215

THE USE OF GEOMETRIC INFORMATION IN HEURISTIC OPTIMIZATION

Thesis submitted in accordance with the requirements of the University of Stirling for the degree of Doctor of Philosophy

by

Anthony Ian Hinxman

May 1977

ACKNOWLEDGEMENTS

This thesis owes its birth to Austin Tate, its growth to Robert Ross, and its completion to Brian Boffey. Particular thanks are due to them.

Thanks are also due, for their contributions in various ways, to Professor J.Leech, Dr.P.D.Smith, Dr.R.G.Dyson, Mr.R.Bagnall, members of the artificial intelligence community at Edinburgh University, Pilkington Brothers Ltd., the Science Research Council, and the University of Stirling.

I am grateful for access to computing services at Edinburgh University and the Edinburgh Regional Computing Centre. Further computing facilities were provided by the Liverpool University Computer Laboratory.

This thesis was produced using the GATE text editing system on the Computational and Statistical Science departmental computer system at Liverpool University. Many of the diagrams and tables were prepared by Miss M.Ross. Other secretarial services were provided by Liverpool University.

C O N T E N T S

Sources of diagrams

Figure 2.3.1	Nilsson (1971)	p3
Figure 2.3.2	ibid.	p47
Figure 2.4.4	ibid.	p127
Figure 2.5.1	Lawler and Wood (1966)	p708

Chapter 1 Introduction

1.1 Motivation and structure

The trim-loss, or cutting stock, problem arises whenever material manufactured continuously or in large pieces has to be cut into pieces of sizes ordered by customers. The problem is so to organize the cutting as to minimize the amount of waste (trim-loss) resulting from it.

Brown (1971) remarks that no practical solution method has been found for the generalized 2-dimensional trim-loss problem. This thesis discusses the applicability of heuristic search methods as solution techniques for this and other problems.

Chapter 2 describes three types of combinatorial search method, state-space search, problem reduction, and branch-and-bound. There is a discussion of the ways in which heuristic information can be incorporated into these methods, and descriptions of the versions of the methods used in the work described in succeeding chapters.

In the 1-dimensional trim-loss problem order lengths of some material such as steel bars must be cut from stock lengths held by the supplier. Gilmore and Gomory (1961, 1963) have formulated a mathematical programming solution of this problem, which also arises with the slitting of steel rolls, cutting of metal pipe and slitting of cellophane rolls. Their approach has been developed by Haessler (1971, 1975) who is particularly concerned with problems arising in the paper industry.

In the 1½-dimensional case the material is manufactured as a continuous sheet of constant width and it

is required to minimize the length produced to satisfy orders for rectangular pieces. In the 2-dimensional case the orders are again for rectangular pieces, but here the stock is held as large rectangular sheets. In both cases there may be restrictions as to the way in which the material may be cut; the generalized problem in each case occurs when no such restrictions exist.

The 1½-dimensional problem appears to be easier of solution than the 2-dimensional case since in the latter it is necessary not only to determine the relative positions of the required pieces in a cutting pattern, but also to partition the pieces into sets to be cut from separate stock sheets. A solution method for the easier problem might provide some insight into possible methods of solution of the more difficult. In chapter 3 a state-space search method for the solution of generalized 1½-dimensional problems where the number of pieces in the order list is fairly small and the dimensions are small integers is described.

This method can be developed to solve 2-dimensional problems in which the order list is fairly small and the size of stock sheets variable but affecting the cost of the material. This development is described in chapter 4.

A similarly structured state-space search can be used for finding solutions to optimal network problems. Such searches do not prove the solutions they find to be optimal, so it is of interest also to develop a method for finding solutions to the problems that proves them to be optimal. In chapter 5 the state-space search method is compared with one using branch-and-bound.

It transpires that the characteristics of trim-loss problems change when large numbers of identical pieces are ordered, so a solution method with a different structure is required. Chapter 6 describes a problem reduction method for generalized 2-dimensional problems in which the order lists are large and the dimensions are small integers.

Even when there are restrictions on the way in which the material may be cut, the presence of other constraints may make a mathematical formulation of the 2-dimensional trim-loss problem intractable, so again a heuristic solution method may be desirable. In a problem where there are sequencing constraints on the design of successive cutting patterns, problem reduction is again found to provide a useful solution method. This is described in chapter 7.

Some conclusions about the efficacy and potential of the methods used are drawn in chapter 8. The remainder of the present chapter is concerned with setting the work described in this thesis in the context of other work on the same and related problems.

1.2 The 1½-dimensional trim-loss problem

There has been little work published on the 1½-dimensional trim-loss problem. Abraham, Kirby, and Ng (1976) remark, "the amount of literature dealing with the cutting stock problem is very limited due to the fact that many models, procedures and programming have been developed commercially and hence kept confidential".

The problem they consider is one in which there are severe constraints on the design of cutting patterns and

there are sequencing constraints to which the patterns must conform. Both a generalized linear programming and an heuristic approach are discussed and the conclusion is reached that both produce acceptable production schedules. They also consider the problem of optimization of the width in which the material is produced.

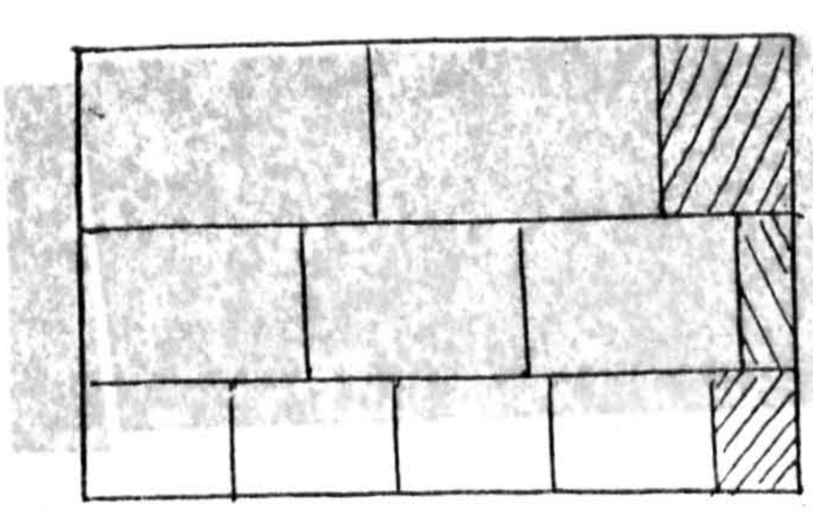
The problem considered in chapter 3 of the present work is one in which there are no restrictions on the way in which the material may be cut. This is reduced to one or more generalized 2-dimensional sub-probems. Pfefferkorn (1975) solves small problems of this type as a specific application of his Design Problem Solver. This he describes as slow, which is to be expected of a program which does not incorporate knowledge specific to the problem. With the size of problem considered here it would seem essential to do so.

There is no evidence that the results obtained in the present work could be improved on in terms of accuracy, and the program runs at an acceptable speed.

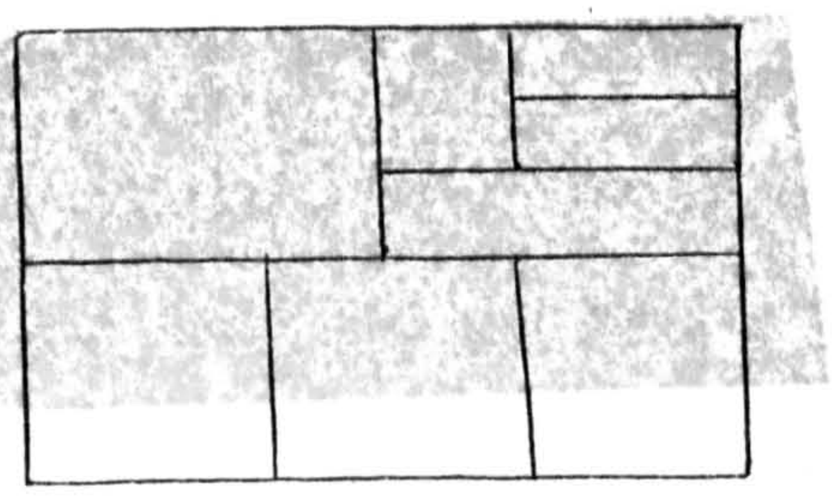
1.3 The 2-dimensional trim-loss problem

With materials such as glass there is a restriction that when they are cut all cuts must be guillotine cuts. For the 2-dimensional trim-loss problem this means that a cut must be a straight line from one side of the sheet to the opposite side, or a similar cut in a sub-sheet resulting from previous such cuts (see figure 1.3.1). The number of stages of cutting may be limited. In two-stage cutting the sheet is cut into strips and the strips into the ordered pieces; in three-stage cutting the strips are cut into sub-strips which are then cut into the ordered

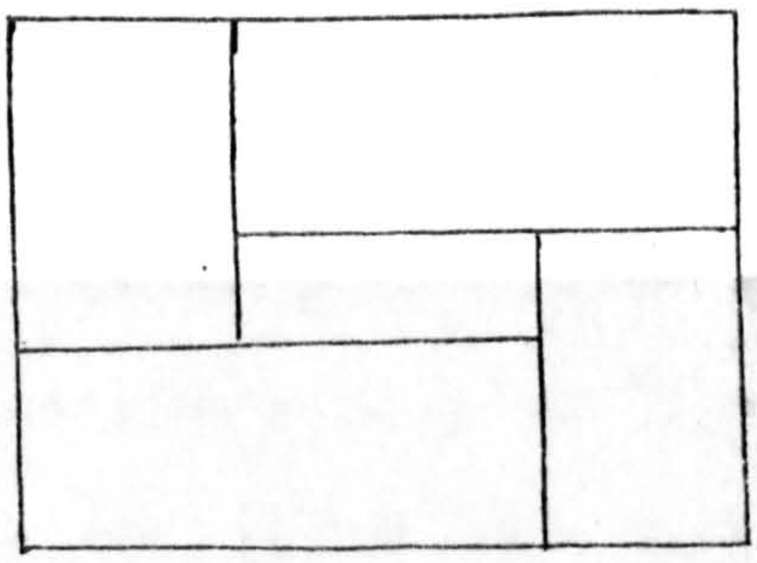
pieces.



two-stage guillotine cutting



multi-stage guillotine cutting



non-guillotine cutting

Types of cutting

Figure 1.3.1

Gilmore and Gomory (1965) give a formulation of the two-stage cutting problem which can be solved by solving two knapsack problems, and a formulation as a staged linear programming problem. They then present applications of their knapsack formulation to a number of other problems. In some circumstances it may be necessary to divide the strips resulting from the first stage cutting into p groups where all the strips in the same group must be cut in the same manner. They discuss the two-stage problems in which p=1 and p=2, three-stage cutting, and problems with many stock sizes and free two-stage cutting. They then consider the three-stage 3-dimensional problem, in which cuboids are cut into layers, the layers into strips and the strips into cuboids, and two-stage problems in which the value of a rectangle depends on its position in the parent rectangle. They finish by presenting a practical application of their

methods to a problem arising in the manufacturing of corrugated boxes.

Hahn (1967) describes a dynamic programming method using an adaptation of an algorithm of Gilmore and Gomory (1966) for minimizing the trim-loss occurring in the three-stage cutting of sheets that include flaws. Escudero and Garbayo (1973) consider a two-stage cutting process. They enumerate all patterns conforming to certain criteria and use mixed integer programming to find a combination of them that optimizes the objective function.

Dyson and Gregory (1974) are concerned with a two-stage problem in which it is desirable that the cutting patterns used should conform to certain sequencing requirements. Their method is to generate a set of cutting patterns using the techniques of Gilmore and Gomory and then to treat the problem of sequencing these in the most desirable way as a travelling salesman problem to which is applied the method of Little, Murty, Sweeney and Karel (1963).

Adamowicz and Albano (1972, 1976), considering the unrestricted problem, group identical pieces into "strips" and then use dynamic programming to arrange these strips into optimal cutting patterns. In their work the pieces have sizes that are small in comparison with the size of the stock sheet.

Haims and Freeman (1970), and Herz (1972) address themselves to a related probem. Here the sizes of pieces to be cut are specified, but the numbers are not. The problem is to find arrangements of pieces that minimize the trim-loss when the stock sheets are cut. Chambers and Dyson

(1976) consider the problem of what size sheets should be held as stock.

None of this work directly indicates a solution method for the 2-dimensional trim-loss problem with varying stock costs discussed in chapter 4. The technique presented there is known to produce sub-optimal solutions, but the degree of sub-optimality appears to be small. In the absence of any other solution methods, such a technique may well be useful.

Preliminary analysis of the abstract 2-dimensional trim-loss problem considered in chapter 6 shows that most of the cutting patterns in the solution will be of the multi-stage guillotine form. The only work in the summary above that might point to a solution method is then that of Adamowicz and Albano.

This method, however, would run into end-effect difficulties on the type of data being considered. Having designed a cutting pattern for one sheet, they eliminate from the order list the pieces cut in that pattern and design the next pattern using the reduced order list. Their choice of pieces to be cut from a sheet is made purely on immediate trim-loss considerations. They do not consider the possibility that certain selections of pieces to be cut in early patterns in a sequence can mean that later patterns must involve more trim-loss than would have been the case if other selections giving the same trim-loss had been made in the early patterns.

During the writing of this thesis, Christofides (1977) published work on the multi-stage guillotine cutting problem. This develops the methods of Gilmore and Gomory

and could be applied to the present problem. Of the work that is described here it can be said that it produces solutions which there is reason to believe cannot be improved upon, and also allows non-guillotine cutting to be specified in the small number of cases where this is necessary for minimal trim-loss solutions to be found.

Dyson and Gregory's problem is similar to the 2-dimensional trim-loss problem with sequencing constraints of chapter 7. Their method is optimal for trim-loss, but by no means fully satisfies the sequencing desiderata. The method developed here considers sequencing requirements during, instead of after, the design of cutting patterns. By this means the sequencing desiderata can be fully satisfied. It is known from consideration of particular cases that it is possible to produce solutions satisfying the sequencing requirements and giving less trim-loss. However the degree of sub-optimality of the present method again appears to be small.

1.4 The optimal network problem

The optimal network problem problem, which is one of minimizing user costs in a communication network subject to a limit on the construction cost of that network, has been thoroughly reviewed by Pearman (1974). The best results in previously published work for the version of the problem considered in chapter 5 are those of Boyce, Farhi and Weischedel (1973), who apply an adaptation of an algorithm of Beale (1970) for selecting optimal subsets.

The branch-and-bound method described here has distinctly better performance. The state-space search method in most cases finds an optimal solution and has

always found one that is nearly optimal. There is some evidence that on problems for which the branch-and-bound method fails to terminate the state-space method is more efficacious in finding good solutions.

Chapter 2 Combinatorial search methods

2.1 Introduction

For a large class of problems it is the case that a solution can be constructed by selecting a finite subset of a finite or denumerably infinite set of elements and arranging them in one of a finite number of ways. Finding a solution can be regarded as searching through the set of alternative combinations until one is found. For such searching to be a practicable proposition it must be suitably organized. In this chapter are described three methods of search organization, all of which can be expressed in graph theoretic terms using trees.

In section 2.2 some graph theoretic terminology is introduced. Section 2.3 describes state-space search, section 2.4 problem reduction and two ways in which problem reduction and state-space search can be combined, and section 2.5 branch-and-bound. Adaptations of the methods for the problems dealt with in the present work are explained, some illustrative examples given, and the relationships between the methods discussed.

2.2 Some concepts in graph theory

A graph G consists of a set V of nodes together with a prescribed set X of unordered pairs of points of V. Each pair $x=\{v_1, v_2\}$ in X is an arc in G. A subgraph S of G is a subset of the nodes of G together with those arcs $\{v_1, v_2\}$ of G for which both v_1 and v_2 belong to S.

A path in G is a sequence of nodes $v_1, v_2, \ldots, v_n$ where the $\{v_i, v_{i+1}\}$, $i=1,2,\ldots,n-1$, are arcs of G. G is said to be connected if for every pair of nodes v_1, v_2 of G there is

a path between v_1 and v_2. If G is not connected then it may be divided into components. A component C of G is a subgraph of G with the property that if v_1 and v_2 are nodes of C then there is a path in G between them and if v_1 is a node of C and v_2 a node of G not belonging to C then there does not exist a path in G between them.

A directed graph D consists of a set V of nodes together with a prescribed set X of ordered pairs of points of V. Each pair $x=(v_1, v_2)$ in X is a directed arc in D.

If D has a directed arc (v_1, v_2), then node v_2 is said to be a successor of node v_1, and node v_1 is said to be a parent of node v_2. A sequence of nodes $v_1, v_2, \ldots, v_n$ with v_i a successor of v_{i-1} for $i=2,\ldots,n$ is called a path of length n-1 from node v_1 to node v_n. If a path exists from node v_1 to node v_2, then node v_2 is said to be a descendant of node v_1 and node v_1 is said to be an ancestor of node v_2.

