", Management Science, Vol 5, pp 89-96.

Wang P. Y. [1983] " Two algorithms for constrained two-dimensional cutting - stock problems ", Operations Research, Vol 31, No. 3, pp 573-586.

Weingartner H. M. and Ness D. N. [1967] " Methods for the solution of the Multi-Dimensional 0/1 knapsack problem ", Operations Research, Vol 15, pp 83-103.

White D. J. [1969] " Dynamic Programming ", Oliver and Boyd, Edinburgh.

Wolfe C. S. [1984] " Cutting Plane and Branch and Bound for solving a class of Scheduling Problems ", IIE Transactions (US), Vol 16, pp 50-58.