D may have a cost function P associated with it, $P(v_1, v_2)$ being the cost of (v_1, v_2). The cost of the path $v_1, v_2, \ldots, v_n$ is then $P(v_1, v_2)+P(v_2, v_3)+\ldots+P(v_{n-1}, v_n)$.

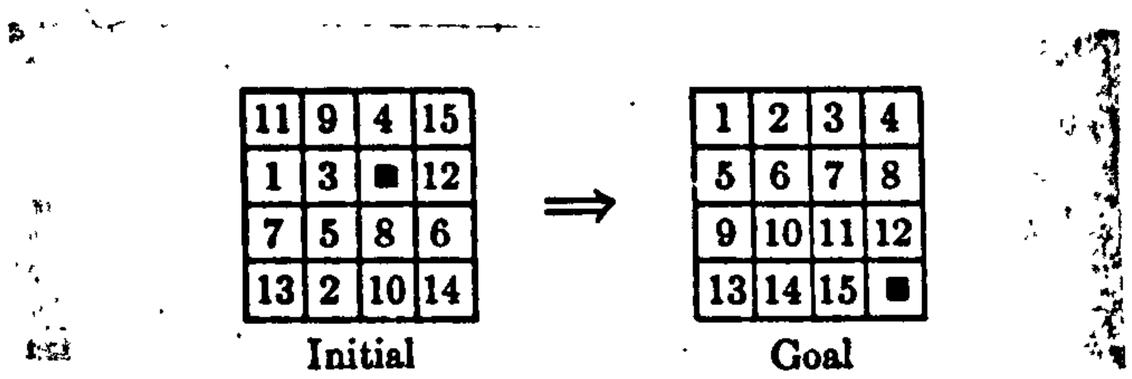
A tree is a directed graph with a distinguished node, called the root node, from which each other node can be reached by one and only one path. A node in a tree which has no successors is called a terminal node. The distance of a node from the root node, i.e. the length of the path to it from the root node, is called its depth in the tree. A binary tree is one in which each non-terminal node has exactly two successors.

2.3 State-space search

2.3.1 Definitions

Consider the 15-puzzle. This consists of 15 numbered,

moveable tiles set in a 4x4 frame. One cell of the frame is always empty, making it possible to move an adjacent numbered tile into the empty cell, leaving the cell from which it came empty. The problem is to find a sequence of moves that will transform a given initial arrangement of the tiles into some other specified arrangement (see figure 2.3.1). Not all such problems are soluble. The arrangements may be divided into two sets with the property that any arrangement in a set can be transformed into any other arrangement in that set and no arrangement in one set can be transformed into an arrangement in the other set (Johnson and Story, 1879; Tait, 1880).



The 15-puzzle.

Figure 2.3.1

In this and many other problems we have a set of configurations and a set of rules describing the possible ways in which one configuration may be transformed into another. From this description can be abstracted the concept of a <u>state-space</u> (Ernst and Newell, 1969).

A <u>state</u> is a configuration in the development of a solution to the problem. An <u>operator</u> is a rule for converting one state into another. The state-space is the set of all possible states and the operators that can be applied to them.

A solution to a problem expressed in these terms will be a sequence of states and the operators that transform

each into the next. So the original problem is converted into that of searching for an appropriate sequence in the state-space.

A significant sub-class of state-space representations for problems is that in which the initial state, the starting point for the solution of the problem, is the empty set and other states are partial solutions. As an example, consider a representation of the travelling salesman problem.

The problem is to plan a trip in a network of n towns such that each town, other than that from which the trip started, is visited once and once only, that the town from which the trip started is visited only after every other town has been visited, and that the trip has minimal length. In the representation the states are ordered lists of towns so far visited. The initial state is the empty list and one state is transformed into another by adding a town to the list of the earlier state. States which are lists satisfying the routing constraints of the problem are candidates as solutions of the problem. That candidate for which the trip distance is least is the required solution.

There is a natural representation for a state-space as a directed graph. The states label nodes of the graph and the operators label arcs. The solution of a 15-puzzle is given by the sequence of labels of arcs forming a path from the initial state to the goal state, the required configuration. The solution to the travelling salesman problem is given in the label of the state representing the solution.

Nilsson (1971) states that for a complete state-space

representation of problem three things must be specified:

i) the form of the state description and, in particular, the description of the initial state,

ii) the set of operators and their effects on state descriptions,

iii) the properties of a goal state configuration.

Observe that for problems such as the travelling salesman, where a solution is required satisfying a number of requirements including an optimality condition, all states satisfying the requirements with the optimality condition ignored are regarded as goal states. They represent <u>feasible solutions</u> to the problem. Amongst their number will be found the <u>optimal solution</u>, which also satisfies the optimality condition.

2.3.2 <u>Computational practice</u>

When a state-space representation of a problem has been established the task of finding a solution to the problem becomes equivalent to the task of finding a path in the state-space graph from an initial state to a goal state. Usually the state-space graph is not specified explicitly. Instead the specification consists of a set $\{s_i\}$ of <u>start nodes</u> and a <u>successor operator</u> Γ that can be applied to any node to give all the successors of that node and the associated arcs. The process of searching through a state-space for a path from an initial state to a goal state then corresponds to making explicit a sufficient portion of an implicit graph to include the required goal node.

State-space search methods can hence be modelled by a graph theoretic process.

i) A start node is associated with the initial state description.

ii) The successors of a node are calculated by applying Γ to the node. This process is called <u>developing</u> the node.

iii) Pointers are set up from each successor back to its parent node. These pointers indicate a path back to the start node when a goal node is finally found.

iv) The successor nodes are checked to see if they are goal nodes. If a goal node has not yet been found, the process of developing nodes continues. When a goal node is found, the pointers are traced back to produce a solution path.

The model as stated is over-simplified. One complication arises from the fact that the state-space graph is not usually a tree. This means that nodes labelled with the same state can be generated by the application of Γ to different nodes. A second complication arises when the search is for an optimal, rather than simply a feasible, solution. Goal states must be generated until the optimal solution has been produced. It is necessary to determine when this has occurred. A third complication is that a decision must be made as to the order in which the nodes are developed.

The third of these points will be considered first, the other two being ignored for the moment. The state-space is a tree and any feasible solution is acceptable. Such a space may be searched by a method which takes no account of any information as to whereabouts in the search space the solution is likely to

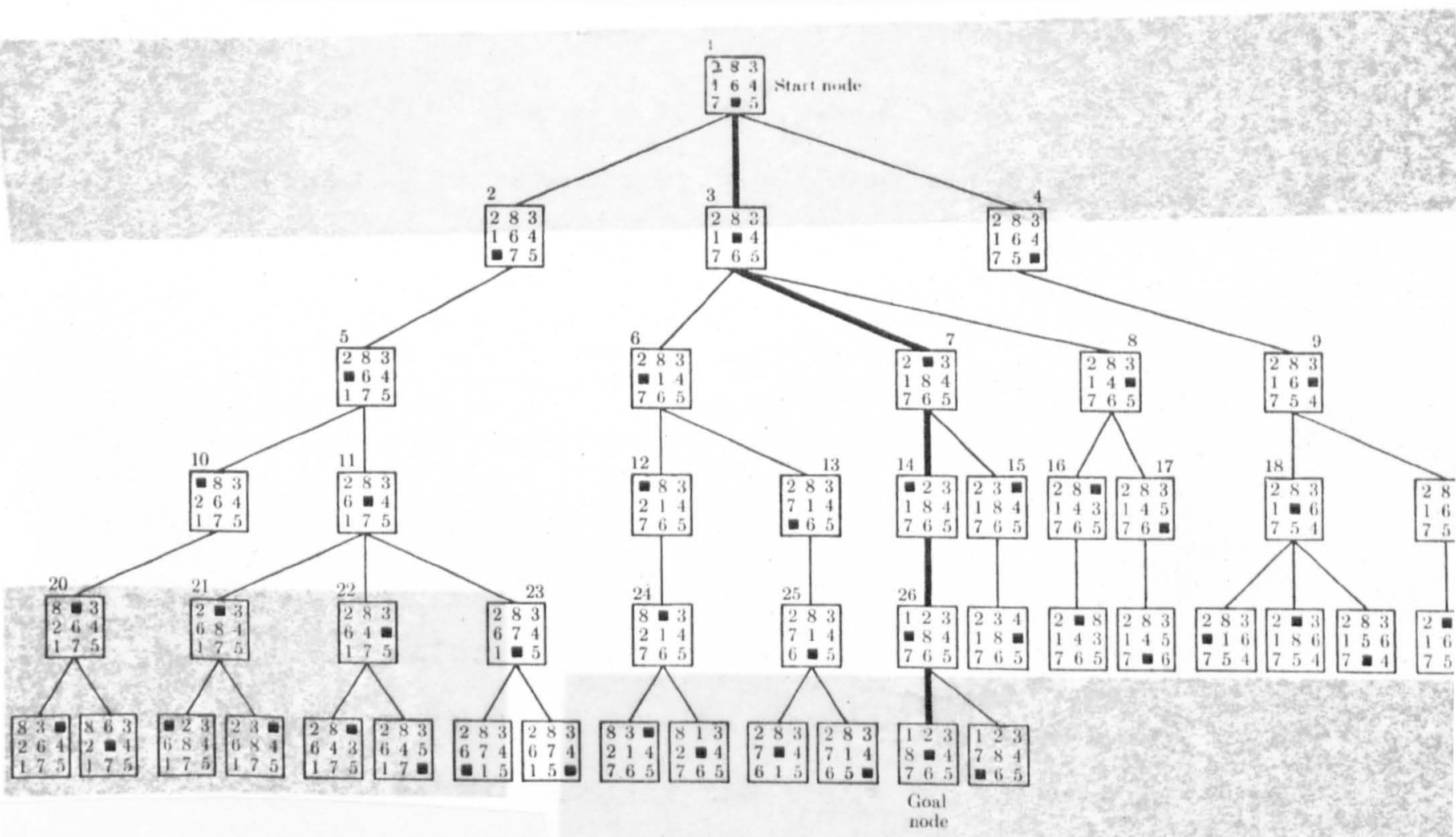
be found. Such methods are called blind-search procedures. A typical blind-search procedure is breadth-first search. Its steps are:

i) Put the start node on a list called OPEN.

ii) If OPEN is empty, exit with failure, otherwise continue.

iii) Remove the first node from OPEN and put it on a list called CLOSED; call this node n.

iv) Develop node n, generating all of its successors. If there are no successors, go immediately to (ii). Put the successors at the end of OPEN and provide pointers from these successors back to n.



The tree produced by a breadth-first search.

Figure 2.3.2

v) If any of the successors are goal nodes, exit with the solution obtained by tracing back through the

pointers; otherwise go to (ii).

Breadth-first search will find the path containing the minimal number of arcs if any solution exists. If no path exists it will exit with failure for finite graphs and will never terminate for infinite graphs. Figure 2.3.2 shows a breadth-first search for an 8-puzzle problem.

The tree generated by a search process is a fragment of the entire state-space and is known as the search tree. A state-space in which a search is being conducted may be referred to as a search space. Search spaces are often extremely large, and a blind search tends to cover a large proportion of a space, producing a large search tree.

Heuristic search methods, such as the Graph Traverser of Doran and Michie (1966), attempt to reduce the size of the search tree generated by using information about the particular space in which the search is being conducted. Such problem-dependent information is used to decide the order in which nodes of the search tree should be developed.

In the Graph Traverser and related methods the heuristic information is used to construct an evaluation function which takes as its argument a state description and produces as a result the value of the state. This value is an estimate of the cost of constructing a path from the present state to the goal state. The values can be used to order the undeveloped nodes of the search tree. Thus instead of developing the first node on OPEN as in breadth-first search, each iteration of the

heuristic search develops the undeveloped node with lowest value.

Consider again the case where the state-space graph is not a tree. In this case the development of a node may result in the generation of a node labelled by a state which also labels a node that has been generated in some previous development. In other words, a development can result in duplicated state labels. In a blind search for a feasible solution the only addition necessary to the basic search algorithm is not to add to OPEN any node whose generation brought about a duplication of state labels. If the search is for an optimal solution then consideration must be taken of the possibility that the cost (the quantity which is tested in the determination of optimality) of a goal state may be a function of the path to it being considered. Now when duplicated state labels arise the paths to the relevant nodes are considered. One of these paths taken together with a path from the duplicated state to a goal state will result in a better cost than the other. If the node just generated is on this path it is put on OPEN and the other node is put on CLOSED, otherwise the node just generated is put on CLOSED.

Similar considerations arise if an heuristic search is being conducted. Here the differing paths to nodes labelled with the same state may be associated with differing values given to this state by the evaluation function. The node for which the value is least is the one that will be retained as a candidate for future development.

Thus the steps of an heuristic search of a state-space graph are:

i) Put the start node, s, on a list called OPEN and compute f(s), where f is the evaluation function.

ii) If OPEN is empty, exit with failure; otherwise continue.

iii) Remove from OPEN that node, n, for which f(n) is smallest and put it on a list called CLOSED. (Resolve ties for minimal values arbitrarily, but always in favour of any goal node).

iv) If n is a goal node, exit with the solution path obtained by tracing back through the pointers; otherwise continue.

v) Develop node n, generating all of its successors. If there are no successors, go to (ii). For each successor, n_i, compute $f(n_i)$.

vi) Associate with the successors not already on either OPEN or CLOSED the values just computed. Put these nodes on OPEN and direct pointers from them back to n.

vii) Associate with those successors that were already on OPEN or CLOSED the smaller of the values just computed and their previous values. Put on OPEN those successors on CLOSED whose values were lowered, and redirect to n the pointers from all nodes whose values were lowered.

viii) Go to (ii).

The problem of determining when an optimal solution has been found in the case where several feasible solutions exist has been considered by Hart, Nilsson and

Raphael (1968). They give an algorithm (A*) of the Graph Traverser type which, by requiring the heuristic evaluation function to satisfy certain conditions, ensures that the first feasible solution found is optimal. Harris (1973) describes a "bandwidth" search which by a different restriction of the evaluation function ensures that the first feasible solution found differs in cost from the optimal solution by a bounded amount. Pohl (1973), in his Heuristic Path Algorithm, uses dynamic weighting of the heuristic information and guarantees the production of a near optimal solution.

2.3.3 More sophisticated search methods

The branching ratio of a graph is the average number of successors possessed by a node of the graph. If a search is being conducted in a graph with a high branching ratio, development of a small number of nodes will result in the generation of a large number of nodes. This causes difficulty in terms of the use of a computer since a large amount of storage would be required to hold representations of all the nodes.

A possible strategy in such a case is the partial development of nodes (Michie, 1967). When a node is first selected for development a subset of the available operators is applied to it and it remains a candidate for development at some future time. If it is later again selected for development a subset of the available operators not already used are applied to it. It will remain a candidate for development until all possible operators have been applied to it.

Michie (1967) also considers the possibility of

synthesizing compound operators from simple ones. The application of a compound operator to a state will generate a sequence of successors corresponding to the successive application of the simple operators of which it is composed, each to the state generated by the previous one. Attempts to make use of this technique encounter the problem that a large number of compound operators could be constructed and it is difficult to determine which of them will in practice be useful. In the same paper, consideration is given to the use of information about a state to select the operator to be applied to the state. This idea can be extended to that of the ordering of operators (Michie and Ross, 1969) to determine which shall be applied during successive partial developments.

Means-end analysis (Ernst and Newell, 1969) hypothesizes that certain states will occur on the path between the present state and a goal state and then attempts to construct the path fragments linking them. It is of use when such hypotheses can be made using information about the current state, but this is not possible in the work described in this thesis.

2.3.4 Ordered operator search

Where state-space search has been used in the present work satisfactory evaluation functions have not been available. Instead a search method, which will be referred to as ordered operator search, in which the heuristic guidance for the search is provided by an ordering of operators, is used. When a node is generated the set of operators that may be applied to it is

ordered, that operator which it is thought most likely will, when applied, produce a node on the desired solution path being placed first, and the ordering being in decreasing order of likelihood. When a node is selected for development the first operator in its list of operators is applied and removed from that list.

The value of a node is thus no longer a simple measure of the probability that the node lies on the path to the desired solution, but rather of the probability that the operator, if any, which will generate the successor node on the path from this node to the desired solution, remains to be applied. A simple mechanism has been found adequate for the generation of such values. When a node is first generated it is given value 0. Whenever an operator is applied to a node an increment is added to the value of that node. The size of the increment may be related to the operator applied. A large value will reflect the belief that the application of operators not yet applied is unlikely to produce a node on a solution path. This mechanism is similar to that of local smoothing (Michie and Ross, 1969) where the initial value assigned to a state by an evaluation function is adjusted in the light of information gained during the development of that state.

A limitation of the use of operator ordering instead of an evaluation function for the guidance of the search is the absence of a means of determining that an optimal solution has been found, when such is wanted. It is now necessary to terminate the search after some arbitrary number of search steps which experience suggests will be

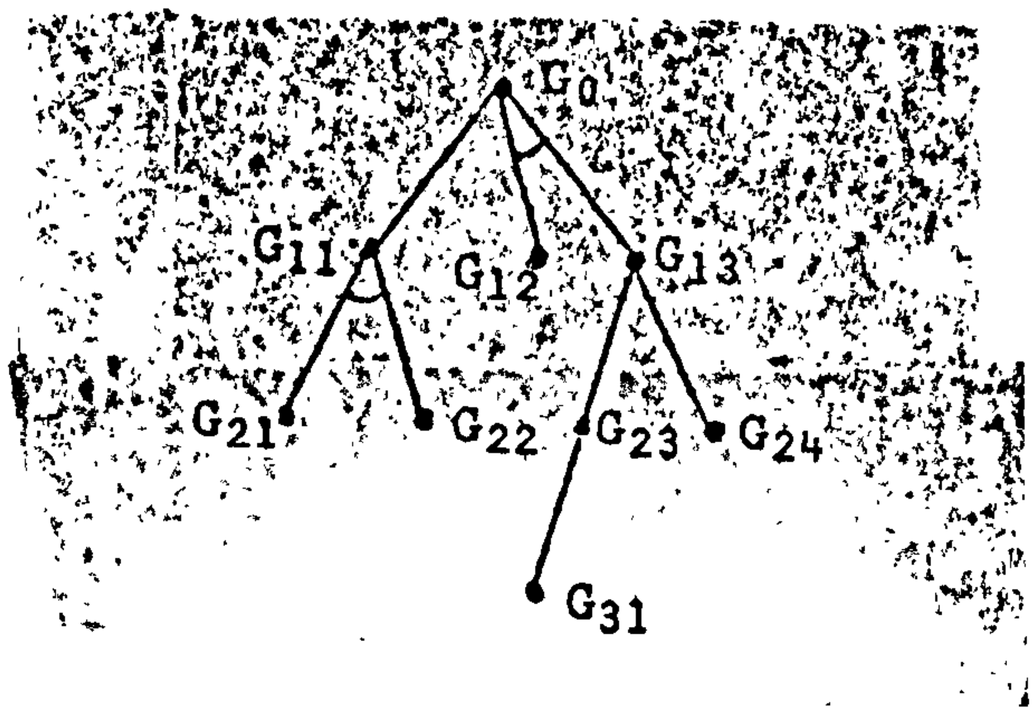
sufficient to include a solution which, if not optimal, is close enough to optimal for the purpose for which it is needed. A similar problem arises in deciding on the point at which a search which has not yet found any feasible solution should be abandoned on the ground that if a solution has not yet been found it is unlikely that one exists at all.

2.4 Problem reduction

2.4.1 AND/OR graphs

Suppose a solution is required to a complex problem. It may be possible to break the problem into component parts whose solutions taken together provide a solution to the original problem. More formally, a goal, G, can be achieved by the achievement of a conjunct of subgoals $G_1, G_2, \ldots$. The assumption will be made throughout the present discussion that the subgoals are independent, that is, that the way in which one is achieved does not affect the possibility of achieving another. If this is not the case then a different approach is needed, see, for example, the work of Sacerdoti (1975).

It may be the case that a subgoal can be split into further subgoals. A subgoal that cannot be divided in this way is called a primitive subgoal. It may also be the case that there is more than one way of achieving a subgoal. For example the achievement of G_2 may require the achievement of either subgoals G_{21} and G_{22} or subgoals G_{23} and G_{24}.



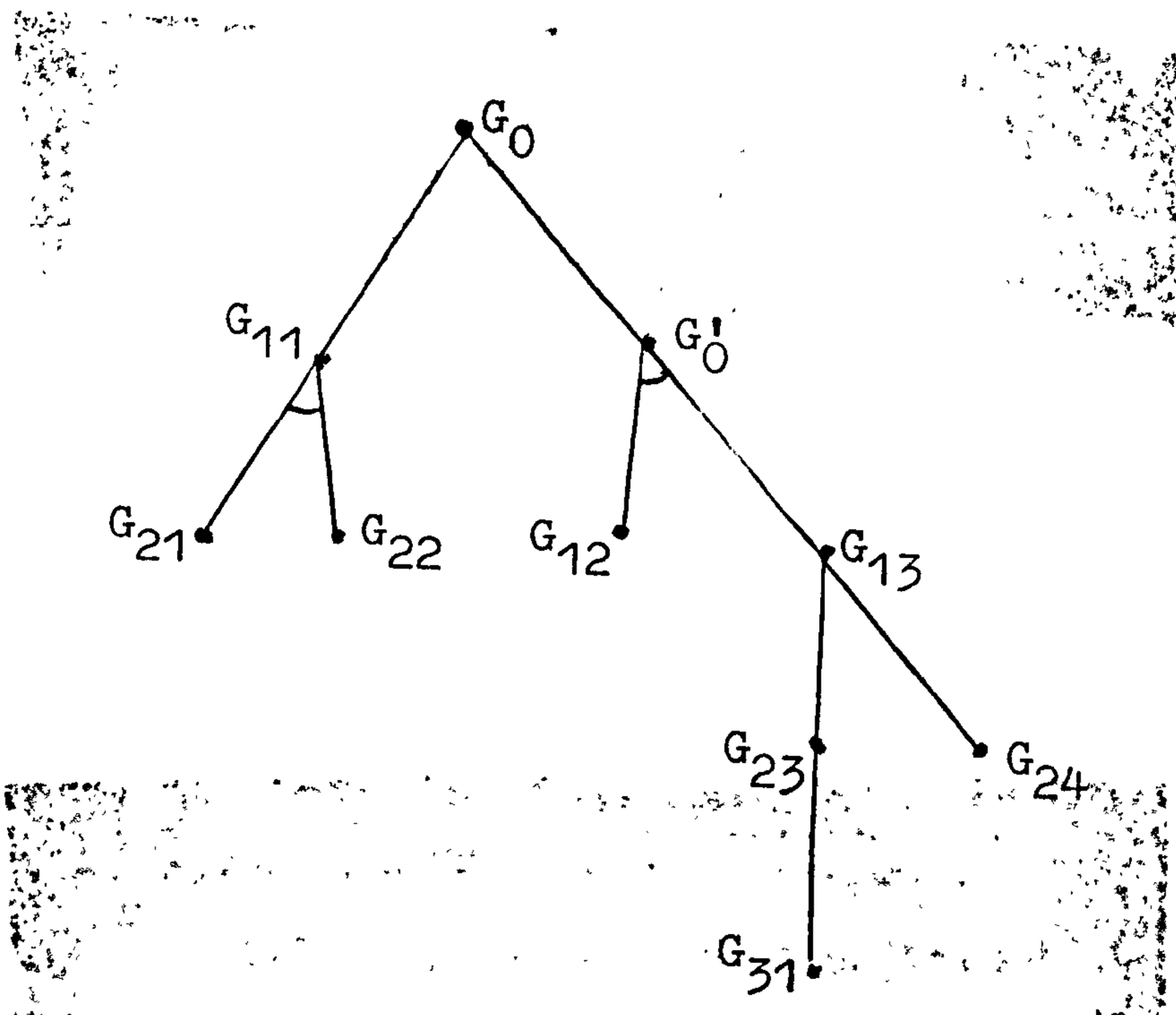
Graphical representation of goal structure

Figure 2.4.1

This sort of situation may be represented graphically. The goal is represented by the root node of a tree; other nodes of the tree are labelled with subgoals, successor nodes being labelled with subgoals of the subgoal which labels their parent node. The requirement that all of a set of subgoals must be achieved for the achievement of the subgoal of which they are components is indicated by a brace across the arcs connecting the nodes they label to their parent. Such a graph is shown in figure 2.4.1. In this case G_0 can be achieved by the achievement of G_{11} or of both G_{12} and G_{13}. The achievement of G_{11} requires the solution of both G_{21} and G_{22}, whilst the achievement of G_{13} requires the solution of either G_{23} or G_{24}. The achievement of G_{23} requires the achievement of G_{31}.

Such graphs may be converted into a canonical form in which the set of arcs originating from a node either consists of one braced group or contains no braced groups. In the first case we have an AND node; all the subgoals of the subgoal labelling the node must be achieved for that subgoal to be achieved. In the second case we have an OR node; the achievement of any one of the subgoals of the

subgoal labelling the node will enable that subgoal to be achieved. Graphs of this canonical form are called AND/OR graphs (Slagle, 1963). Figure 2.4.2 shows the AND/OR form of the graph in figure 2.4.1.



AND/OR form of figure 2.4.1

Figure 2.4.2

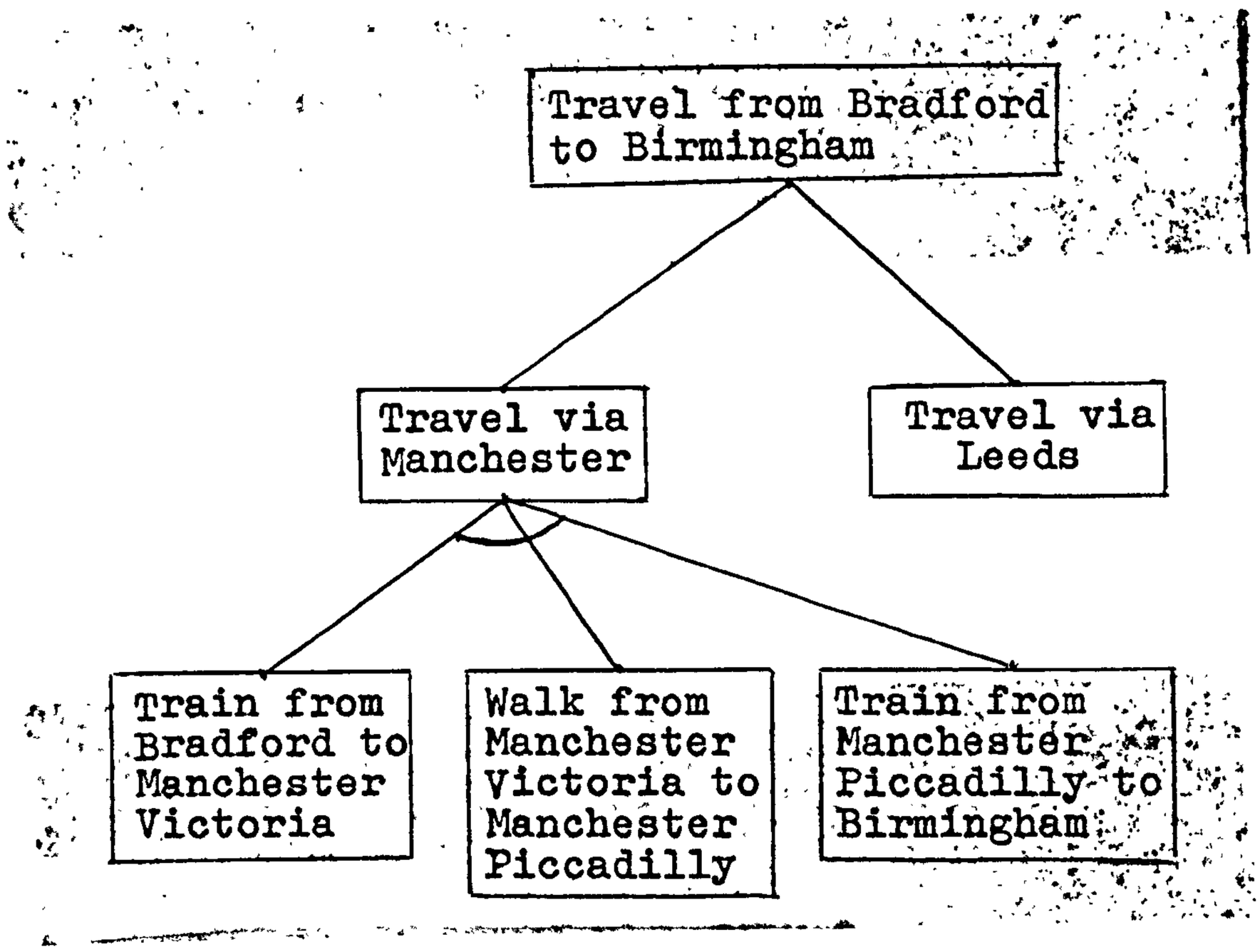
2.4.2 Construction of a solution

The terminal nodes of the AND/OR tree representing the relationships of the subgoals for the solution of a problem are labelled with primitive subgoals. When a primitive subgoal is considered, it will either be the case that it can be achieved, in which case the node it labels is soluble, or that it cannot be achieved, in which case the node is insoluble. Recursively it can be determined whether a non-terminal node is soluble:

i) If the node is an OR node, then it is soluble if and

only if at least one of its successors is soluble,

ii) If the node is an AND node, then it is soluble if and only if all its successors are soluble.



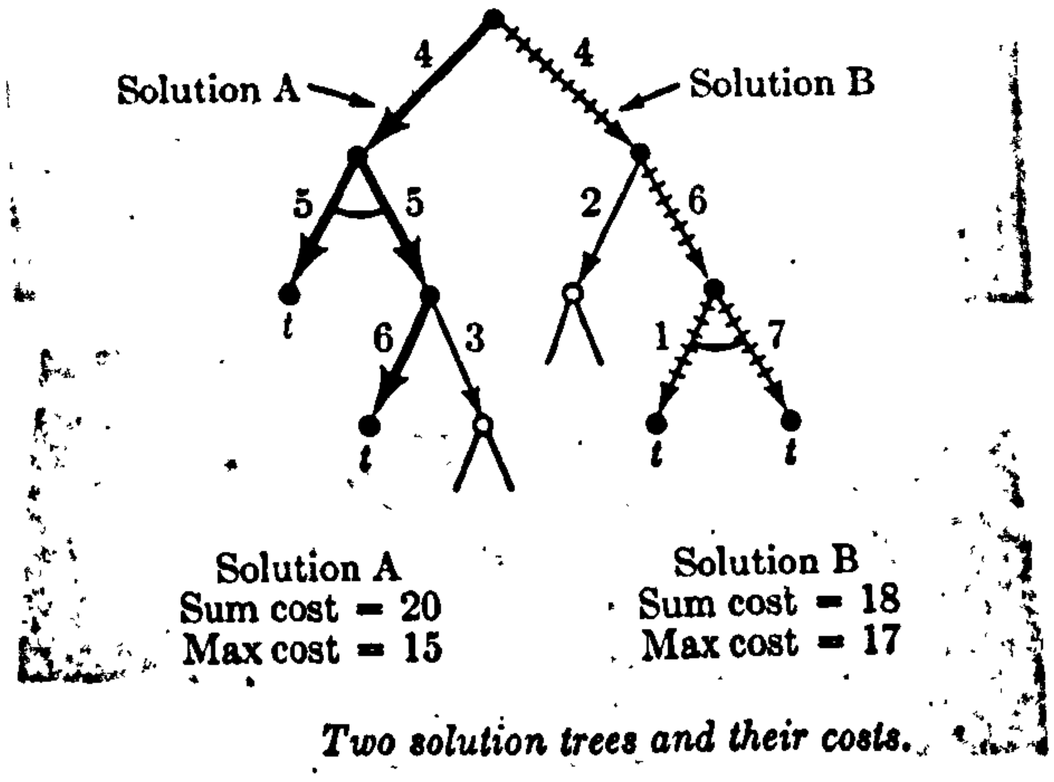
AND/OR search graph for travel problem

Figure 2.4.3

The AND/OR tree will be constructed piecewise by the repeated application of problem reduction to subgoals labelling nodes. When a sufficient portion of the tree has been constructed it can be determined whether the root node is soluble. If it is, the information as to how the subgoals were determined to be soluble can be collected and organized into a statement as to how the problem can be solved. Figure 2.4.3 shows how this can be done for the problem of planning a journey from Bradford to Birmingham.

As when searching a state-space it may be required to find not

simply a feasible solution, but an optimal solution. A solution tree is defined to be a subtree of an AND/OR tree containing the information that is both necessary and sufficient to prove that the root node is soluble. Nilsson (1971) gives two alternative definitions of the cost of a solution tree, both of which are based on costs associated with arcs of the tree. The sum cost is the sum of all the arc costs in the solution tree. The max cost is based on the costs of paths from the root node to terminal nodes in the solution tree. For any such path the cost is the sum of the costs of the arcs making up the path. The max cost is defined to be the highest such path cost. These definitions are illustrated in figure 2.4.4.



Two solution trees and their costs.

Figure 2.4.4

The problem of finding an optimal solution is equivalent to that of finding a solution tree having minimal cost within the entire AND/OR tree. Such a solution tree is called an optimal solution tree. Let $c(n_i,n_j)$ be the cost of the arc between node n_i and its successor n_j. Let $h(s)$ denote the cost of the optimal

solution tree rooted at the start node s. h(s) is defined recursively in terms of the minimal cost, h(n), for a solution tree rooted at any node n.

i) If n is a terminal node (labelled with a primitive problem) h(n)=0 if n is soluble and is undefined if it is not.

ii) If n is an OR node having successors $n_1,\ldots,n_k$ then

$$h(n)=\min_i [c(n,n_i)+h(n_i)] \quad .$$

iii) If n is an AND node having successors $n_1,\ldots,n_k$ then

$$h(n)=\sum_{i=1}^{k} [c(n,n_i)+h(n_i)] \qquad \text{for sum costs,}$$

$$h(n)=\max_i [c(n,n_i)+h(n_i)] \qquad \text{for max costs.}$$

During the construction of the AND/OR tree it may be possible to use an heuristic function, $\hat{h}$, to estimate the costs of nodes which are being made explicit. Nilsson (1969, 1971) gives an algorithm which ensures that if $\hat{h}$ satisfies certain conditions the first solution tree constructed is optimal.

2.4.3 Preferred reduction search

Where problem reduction trees are used in the present work satisfactory heuristic functions have not been available. The development of a search tree has been guided by a combination of the association of values with nodes and the ordering of possible methods of reduction of a node.

The basic iteration of the process, which will be referred to as preferred reduction search, is:

i) Look for the node with the lowest associated value.

ii) Perform the most preferred of the remaining possible

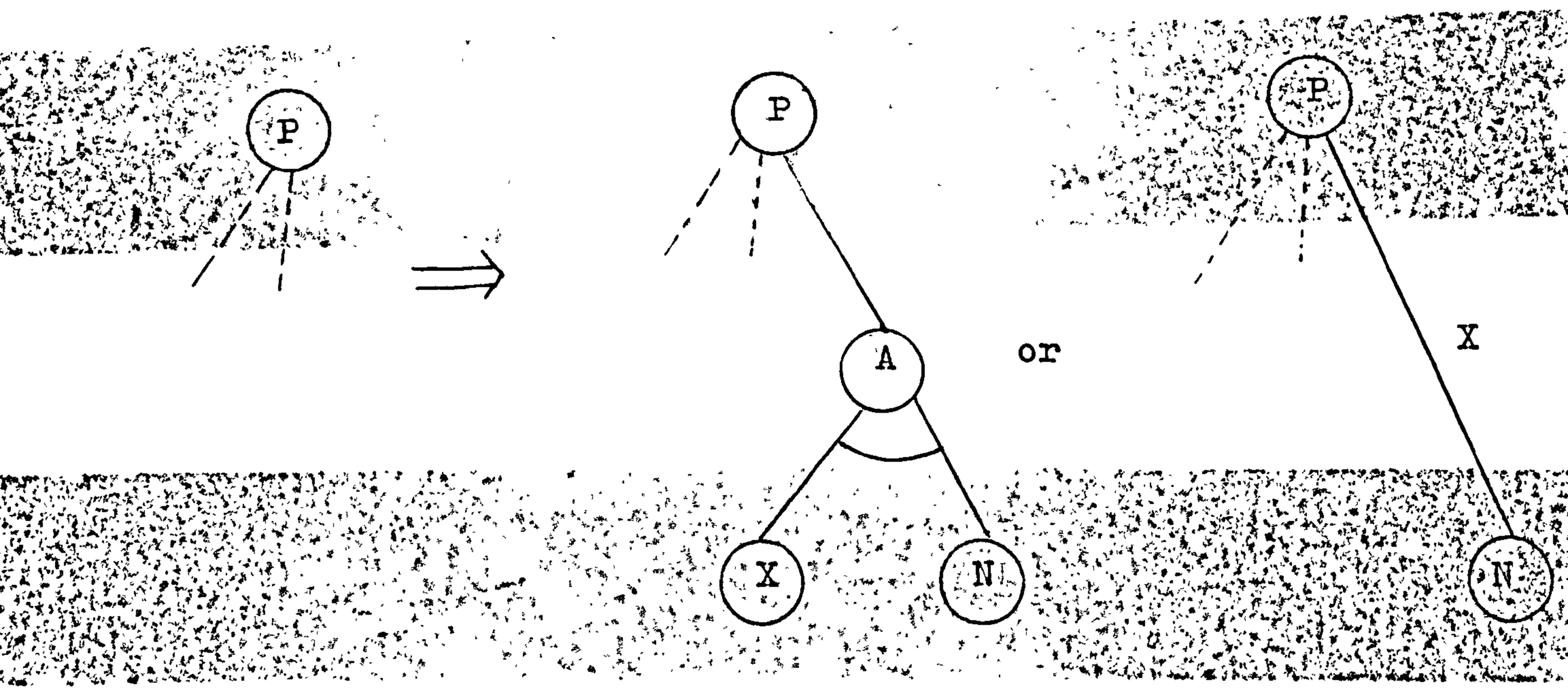
problem reductions for this node (some may have already been performed). This may provide a solution for the problem at this node, or cause the creation of one or more new nodes.

iii) Increment the value of the node to which the problem reduction was applied. The size of the increment will determine the frequency of return to this node during the search. Thus the increment will be small when it is desirable that a large number of alternative reductions should be made and large when it is not.

This search process differs from the theoretical model described above in that a subgoal for which a solution could be directly identified, and which might therefore be regarded as primitive, may be instead reduced, leaving a subgoal remaining to be solved. The reason for this derives from the fact that the goal of the search will be one of a sequence which must be achieved for the solution of the overall problem. The details of the solution of the present goal will affect the nature of later goals in the sequence, and use of the directly identified subgoal solution could lead to the resultant solution sequence being sub-optimal.

The form of the reduction may be to identify a category of reductions that might be performed and to create a new node at which the subgoal is associated with this category of reductions. This is a reduction in the complexity of the problem of choosing possible reductions. Alternatively the reduction may be a simple splitting of the subgoal into component subgoals. A third

possibility is that the reduction may be of the form "the solution is to consist of primitive element X and such other elements as are necessary to complete the solution", the determination of these other elements constituting a new subgoal. A representation of this type of reduction in terms of an AND/OR graph is shown in figure 2.4.5. P denotes the subgoal being reduced. Other reductions may already have been applied to it; these are indicated by the dotted lines. N denotes the new subgoal being set up. A is a constructed AND node linking N to the primitive element X. A simpler notation can be used and is shown in the third part of the figure. Here the arc connecting P to N is labelled with X. The way in which the components of a solution represented in this manner can be collected together is obvious.



Representation of problem reduction form

Figure 2.4.5

Whenever the search process produces an explicit solution for a subgoal some collection will occur. This may involve the modification of ancestor nodes to take

account of the solution information now available or evaluation of a feasible solution of the goal. Again the problem of a criterion for terminating the search arises. In the absence of other criteria, searches are terminated after they have consumed an amount of computing resources which experience indicates usually produce a feasible solution where one exists and an optimal solution when such is being sought. Note also at this point that the trees used in the actual computation differ in their detailed structure from the abstract trees that have been used to describe the method.

2.4.4 Relationship to state-space search

The process of solving a problem by problem reduction can be represented as a state-space search. Each state is a partially constructed problem reduction tree. The operators are the problem reduction operations that extend that tree. The start state is the unreduced problem, and the goal states are trees that include solution trees.

Conversely a state-space graph can be regarded as an AND/OR tree containing only OR nodes. If the alternative notation introduced in section 2.4.3 is extended by replacing terminal nodes labelled with primitive problems by the sequence: node, arc labelled with solution to primitive problem, terminal node, the correspondence becomes obvious.

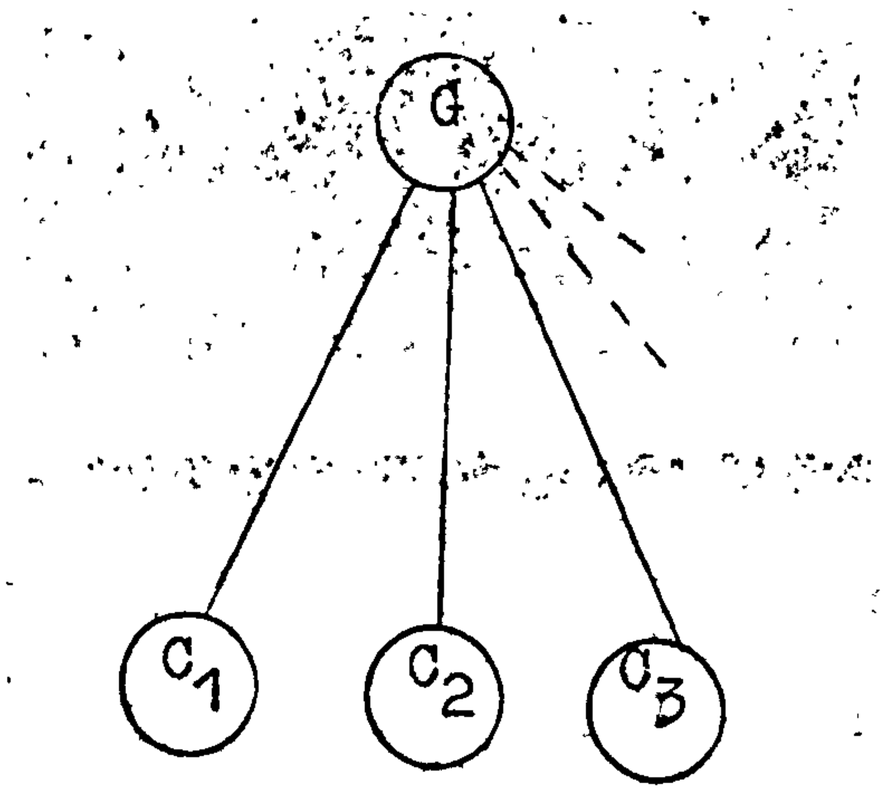
The name applied to a problem solving method may thus on occasion be regarded as somewhat arbitrary. A particular example of this is the case where all the nodes of the tree developed during problem solving are OR

nodes, but the operations used to determine the successors of different nodes are of essentially different types. Clearly this may be regarded as a state-space search, but it can be convenient to regard it as problem reduction, the operations at different nodes being the reduction of different types of subgoal.

2.4.5 Combined search methods

When an optimal solution for a problem is being sought it may be the case that the possible costs of solutions can be listed in ascending order as a sequence that proceeds by discrete steps. In this case a certain amount of computing resources can be devoted to trying to find a solution with the least cost. If one is not found, a certain amount of resources are allocated to trying to determine a solution with the next least cost, and so on. This strategy, which will be referred to as cost alternative reduction, corresponds to the OR graph shown in figure 2.4.6. Finding an optimal cost solution for goal G can be reduced to the problem of finding a solution with the least cost, C_1, or finding a solution with the next least cost, C_2, and so on.

Within the cost alternative reduction, a method must be selected for attempting to find a solution with a given cost. For the abstract 1½-dimensional trim-loss problem (chapter 3), where the distinct costs correspond to distinct alternative amounts of material that may be produced, an ordered operator search is used to search for a solution with a given cost. The total solution technique will be referred to as ordered operator search within cost alternative reduction.



Cost alternative reduction

Figure 2.4.6

Alternatively it may be the case that the cost of solving subgoals is such that once a subgoal has been set up and a solution to it found that is judged to be a component part of the most desirable solution to the overall goal, resources can only reasonably be allocated to finding a solution to the residual problem, and similarly with subsequent reductions. This strategy corresponds to attempting to find a path from the start node to a goal node in a state-space graph allowing only one attempt to determine which successor of a given node lies on the path to the most desirable goal node that can be reached from that node. The success of such a strategy is limited by the precision of methods available for selecting such successors.

A method must be selected for finding solutions of individual subgoals. Where this method is that of preferred reduction search, the total solution technique will be referred to as <u>preferred reduction within non-backtracking search</u>.

2.5 Branch-and-bound

2.5.1 Structure of the method

Branch-and-bound (Lawler and Wood, 1966) uses the construction of a tree as a vehicle for the determination of optimal solutions. The discussion here will be phrased in terms of a search for a minimal cost solution. Each node of the branch-and-bound tree is labelled with a set of features that a feasible solution may possess, the presence or absence of some features being undecided for that node. A node is added to the tree by considering some extant node and creating as a successor to that node one which has all the features determined for its parent and some other determined feature, the inclusion or exclusion of one that was previously undecided. A terminal node of a branch-and-bound tree is one whose label specifies sufficient determined features to define a unique feasible solution.

For branch-and-bound to be an applicable method for the solution of a problem it must be possible to construct a bounding rule. This is a function which, given a set of features of a solution, determines a lower bound on the cost of any solution possessing these features.

A common convention, which will be followed here, is to construct branch-and-bound trees as binary trees. Each time a node is considered two successors are created, one having all the determined features of its parent and some previously undecided feature included, the other having all the determined features of its parent and the same previously undecided feature excluded. The parent node is

then marked as unavailable for further consideration. The method used, when considering a node, to select which currently undecided feature shall be a determined feature of the successors is called the branching rule.

There are several strategies for selecting the order in which nodes are developed, i.e. have their successors constructed. One possible strategy is to develop always that undeveloped node having least lower bound. This can be expensive in terms of computing resources, since each development iteration will involve a complete reconstruction of the solution details labelling the current node. Often it is more economic to repeatedly develop newly created nodes. Suppose a node has been selected for development, as being the undeveloped node with least lower bound. Its successors are created. The successor which includes the previously undecided feature becomes the next node to be developed. The process repeats until either a terminal node is reached or the lower bound of the node due to be developed exceeds the cost of a known feasible solution. This is the go-right strategy. The go-left strategy is similar, the choice of successor for development being that in which the previously undecided feature is excluded. The advantage obtained with either of these strategies is that the iterations involving the development of successors do not require a complete reconstruction of solution details, but only a modification of those already stored to take account of the decision that has just been made.

Whichever strategy is used, the determination of a feasible solution will be followed by an analysis of the

tree so far constructed to establish whether nodes as yet undeveloped can be eliminated as candidates for development. When a feasible solution is determined its cost is evaluated. If this is less than the cost of any previously determined feasible solution then the lower bound of each undeveloped node is compared with it. Any node whose lower bound is greater than the cost of a feasible solution can be eliminated as a candidate for development. This can be done by marking it as "already developed".

The root node of the tree, which will be the first node to be developed, has no determined features. Development of the tree continues until all nodes are marked as "already developed". The lowest cost feasible solution found during the development is then the optimal solution.

2.5.2 An example

To explicate some of the points in the description above, the approach of Little, Murty, Sweeney and Karel (1963) to the travelling salesman problem will be considered, using the asymmetric 10-city problem defined by the matrix in figure 2.5.1 as an illustrative case. Here and elsewhere where problems derived from graphs are being described, vertex and link will be used as synonyms for "node" and "arc" when the problem graph is being referred to, "node" and "arc" being reserved to describe the graph developed by the problem solving method.

	1	2	3	4	5	6	7	8	9	10
1	∞	24	18	22	31	19	33	25	30	26
2	15	∞	19	27	26	32	25	31	28	18
3	22	23	∞	23	16	29	27	18	16	27
4	24	31	18	∞	19	13	28	9	19	27
5	23	18	34	20	∞	31	24	15	25	8
6	24	12	17	15	10	∞	11	16	21	31
7	28	15	27	35	19	18	∞	21	21	19
8	13	24	18	13	13	22	25	∞	29	24
9	17	21	18	24	27	24	34	31	∞	18
10	18	19	29	16	23	17	18	31	23	∞

Data for travelling salesman problem

Figure 2.5.1

The bounding rule is:

i) Set the lower bound to zero.

ii) For each explicitly included link (i,j) add its cost to the lower bound and delete row i and column j of the matrix.

iii) For each path consisting of explicitly included links determine the starting vertex p and the ending vertex m and set element (m,p) of the matrix to ∞.

iv) For each explicitly excluded link $\overline{(k,l)}$ set element (k,l) of the matrix to ∞.

v) Subtract the smallest element of each row of the matrix from every element of that row, and, having done that, subtract the smallest element of each column from every element of that column. Add all these elements to the lower bound.

The branching rule is:

i) Let $\propto(k)$ be the second smallest element in row k of the matrix constructed by the bounding rule and $\beta(l)$ the second smallest element in column l.

ii) Find the pair (k,l) for which the (k,l) element of the matrix is 0 and $\Theta(k,l)=\alpha(k)+\beta(l)$ is largest.

This chooses (k,l) so that the successor node having (k,l) excluded has the largest possible lower bound.

The development strategy is that of always developing the node with least lower bound. The resultant branch-and-bound tree is shown in figure 2.5.2. The circles represent the nodes; inside the circle is shown the inclusion or exclusion which created it, to the left the sequence number of its creation, and to the right its lower bound. Full details of the determined features at a node are obtained by collecting the inclusions and exclusions from the node and its parents. Such collection would occur if the method were implemented on a computer. Consideration of this point indicates the possible merits of go-right or go-left as development strategies.

2.5.3 The use of heuristic information

In the example just described the branching rule takes no account of the particular problem being considered. Branching to maximize the bound at one of the successors could be equally applicable to other problems to which branch-and-bound was applied. Its merit is that it is likely to restrict the size of the tree constructed. This is usually desirable when branch-and-bound is being used. Such a branching rule may be called domain-independent, as it is independent of any particular problem domain.

However, it may be useful to construct a branching rule which takes account of the particular problem domain. Such a rule may be called domain-specific. One

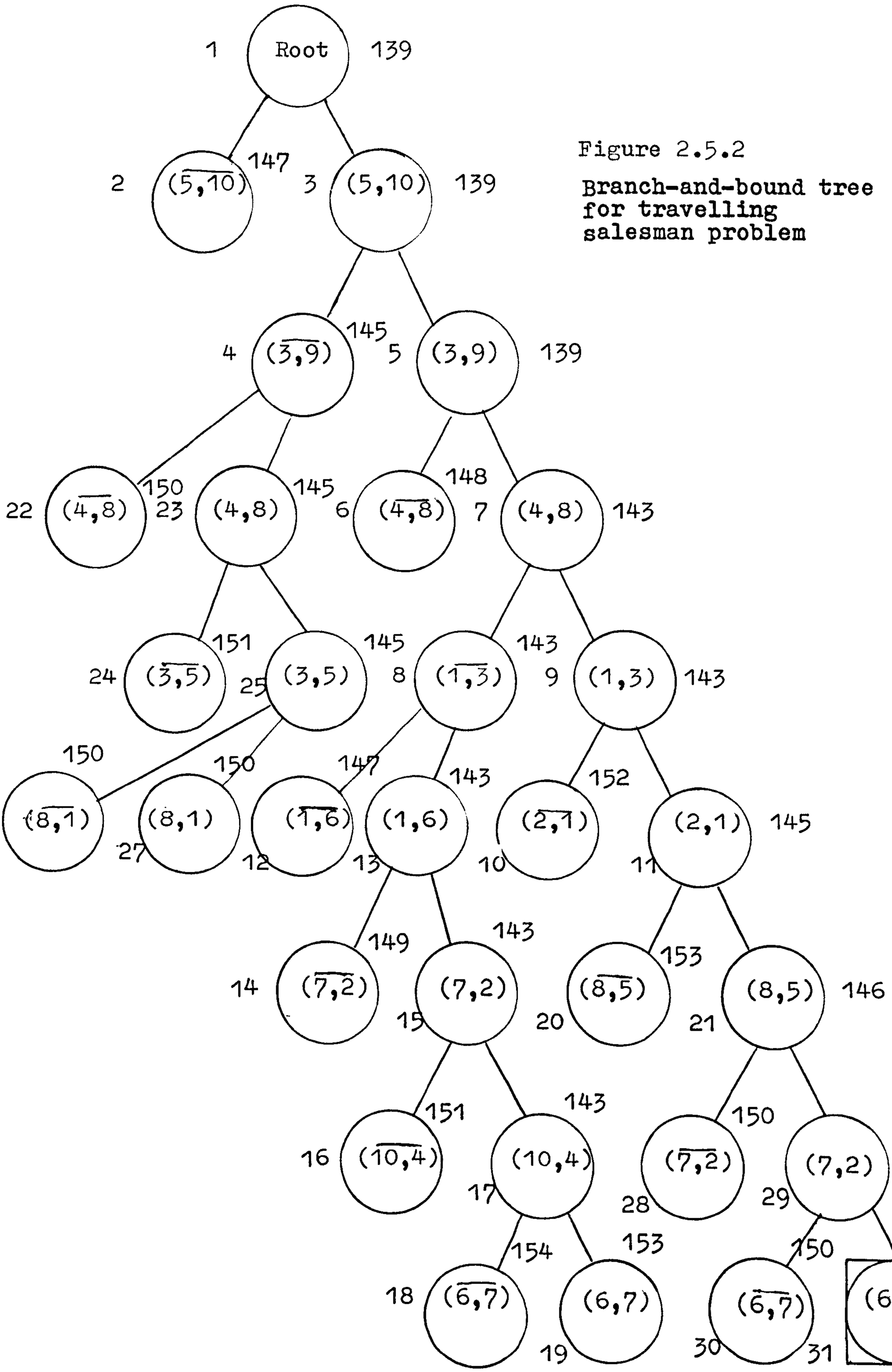


Figure 2.5.2

Branch-and-bound tree for travelling salesman problem

advantage of a domain-specific branching rule is that it may cause the optimal solution to be generated early in the construction of the branch-and-bound tree, in which case, as nodes will be marked as unavailable for development which would not be if their lower bounds were compared with the cost of a non-optimal feasible solution, the size of the branch-and-bound tree constructed is reduced.

As an example of a domain-specific branching rule, let us consider one used in finding p-medians (Boffey and Hinxman, 1976). A p-median of a graph with n vertices is a subset, V, of the vertices such that $|V|=p$ and $\sum_j h_j d(V,j)$ is minimal, where $d(V,j)=\min_{r \epsilon V} d(r,j)$. $h_j > 0$ is the weight of vertex j and d(r,j) is the shortest distance between vertices r and j. A feasible solution of this problem is then any subset of p vertices, the optimal solution satisfying the minimality condition.

If V_f is a feasible solution and j any vertex in the graph, then there is some vertex $r \epsilon V_f$ for which d(r,j) is minimal. j will be said to be <u>associated</u> with r. On average n/p vertices are associated with a vertex in V_f. The heuristic assumption is that in the optimal solution the sets of vertices associated with the different vertices of V are of approximately equal size.

At a node x of the branch-and-bound tree some vertices will have been explicitly excluded from appearing in any descendant solution, some will have been explicitly included, and the remainder will be undecided. Denote these sets respectively by E_x, I_x, U_x. The branching rule is:

i) Let $W_x(i)$ be the $\lceil n/p \rceil$ vertices nearest to $i \in I_x$ (including i itself), where $\lceil\ \rceil$ denotes the smallest integer not less than. Let $W_x = \cup W_x(i)$.

ii) For each $j \notin W_x \cup E_x$ find the sum of the $\left\lceil \frac{n-|W_x \cup E_x|}{p-|I_x|} \right\rceil$ smallest values of $d(j,k)$, $k \in W_x \cup E_x$.

iii) Branch on the j for which the sum is smallest.

2.5.4 Relationship to state-space search

Branch-and-bound may be regarded as a special case of state-space search. The states are lists of determined features. The operators applicable to a state are the feature exclusion and inclusion defined by the branching rule. A goal state is one containing sufficient determined features to define a feasible solution. Development of the tree continues until it can be proved that a generated feasible solution is optimal.

Chapter 3 An abstract 1½-dimensional trim-loss problem

3.1 Statement of the problem

Material of constant width is produced as a continuous sheet. Orders for rectangles of the material are received and it is wished to minimize the amount of material produced in order to satisfy the orders. There are no constraints on the way in which the material may be cut.

The width of the sheet and the dimensions of the order rectangles are integers. The dimension of the sheet perpendicular to the width is referred to as the depth.

3.2 Choice of method

The basic operation in the solution of the problem is to specify the cutting of one of the order rectangles from the sheet. Recognition of this fact leads to a natural state-space representation of the problem. An operator in this representation is either the specification cutting of an order rectangle from the sheet, or the designation of a unit area of the sheet as scrap, i.e. not included in any order. The states are partial cutting patterns, that is, instructions as to how part of the order set is to be cut.

An evaluation function for this representation would have to estimate the effect of successively applying best operators. It is not clear how such an estimate could be expressed as a numeric value, and most of the computation involved in creating the states resultant from applying these operators would in any case have to be done to provide the estimate. These facts, together with experience that ordered operator search was a useful method for finding solutions to pentomino puzzles (Golomb, 1965), suggested

ordered operator search as a suitable method to be applied to the problem.

At first sight it would seem that one search could be conducted which would be guided towards solutions which required the minimal amount of material to be used. However, in order to provide this guidance it would be necessary to predict accurately the depth of material required by the best cutting pattern which included a given partial cutting pattern. To do so would involve solving in essence the original problem. In addition, the amount of geometric information that can be extracted from the states of this search is far less than that which can be extracted from a state of a search for a solution to the problem of finding a cutting pattern having a specified depth. For these reasons the method adopted is ordered operator search within cost alternative reduction.

Define the theoretical minimum sheet depth for an order set to be the minimum sheet depth necessary for the total area of the order rectangles not to exceed the area of the sheet. A search is made for a solution of theoretical minimum sheet depth. If no such solution is found before the resources available for the search are exhausted, 1 is added to the sheet depth for which a solution is sought and a new search initiated. The process is repeated until a solution is found. The apparent inefficiency of discarding the search tree generated for each sheet depth is more than compensated for by the advantages of seeking solutions to better defined problems.

The following discussion relates to the solution of these subproblems. Hence the sheet can be regarded as

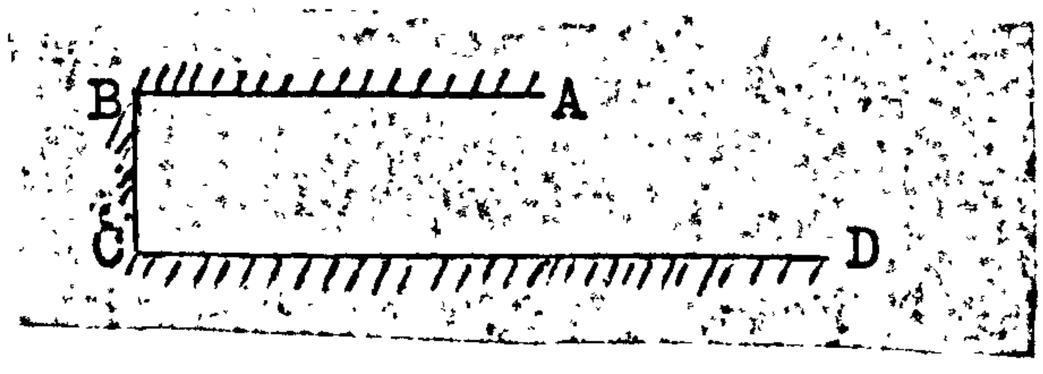
having both dimensions fixed.

3.3 Solution of the subproblems

3.3.1 Designated positions

Consider the uncut area in a partial cutting pattern. Its boundary may be regarded as including a number of segments of the form shown in figure 3.3.1, possibly with sides in common. Consider the lines of type BC. The shorter such a line is, the stronger the constraint as to which order rectangles can have sides lying along it. It would therefore seem desirable that when an operator is applied to a state it should be one that specifies the cutting of an order rectangle with a side along the shortest such line. In fact such a strategy leads to an over-concentration on lines parallel to the shorter side of a significantly non-square sheet. Better results are obtained by considering the line for which

length of BC x sheet dimension perpendicular to BC is smallest.

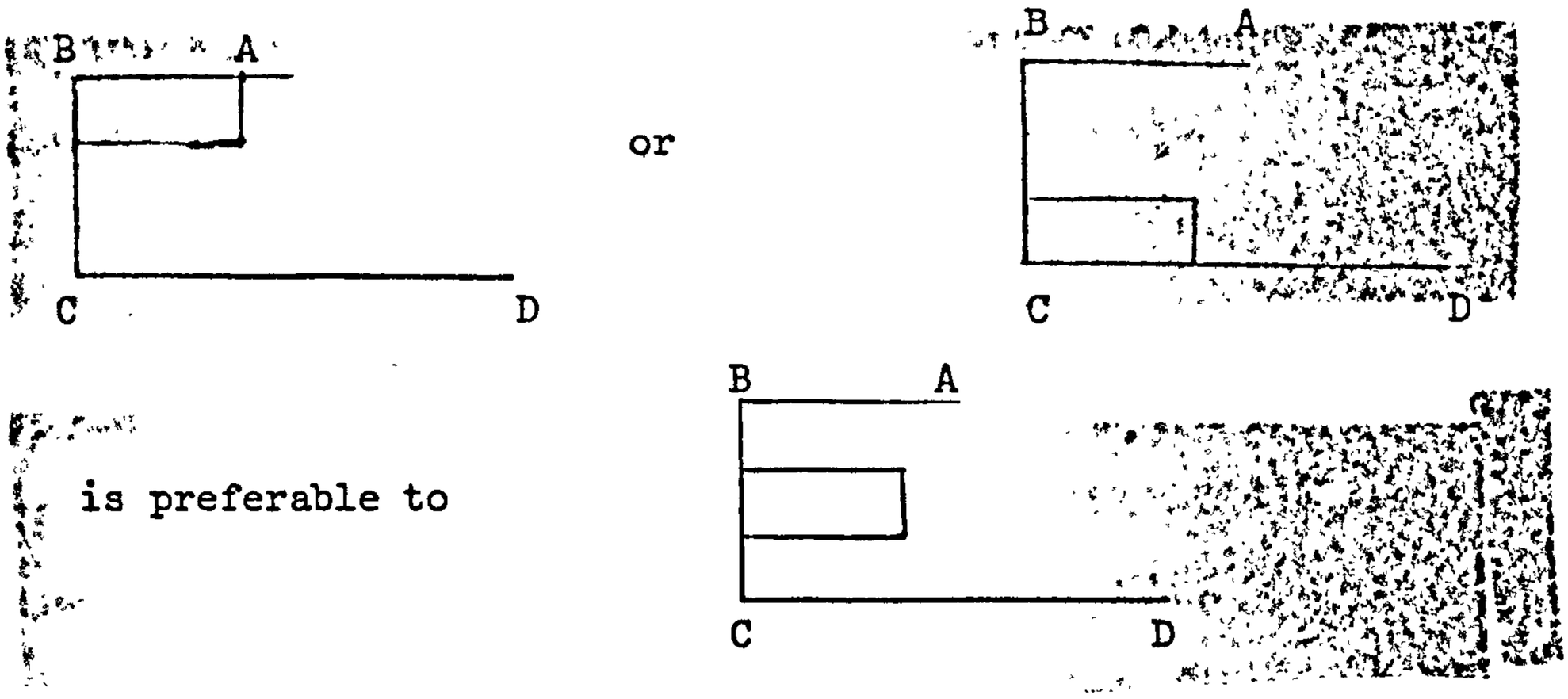


Segment of boundary of uncut area

Figure 3.3.1

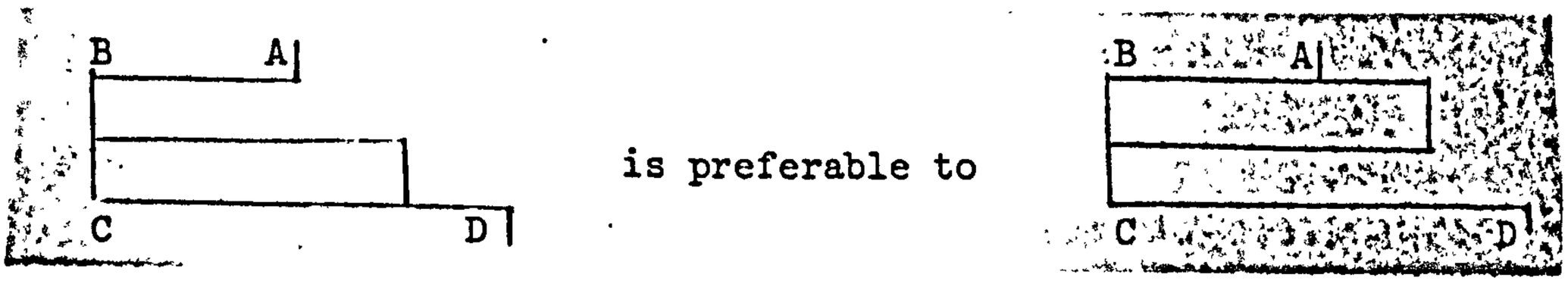
The operator applied should be chosen so as to make as weak as possible the constraints on the development of the resultant state. This can be regarded as maximizing the number of ways it can be developed. Clearly, for this to be case in the example, a corner of the order rectangle to be cut must correspond to either ABC or BCD, otherwise new,

stronger constraints will be generated (see figure 3.3.2). Consider, however, an order rectangle for which the side that will be perpendicular to BC is longer than AB but no longer than CD. Fewer constraints will be created if this side lies along CD (see figure 3.3.3).



Effect of positioning at corner

Figure 3.3.2



Effect of choice of corner

Figure 3.3.3

Hence the corner BCD is defined to be the <u>designated position</u> for the state under consideration. One corner of any order rectangle specified to be added to this partial cutting pattern will fit against this corner.

3.3.2 <u>Ordering of operators</u>

A grid of unit squares will be considered to be imposed on the sheet. For each non-square order rectangle two orientations are possible; its longer side may be

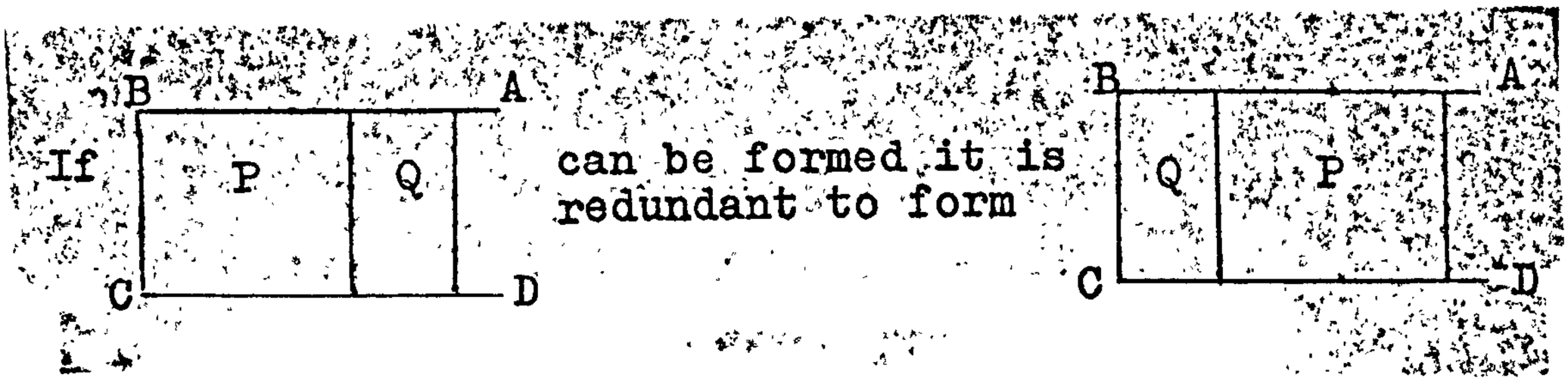
parallel to the width or the depth of the sheet.

Given that any order rectangle cut will be at the designated position the operator set for a state can be considered to consist of the (rectangle, orientation) pairs that specify the cutting of order rectangles not yet cut at the designated position, together with scrapping the grid square at that position, provided such scrapping is consistent with the solution of the problem.

The operator set can be ordered on the criterion of flexibility of the resultant state. The most significant factor is whether or not the edge placed along BC is equal in length to BC. Operators for which this is the case are preferred to those for which it is not. Within this, those operators for which the edge placed along CD is no longer than CD are preferred to those for which this is not so. Within the classes so defined the operators are arranged in descending order of the areas of the order rectangles to which they relate. The scrapping operator is least preferred.

Certain possible operators are regarded as redundant. These are:

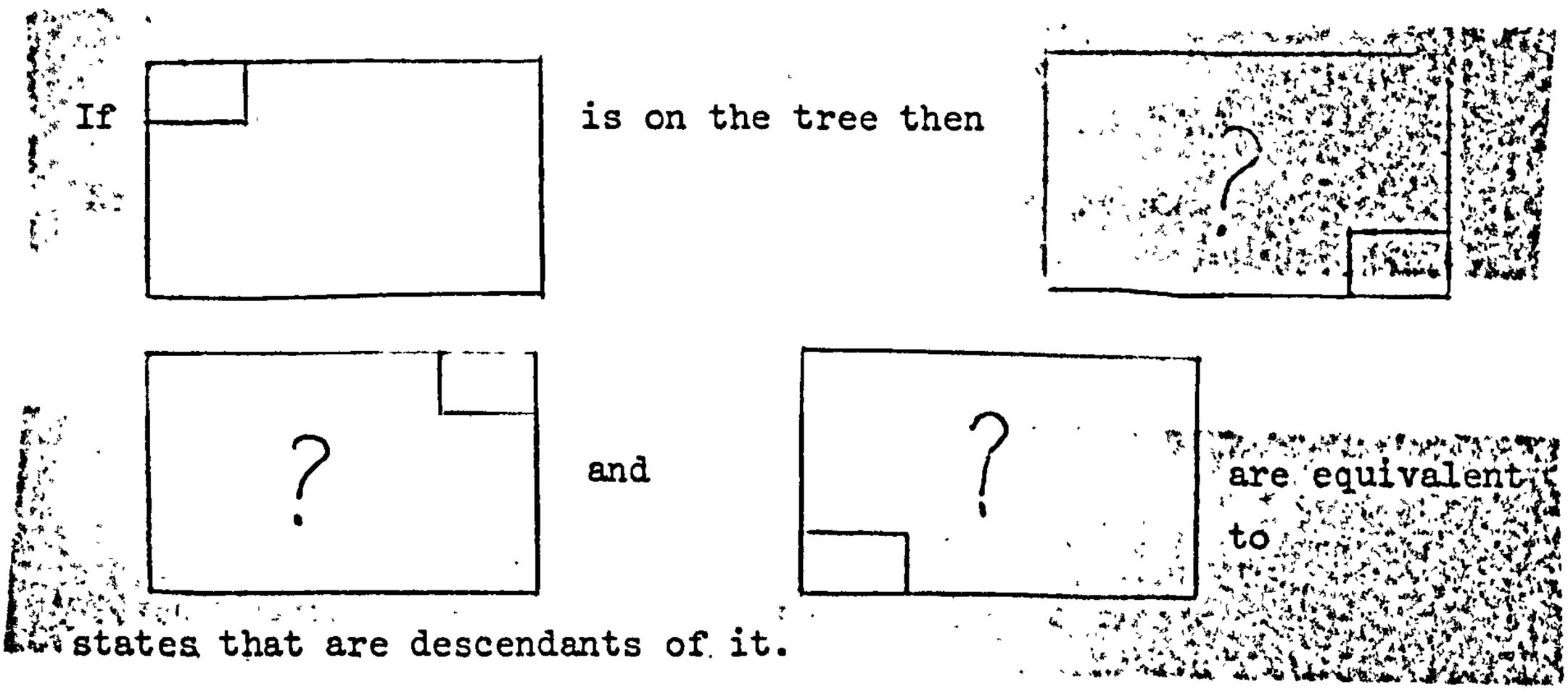
i) operators which have the same dimension in the BC direction as the order rectangle on the other side of the C end of BC where the other end of this side of that order rectangle is at B and its area is smaller, (see figure 3.3.4),



Redundancy due to equal side length

Figure 3.3.4

ii) operators which involve placing at a corner of the sheet other than the top left-hand corner an order rectangle which has been placed with that orientation in the top left-hand corner of the sheet in another state, (see figure 3.3.5).



Redundancy due to symmetry

Figure 3.3.5

These operators are not included in the operator list.

3.3.3 Feasibility of states

When an operator is applied to a state, the following checks are made on the resultant state:

i) that there is at least one operator that can be applied at the designated position,

ii) that there are no more isolated unused grid squares

System 4-75. The results of these tests are shown in table 3.4.2.

Where solutions of minimum theoretical sheet depth were found the computing time required never exceeded 4 minutes. In the case where the solution found did not have minimum theoretical sheet depth there were strong indications that no such solution existed. The long time taken to find the solution of this problem ($\sim$ 10 minutes) would have been smaller if a different criterion had been used for the termination of the first search.

Table 3.4.1

Listing of test data

In all cases the sheet is 20 units wide.

E: 2 × 10, 1 × 7, 4 × 2, 2 × 8, 2 × 3, 1 × 7,
5 × 9, 6 × 6, 3 × 8, 6 × 1, 10 × 2, 1 × 10, 3 × 5

F: 8 × 6, 1 × 10, 5 × 1, 5 × 5, 9 × 2, 5 × 2,
4 × 4, 2 × 5, 7 × 4, 4 × 9, 10 × 4, 4 × 9, 3 × 6

G: 10 × 3, 10 × 4, 1 × 10, 3 × 3, 5 × 3, 7 × 10,
1 × 1, 5 × 4, 4 × 8, 6 × 3, 9 × 4, 6 × 10, 3 × 3, 5 × 2

H: 9 × 4, 4 × 9, 5 × 7, 3 × 8, 9 × 4, 7 × 10,
4 × 9, 3 × 1, 3 × 8, 6 × 7, 2 × 3, 4 × 2, 2 × 2

I: 2 × 9, 6 × 5, 4 × 10, 8 × 8, 7 × 8, 7 × 1,
3 × 7, 1 × 8, 4 × 8, 6 × 4, 10 × 6, 5 × 7, 5 × 1

J: 2 × 2, 1 × 5, 7 × 8, 1 × 5, 6 × 9, 8 × 4,
3 × 2, 5 × 2, 3 × 2, 5 × 4, 1 × 5, 1 × 2, 3 × 5

K: 5 × 4, 10 × 2, 10 × 1, 3 × 7, 3 × 8, 3 × 7,
1 × 2, 9 × 3, 9 × 3, 2 × 9, 1 × 2, 1 × 8

L: 2 × 7, 3 × 10, 3 × 10, 5 × 5, 9 × 1, 8 × 7,
5 × 10, 5 × 7, 5 × 8, 5 × 1, 4 × 9, 3 × 6, 4 × 3

Chapter 4 A 2-dimensional trim-loss problem with varying stock costs

4.1 Statement of the problem

The problem considered in this chapter is based on one arising in the oil-rig construction industry. A set of steel rectangles (pieces) is required. Large steel plates of the appropriate thickness can be ordered from the steel works. The cost per unit area of such plates depends upon their dimensions. The problem is to determine a set of sizes of plates to be bought from the steel works, and the way in which these plates are to be cut into the required pieces, such that the total cost of the steel is minimized.

4.2 Preliminary analysis

The length of a piece or plate is defined to be its longer dimension and its breadth its shorter dimension. The cost of a plate with length l metres and breadth b metres is calculated as l*b*(c+e(l,b)), where c is the basic cost of steel in £/sq m and e(l,b) the excess cost, in £/sq m, charged for a plate of those particular dimensions. Here the values taken are c=10 and e as given by table 4.2.1. c and e have been chosen arbitrarily, but the relationships between the values of e are based on a practical case.

For given l and b, c+e(l,b)) will be referred to as the directed unit area cost, denoted by d(l,b) and min(c+e(l,b),c+e(b,l)) as the (undirected) unit area cost, denoted by u(l,b). It can be observed that for fixed l the directed unit area cost is minimal when b is

M: 1 × 2, 5 × 3, 9 × 6, 4 × 10, 4 × 5, 10 × 4,
7 × 7, 9 × 7, 3 × 6, 1 × 4, 9 × 9, 2 × 7

N: 5 × 2, 9 × 5, 6 × 9, 8 × 5, 10 × 3, 8 × 3,
5 × 1, 8 × 7, 8 × 5, 5 × 6, 2 × 2, 3 × 7, 1 × 1

O: 4 × 4, 9 × 1, 9 × 9, 4 × 9, 8 × 9, 3 × 9,
9 × 4, 6 × 10, 4 × 8, 3 × 7, 5 × 2

P: 10 × 6, 7 × 7, 6 × 4, 7 × 2, 5 × 9, 2 × 6,
10 × 8, 5 × 1, 2 × 5, 5 × 7, 1 × 6, 2 × 3, 2 × 7

Q: 10 × 3, 10 × 4, 1 × 10, 3 × 3, 5 × 3, 7 × 10,
1 × 1, 5 × 4, 4 × 8, 6 × 3, 9 × 4, 6 × 10, 19 × 1

R: 9 × 1, 10 × 2, 1 × 9, 9 × 6, 4 × 7, 5 × 9,
8 × 9, 6 × 5, 2 × 7, 8 × 4, 3 × 10, 9 × 2

S: 6 × 3, 3 × 7, 2 × 6, 2 × 4, 2 × 3, 6 × 6,
10 × 4, 5 × 10, 6 × 5, 10 × 4, 6 × 5, 6 × 5

Table 3.4.2

Results of test runs

Case	Trim Loss	C.P.U. secs	STCT	Comments
E	0	212.16	407	
F	0	28.97	59	
G	0	113.73	191	
H	0	25.93	40	
I	0	149.59	207	
J	0	4.47	12	
K	0	18.02	43	
L	0	27.26	46	
M	0	12.12	16	
N	0	52.71	83	
O	20	596.75	500 + 110	Apparently best possible
P	0	82.76	140	
Q	0	152.55	175	
R	19	46.06	87	
S	19	45.01	110	

STCT denotes the number of developable states generated during the search. The abandonment of search in a subproblem occurred when STCT reached 500 and a solution had not yet been found.

Chapter 4 A 2-dimensional trim-loss problem with varying stock costs

4.1 Statement of the problem

The problem considered in this chapter is based on one arising in the oil-rig construction industry. A set of steel rectangles (pieces) is required. Large steel plates of the appropriate thickness can be ordered from the steel works. The cost per unit area of such plates depends upon their dimensions. The problem is to determine a set of sizes of plates to be bought from the steel works, and the way in which these plates are to be cut into the required pieces, such that the total cost of the steel is minimized.

4.2 Preliminary analysis

The length of a piece or plate is defined to be its longer dimension and its breadth its shorter dimension.

The cost of a plate with length l metres and breadth b metres is calculated as l*b*(c+e(l,b)), where c is the basic cost of steel in £/sq m and e(l,b) the excess cost, in £/sq m, charged for a plate of those particular dimensions. Here the values taken are c=10 and e as given by table 4.2.1. c and e have been chosen arbitrarily, but the relationships between the values of e are based on a practical case.

For given l and b, c+e(l,b)) will be referred to as the directed unit area cost, denoted by d(l,b) and min(c+e(l,b),c+e(b,l)) as the (undirected) unit area cost, denoted by u(l,b). It can be observed that for fixed l the directed unit area cost is minimal when b is

in the range 2.251 to 2.5 and that for fixed b the directed unit area cost is minimal when l is in the range 4.0 to 7.999. The directed unit area cost considered as a function of l and b has an unique minimum.

Table 4.2.1

Table of excess costs e(l,b)

		Length l				
	min(mm)	1500	3000	4000	8000	12001
	max(mm)	2999	3999	7999	12000	16000
Breadth b						
min(mm)	max(mm)					
450	599	9.0	8.0	7.0	8.0	9.0
600	799	8.0	7.0	6.0	6.5	7.5
800	999	7.0	6.0	5.0	5.5	6.5
1000	1299	6.5	5.5	4.5	5.0	6.0
1300	1600	5.5	4.5	3.5	4.0	5.0
1601	1800	4.5	3.5	2.5	3.0	4.0
1801	2100	3.5	2.5	1.5	2.0	3.0
2101	2250	3.0	2.0	1.0	1.5	2.5
2251	2500	2.0	1.0	0.0	0.5	1.5
2501	2750	3.0	2.0	1.0	1.5	2.5
2751	3000	3.5	2.5	2.0	2.5	3.5
3001	3250	4.5	3.5	3.0	3.5	4.5
3251	3500	5.5	4.5	4.0	4.5	5.5
3501	3750	6.5	5.5	5.0	5.5	7.0
3751	3950	8.0	6.5	6.0	6.5	8.5

For given l and b there can be determined the minimum value of the unit area cost for any plate from which a piece with these dimensions can be cut. This value will be referred to as the minimal unit area cost for that piece and denoted by m(l,b). Let P_{lb} denote a piece of length l and breadth b and $\mathcal{P}$ a set of such pieces $\{P_{l_1 b_1}, P_{l_2 b_2}, \ldots, P_{l_n b_n}\}$. Then a lower bound for the cost of obtaining $\mathcal{P}$ is given by $\sum_{i=1}^{n} l_i b_i m(l_i, b_i)$. If any l_i or b_i exceeds the largest corresponding value in table 4.2.1, then there is no way of obtaining $\mathcal{P}$ and the cost of $\mathcal{P}$ may be regarded as infinite. Such sets will be ignored. Let l_i', b_i' denote the length and breadth of the smallest plate that can be ordered from which a piece of

length l_i and breadth b_i can be cut. It is only in the case where l_i or b_i is less than the corresponding minimum value in table 4.2.1 that these lengths and breadths will not be equal. An upper bound for the cost of obtaining P is given by $\sum_{i=1}^{n} l'_i b'_i u(l'_i, b'_i)$.

4.3 Choice of method

The geometry of the problem, a small order set and no restrictions on the way in which a sheet can be cut, is similar to that of the 1½-dimensional trim-loss problem. As the use of ordered operator search has proved a satisfactory approach to that problem, it is reasonable to investigate it as a method for finding solutions to the present problem.

A stepwise method for the solution of the problem will have the structure:

i) Label all the pieces "unallocated".

ii) Select a subset of the unallocated pieces to be cut from a single plate, and determine the size of the plate and the instructions as to how it should be cut. (The plate size and cutting instructions together form a cutting pattern).

iii) Mark the pieces belonging to the subset as "allocated".

iv) If any pieces remain unallocated, go to (ii), otherwise exit with a feasible solution to the problem.

As a state-space search is being employed these steps must be broken down into a sequence of simple operations that will be the operators in the state-space search. A suitable formulation of the problem will now be

described.

Each state of the space will be a list of cutting patterns, together with a list of pieces not yet allocated to cutting patterns. One of the cutting patterns in the list may be incomplete, that is, it may specify certain pieces to be cut from a plate and their positions in it and certain areas of it that are to be regarded as <u>scrap</u> (trim-loss), but not specify the dimensions of it or the disposition of a possible remaining part of it. The start state has no list of cutting patterns, only a list showing all the pieces as unallocated. States in which the list of unallocated pieces is empty label terminal nodes of the state-space search tree. The conventional notion of goal state is not applicable; any state not including an incomplete cutting pattern represents a feasible solution. Let S_{lb} denote a plate of length l and breadth b. Then a feasible solution will consist of cutting patterns for plates $S_{l_i b_i}, \ldots, S_{l_j b_j}$ and a list of unallocated pieces $P_{l_{j+1} b_{j+1}}, \ldots, P_{l_k b_k}$. The cost of this solution is $\sum_{i=1}^{k} l'_i b'_i u(l'_i, b'_i)$.

Operators for this representation fall into three categories. Suppose a non-terminal state does not contain an incomplete cutting pattern. Then in a successor of it one of the currently unallocated pieces will be allocated as the first piece of a new (incomplete) cutting pattern. An operator which does such an allocation will be called a <u>new plate</u> operator. If a state contains an incomplete cutting pattern, the set of positioned pieces together with the areas designated as scrap may form a rectangular

area. It is possible to complete the pattern by giving the dimensions of this area as the dimensions of the plate from which it is to be cut, or alternatively some unallocated piece may be added to the incomplete pattern, giving another incomplete pattern. Operators applicable in this case will be called rectangular plate operators. The third possibility is that the state includes an incomplete cutting pattern that does not describe a rectangular area. Either an unallocated piece can be added to the cutting pattern, or an area can be designated as scrap. Operators applicable in this case will be called irregular plate operators.

Heuristic guidance for the search is based on operator ordering. When a node is added to the state-space search tree it is given a value of 0 and the operators that may be applied to it are arranged in an order of preference. Each iteration of the search has the form:

i) Locate the node with lowest value which has an operator that has not yet been applied.

ii) Apply the most preferred such operator to generate a new node.

iii) Increment the value of the node to which the operator was applied.

4.4 New plate operators

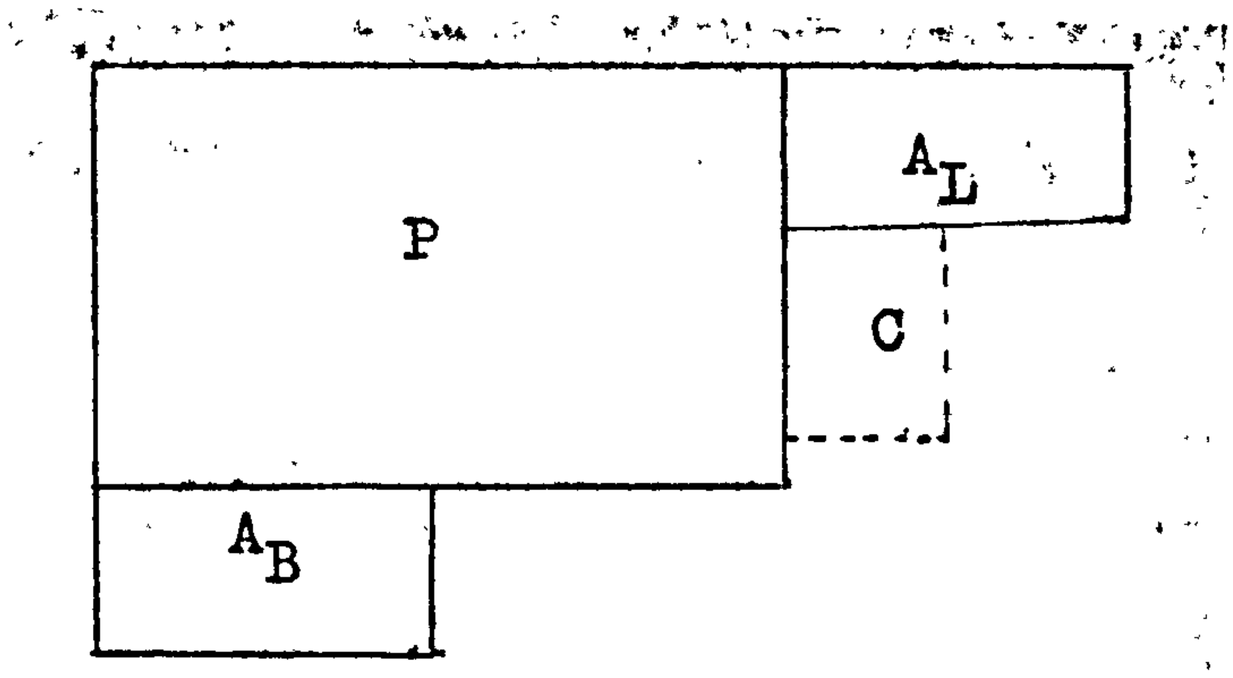
The broader a piece is, the more constrained are the dimensions of possible cutting patterns including it. Therefore the pieces may be ordered in decreasing order of breadth as a first step towards constructing an ordered operator list. However, an operator must specify

not only which piece is to be cut, but what its orientation is to be. This is done by relating the direction of longer dimension of the piece to the direction of the dimension of the plate from which it is to be cut which is expected to be larger when the design of the cutting pattern is complete. When these are the same the piece will be said to be oriented <u>along</u> the plate; when they are different the piece will be said to be oriented <u>across</u> the plate. At this stage of the design of the cutting pattern the best available estimate of the breadth of the plate is the minimum value it must have for the piece specified by the operator to be cut with the specified orientation. So, for each piece there is a pair of operators, one for each orientation. The pairs are sequenced on the breadth of the pieces; the sequencing within pairs is done by comparing the directed unit area costs for the two orientations, c+e(l,b) for orientation along the plate and c+e(b,l) for orientation across the plate. The orientation for which the directed unit area cost is less is the one that appears first in the sequencing. The quality of the heuristic information in this ordering is poor compared with the other operator orderings used, so the increment added to the cost of a node when one of these operators is applied is 5.

4.5 Rectangular plate operators

When these operators are to be applied, an incomplete cutting pattern exists which specifies how a rectangular area of a plate is to be cut. As possible operators are determined they are assigned values. The lower the value of an operator the more it is preferred.

The first operator to be considered is that which completes the cutting pattern by defining the dimensions of this area to be the dimensions of the plate from which it is to be cut. This operator is given value 0.



Alternative operator positions

Figure 4.5.1

The other operators involve the addition of instructions for cutting pieces to the incomplete cutting pattern. Let P be the rectangular area for which cutting instructions already exist (see figure 4.5.1). Let A be a piece with fixed orientation the instructions for cutting which are to be added to the cutting pattern. Then these instructions are such that one of the sides of A is juxtaposed with one of the sides of P and another side of A is collinear with a side of P. This can be done so that it is the longer side of P which is collinear with a side of A (position A_L) or the shorter side of P which is collinear with a side of A (position A_B). Let l_p be the length of P, b_p the breadth of P, l_A the dimension of the oriented piece A in the direction of the length of P, b_A the other dimension of A, and m_A the minimal unit area cost for A. The possibility of placing each unallocated piece with each orientation in the A_B position is considered and if the application of such an operator

would not create a pattern which could not be cut from a plate of an allowable breadth the operator is given value $(d(l_p,b_p+b_A)-d(l_p,b_p))*100-m_A$. This reflects the desirability of producing patterns to be used on plates which by virtue of their breadth have low unit area cost, and within this of finding patterns that include pieces of high minimal unit area cost.

Similarly, it is desirable that the length of the plate on which a pattern is to be used should be such that the plate has low unit area cost. If already $l_p \geq 4.000$, then no benefit is likely to accrue from placing a piece in the A_L position. Increasing the plate length will not reduce the unit area cost, and it is more likely that the piece being considered can be economically included in a new cutting pattern in which there is still flexibility as to the possible breadth than in the present one, where the breadth is constrained by the value of b_p. So no such operators will be included in the operator list for this state.

If, on the other hand, $l_p < 4.000$, then it may be worth placing a piece in the A_L position. Given an oriented piece A, all possible oriented pieces C that could be placed with one edge juxtaposed with an edge of A and another edge juxtaposed with an edge of P are considered. Let b_c be the length of the edge of C to be juxtaposed with an edge of P, and let $\mathcal{C}$ be the set of all oriented unallocated pieces that can be placed in the required position without the maximum possible pattern width being exceeded, together with an imaginary piece with edge lengths 0. Then for A, s(A) is defined as

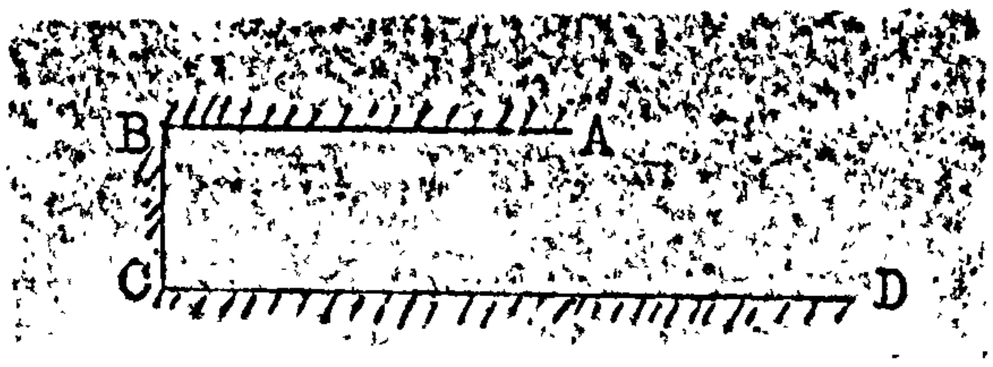
$\min_{c}|d(l_p, b_A+b_c)-d(l_p, b_p)|$. The value of the operator of placing the oriented piece A in the A_L position is then $s(A)*100-m_A$. The application of a rectangular plate operator may be regarded as the start of the design of a rectangular sub-pattern to be juxtaposed with an existing rectangular sub-pattern. The value assigned to the operator reflects that the two sub-patterns should have approximately equal breadth and that within this desideratum it is desirable to find patterns that include pieces of high minimal unit area cost.

Whenever a rectangular plate operator is applied at a node, 10 is added to the value of that node.

4.6 Irregular plate operators

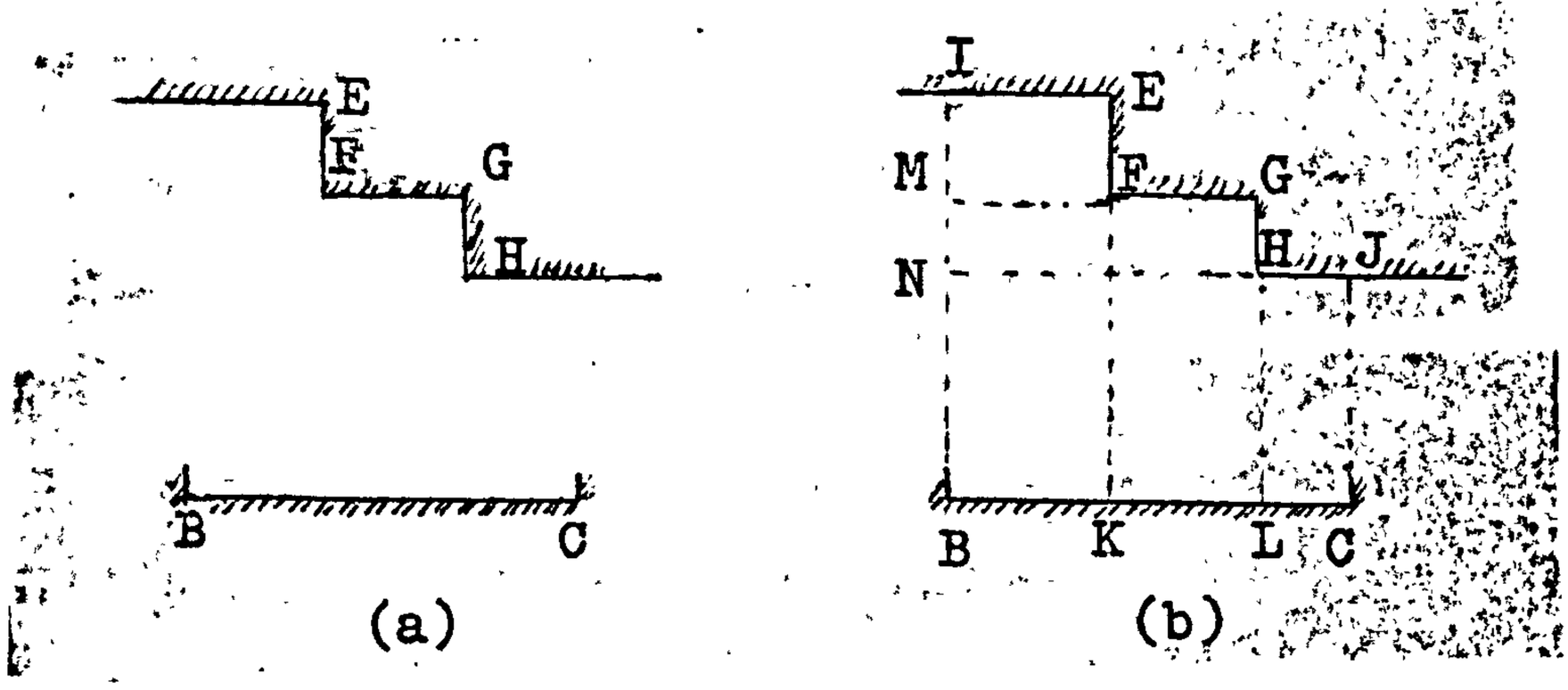
An irregular plate operator is either an addition of instructions for the cutting of another piece to an incomplete cutting pattern or an addition of instructions for designating an area as scrap. An individual designation as scrap is always for a rectangular area, and may be considered as instructions for cutting a dummy piece of the size of the scrap area.

In the generation of such operators the first decision made is as to the location on the plate at which the new cutting is to take place. Whilst this location is being decided the incomplete cutting pattern is considered to relate to the smallest plate size consistent with the pattern dimensions. The areas of the plate not covered by the cutting instructions have boundaries which include a number of segments of the form shown in figure 4.6.1, possibly with sides in common.



Segment of boundary of uncut area

Figure 4.6.1



Inscribed rectangles

Figure 4.6.2

The shortest line of type BC is determined. Reasoning similar to that in section 3.3.1 leads to the decision that the next piece whose cutting is to be specified should have an edge along BC and that one of its corners should be as near to B as possible if AB is longer than CD or otherwise that one of its corners should be as near to C as possible. However, since in this case the locations of the pieces on the plate are not yet finally fixed, some further analysis is necessary. The outline of the uncut area opposite BC is considered. This may be irregular, as shown in figure 4.6.2(a). The set of distinct maximal rectangles that may be inscribed in the uncut area is determined. In the example these are BIEK, BMGL, BNJC, as shown in figure 4.6.2(b).

For an orientation of an unallocated piece

consideration is given to the possibility of the piece being cut with one of its sides along BC, with one of its corners corresponding to the corner of a maximal rectangle determined by the required proximity to B or C described above. Some oriented pieces may exceed in at least one dimension the size of a maximal rectangle. When the possibility of placing such a piece is being considered in terms of such a rectangle, account is taken of the fact that the dimensions of the plate have not been specfied and so the positions in the plate of pieces already mentioned in the cutting pattern may be altered. Given such a piece and rectangle, changes to the positions of pieces in the pattern are hypothesized as follows:

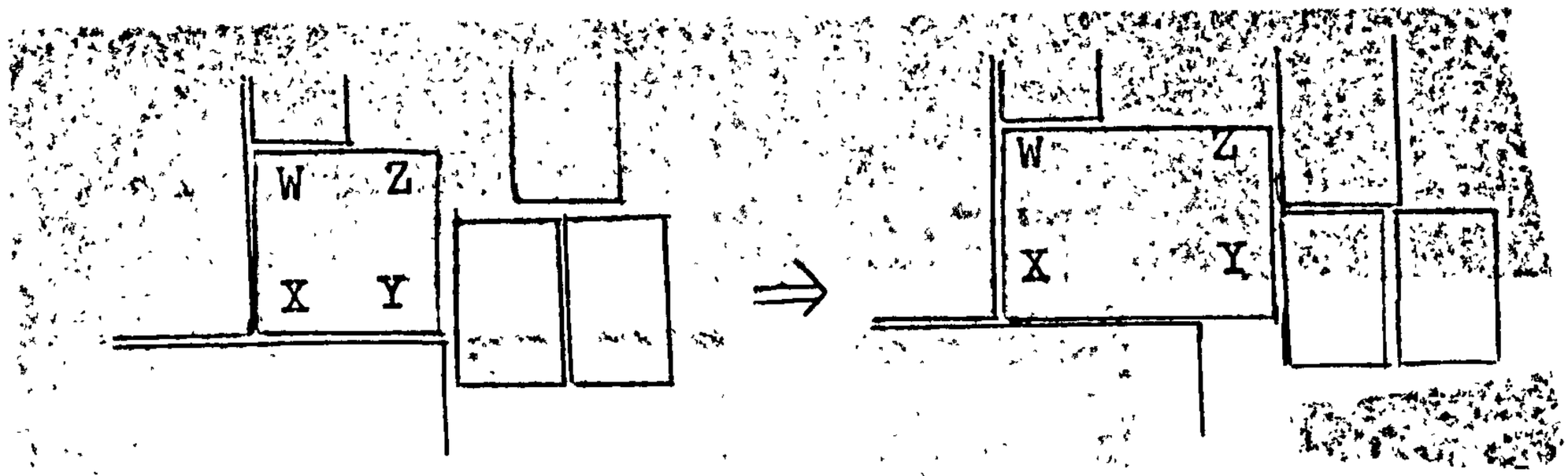
i) Let WXYZ be the minimal rectangle with XY coincident with BC.

ii) Consider the pieces already mentioned in the pattern to be rectangular objects positioned according to the pattern and capable of sliding in the plane of the plate.

iii) If the side of the oriented piece corresponding to XY is longer than XY, displace YZ in the direction of XY by the excess amount. The displacement will push other pieces in the direction of XY. The process is illustrated in figure 4.6.3.

iv) If displacement was necessary in (iii), replace WXYZ by the rectangle into which it was transformed.

v) If the side of the oriented piece corresponding to YZ is longer than YZ, displace ZW in the direction of YZ by the excess amount. This displacement will push other pieces in the direction of YZ.



Displacement of pieces

Figure 4.6.3

Let c be the length of BC and a the length of the shorter of AB and CD. Let s_o, t_o be the dimensions in the direction of BC and AB respectively of the minimal size plate to which the original cutting pattern could be applied and s_N, t_N the corresponding dimensions of the minimal size plate to which the cutting pattern transformed by (i) to (v) above can be applied, if these plate dimensions are allowable. If they are not allowable then the following calculation does not take place. The value of the operator of positioning an oriented piece of dimension p in the direction of BC and q in the direction of AB is calculated as:

i) if there is no displacement in steps (iii) and (v) above,

$(c-p)*a*u(s_o,t_o)$

ii) if there is a displacement only in step (v),

$u(s_N,t_N)*(a*(c-p)+q*(s_o-p)+s_o*t_o)-u(s_o,t_o)*s_o*t_o$

iii) if there is a displacement only in step (iii),

$u(s_N,t_N)*(p*(t_o-q)+s_o*t_o)-u(s_o,t_o)*s_o*t_o$

iv) if there are displacements at both steps (iii) and

(v),

$u(s_N,t_N)*(q*(s_o-p)+p*(t_o-q)+s_o*t_o)-u(s_o,t_o)*s_o*t_o$

These values reflect the costs of areas of the resultant pattern that are likely to become scrap, and the benefit, or otherwise, of the change in estimated plate size.

For each oriented piece the calculation of this value is considered for each maximal rectangle. If the calculation can be done for at least one rectangle, an operator corresponding to the smallest value calculated is added to the operator list, which is ordered in terms of ascending order of these values. This operator consists of instructions to make the necessary displacements in the existing cutting pattern and to add to it the cutting of the oriented piece in the specified position.

One further operator is added to the operator list. This is the operator of designating a rectangular area having BC as one side and the other of length a as scrap. The value of this operator is $a*c*u(s_o,t_o)$. It is fairly unlikely that the application of an operator with a higher value than the scrap operator will lead to the generation of a useful cutting pattern, so when the scrap operator is applied at a node, 20 is added to the value of that node. When any other operator is applied at that node, 10 is added to the value.

Suppose a displacement is made in a cutting pattern which includes areas designated as scrap. The designation of these areas as scrap may not be appropriate in the resultant cutting pattern. For this reason, the pattern produced by the application of an operator involving

displacement consists of all the pieces (not scrap) of the parent pattern displaced suitably, together with the additional piece in the position specified by the operator. It may now be necessary to designate areas of this new pattern as scrap.

4.7 Results

A program embodying these concepts has been written in Algol 68-R (Woodward and Bond, 1972) and tested on an ICL 1906S against the 16 sets of data based on random numbers shown in table 4.7.1. The program performed 1500 iterations of the search for each set of data. It took approximately 1.25 minutes to process each set of data, although this time was strongly dependent on the amount of store allocated to the program. The results of these runs are shown in table 4.7.2.

It can be seen that the average value of the percentage excess of the solution cost over the lower bound is of the order of 10% and the maximum excess of the order of 15%. Simple consideration of the numeric properties of the data shows that solutions with costs equal to the corresponding lower bounds cannot be achieved, so these figures exceed the true difference from optimality.

Results obtained during the course of program development using other heuristics indicate that in specific cases the cost of the solution given by the present program may exceed the cost of the best possible solution by up to 3.6% of the lower bound. Whether such solutions could be regarded as adequate would depend on details of the practical situation to which they were to be applied.

Table 4.7.1

Listing of Test Data

Data Set 1	Data Set 2	Data Set 3
2226 x 603	2688 x 2016	2277 x 693
2712 x 2232	2574 x 2082	3090 x 1230
3147 x 2841	3180 x 1203	2169 x 1158
1734 x 663	2835 x 1767	2094 x 1296
2796 x 2736	1131 x 348	3090 x 1119
1431 x 1350	3078 x 1920	1497 x 1083
1347 x 597	1527 x 1239	2982 x 2679
1173 x 486	2511 x 2007	1941 x 597
3195 x 3135	2007 x 426	1968 x 1746
3051 x 981	2436 x 1005	3108 x 498
3108 x 1377	2820 x 1572	1743 x 1287
2700 x 1872	2679 x 2247	2079 x 2034
3285 x 1830	1962 x 1506	2949 x 1842
2670 x 1170	1983 x 1482	1263 x 861
3051 x 699	1539 x 588	3180 x 1572
1947 x 825	2241 x 1923	1296 x 1029
1722 x 1470	1191 x 552	1605 x 315
1491 x 1311	2580 x 1965	2916 x 1899
1866 x 606	3093 x 2397	2640 x 2457
2331 x 1464	3216 x 900	2928 x 1299
3261 x 861	2700 x 2442	2271 x 900
2877 x 2394	1614 x 786	2193 x 516
2871 x 1137	2943 x 2178	3138 x 2112
3171 x 2874	3267 x 336	2559 x 1125
1689 x 1674	1116 x 428	3021 x 2973

Data Set 4	Data Set 5	Data Set 6
2049 x 1872	3258 x 3063	1476 x 1215
2853 x 975	1344 x 345	2214 x 1335
3174 x 2634	2634 x 1224	1746 x 1611
3231 x 2724	762 x 453	2622 x 1269
3099 x 696	2274 x 1515	1182 x 870
1714 x 1635	2094 x 1509	3078 x 2052
2352 x 480	3201 x 2292	2652 x 1890
1323 x 1035	1290 x 834	2274 x 1644
2370 x 1821	1650 x 1383	2388 x 771
3060 x 2157	2364 x 681	1770 x 1182
3219 x 2172	624 x 576	3060 x 1494
3237 x 2349	2301 x 570	2520 x 1365
3054 x 2607	2199 x 1530	2469 x 2094
2988 x 1011	1812 x 1566	1584 x 579
2919 x 2349	2676 x 2142	3075 x 1080
2943 x 2625	2733 x 2199	3084 x 420
2268 x 2067	705 x 606	3054 x 2433
2988 x 1530	3036 x 2577	2004 x 498
2733 x 1455	1695 x 1428	1635 x 1293
1770 x 1548	2562 x 663	3189 x 612
3045 x 585	3156 x 3090	2700 x 993
702 x 654	1386 x 696	1695 x 384
3228 x 3135	729 x 708	1644 x 1317
3090 x 2472	1362 x 702	1935 x 1395
2544 x 2301	2853 x 1239	2919 x 2199

Data Set 7	Data Set 8	Data Set 9
1776 x 1545	1944 x 1824	1971 x 483
3282 x 2838	2685 x 2118	2958 x 1167
2544 x 1113	2325 x 552	3030 x 927
3039 x 2214	3060 x 1986	2148 x 870
1836 x 1479	2142 x 1689	2178 x 1854
2355 x 1554	1509 x 726	2133 x 423
1824 x 1776	2610 x 1377	1830 x 1698
2178 x 1899	2751 x 813	2826 x 2454
2664 x 1197	2424 x 2166	1470 x 306
2424 x 531	1764 x 1455	2499 x 426
3117 x 1557	3159 x 3081	1062 x 351
1470 x 981	2691 x 2052	3234 x 2808
3204 x 1899	3168 x 2979	2745 x 825
3003 x 1584	2868 x 1185	2877 x 2535
1656 x 1455	2601 x 2001	1716 x 387
3279 x 1341	1869 x 1674	1056 x 918
1989 x 1278	2226 x 1440	2589 x 1185
2778 x 2709	1611 x 1341	3084 x 888
2391 x 1680	2502 x 1941	1902 x 1332
2811 x 1791	2202 x 1968	3281 x 2121
666 x 312	2520 x 576	2811 x 1845
660 x 579	3027 x 2268	3054 x 2031
3216 x 2076	1077 x 699	2724 x 690
3012 x 2688	2676 x 1677	3078 x 2682
1578 x 1098	1470 x 351	2547 x 1698

Data Set A	Data Set B	Data Set C
1563 x 1347	2451 x 351	579 x 384
2685 x 1893	2316 x 1083	2184 x 1134
1977 x 1215	3075 x 1353	3198 x 2502
2445 x 390	1476 x 756	3084 x 1044
3093 x 2499	2772 x 2127	501 x 480
1200 x 300	2379 x 2100	3129 x 2550
1491 x 717	2304 x 633	1812 x 1026
1068 x 1026	3012 x 2991	2721 x 1299
1809 x 1284	1878 x 663	1359 x 1080
2076 x 2016	3039 x 774	3297 x 2055
2757 x 1164	2610 x 1284	2091 x 1071
1419 x 1113	2961 x 1131	960 x 321
3222 x 1950	2685 x 588	3261 x 1026
2763 x 2427	1758 x 819	2847 x 2751
3012 x 2994	2454 x 1308	2304 x 1188
3255 x 2823	3117 x 1575	2829 x 1005
2010 x 1557	2427 x 678	1002 x 969
2844 x 747	1845 x 492	2892 x 363
1974 x 1881	2544 x 978	2127 x 981
3009 x 1581	2832 x 1851	2547 x 1842
2487 x 984	3252 x 1968	1746 x 513
1917 x 1623	2787 x 2013	2955 x 2106
2856 x 654	2400 x 561	2232 x 1635
3081 x 2793	2577 x 2235	2712 x 1617
3255 x 576	2721 x 1239	3234 x 468

Data Set D	Data Set E	Data Set F
2202 x 1926	2991 x 570	2328 x 1086
2526 x 2409	3054 x 1338	3012 x 2571
2772 x 1746	1959 x 1200	2967 x 753
513 x 318	1968 x 1887	2565 x 729
2508 x 1581	1179 x 711	2352 x 1209
2778 x 1584	2550 x 987	1203 x 1080
2718 x 1134	1443 x 1002	1917 x 960
807 x 471	2499 x 2028	2553 x 1272
3057 x 1974	1866 x 321	2340 x 1137
2169 x 1695	1170 x 510	2331 x 744
1770 x 1173	1821 x 438	2076 x 1452
3168 x 333	2709 x 1113	1647 x 951
3078 x 357	2544 x 1938	3249 x 3222
3117 x 1128	2037 x 1278	3156 x 1347
3093 x 2115	2766 x 2547	2332 x 378
2892 x 864	3081 x 372	3252 x 2400
1236 x 318	1686 x 1179	2217 x 405
2352 x 1656	1731 x 588	2682 x 2145
1599 x 1296	3048 x 963	1977 x 786
2994 x 2784	2943 x 1917	2268 x 504
2310 x 1014	2658 x 2370	3288 x 3114
3255 x 1881	2196 x 1053	3138 x 621
2355 x 1665	2436 x 1242	1755 x 1428
2220 x 1947	2652 x 2070	1458 x 957
732 x 501	2772 x 1515	1167 x 717

Data Set G
1572 x 966
3129 x 2982
2469 x 2271
1836 x 846
3174 x 1875
2400 x 1263
1176 x 822
3024 x 2883
3111 x 330
1851 x 378
732 x 318
2367 x 1911
3129 x 2484
3252 x 1911
1533 x 465
2166 x 1245
1620 x 1065
2841 x 1494
2454 x 1311
2535 x 1851
2961 x 2010
2250 x 1914
1326 x 720
2313 x 1212
2754 x 1947

Table 4.7.2

Results of test runs

Data Set	Lower bound on solution cost £	Upper bound on solution cost £	Actual cost of solution £	%age difference between actual solution and lower bound
1	1050	1380	1135	8.1
2	895	1271	993	10.8
3	897	1269	1000	11.5
4	1305	1641	1404	7.6
5	873	1198	934	6.9
6	767	1133	837	9.0
7	1034	1416	1193	15.4
8	1048	1399	1133	8.1
9	907	1245	1014	11.8
A	1003	1334	1125	12.2
B	860	1246	956	11.2
C	837	1199	946	13.0
D	871	1257	976	12.0
E	761	1136	816	7.3
F	892	1222	955	7.1
G	974	1311	1053	8.1
			Mean	10.0

Chapter 5 The optimal network problem

5.1 Statement of the problem

A communication network is to be set up between a set of n locations. The distances and traffic flows between the locations and the costs of constructing possible links of the network are given. It is required to minimize the user costs of the network constructed subject to a budget constraint on the cost of the network.

Formally, the problem can be stated in the following way. Let G be a graph with vertices 1,2,.....,n and links L_{ij}, i<j. Let there be associated with this graph matrices (l_{ij}) giving the length of the link between i and j, (t_{ij}) giving the traffic flow from i to j, and (p_{ij}) giving the "project cost" of the link between i and j. The budget is B and it is required to find a subgraph H of G such that $\sum_{L_{ij} \in H} p_{ij} \leq B$ and $\sum_{i,j} t_{ij} d_{ij}$ is minimal, where d_{ij} is the length of the shortest path from i to j in H, the distance between vertices not joined by any path being considered to be infinite.

5.2 Choice of method

A state-space formulation of the problem presents itself immediately:

i) the states (nodes) are lists of links to be included in the final network,

ii) an operator (arc) is the addition of a link to a list,

iii) the start state is an empty list,

iv) a goal state is a list satisfying the budget constraint on H with the property that no link may

be added to it without violating the budget constraint.

It is required to find the goal state for which the user costs are least.

There are a large number of applicable operators at each node of the state-space graph. An initial consideration of the geometry of the problem suggests an ordering of the operators, so ordered operator search appears to be an appropriate solution method. The manner of its application is described in section 5.3.

It is also possible to formulate the problem as one of branch-and-bound and it is of interest to compare the efficacy of the two methods. The branch-and-bound method is described in section 5.4, the results obtained from the application of both methods to a selection of test problems are presented in section 5.5, and some conclusions drawn in section 5.6.

5.3 Ordered operator search

More detailed consideration of the geometry of the problem indicates that some selectional restriction on the operators that may be applied at a node should be imposed. If the network at a node is not connected, the list of operators that may be applied at that node is restricted to links which will improve the connectivity, i.e. if such a link is added to the network a path will exist between two vertices between which there is not currently a path. Such a list of operators will be called a connecting operator list. It is ordered by the cheapness of the project costs of the links. The lowest cost link that improves connectivity is first on the

list, the second lowest cost second, and so on.

Otherwise the traffic flows that there would be if the network so far constructed were the one finally used are considered. Let f_{ij} denote the traffic flow there would be from i to j. For each L_{ij} not present in the network for which the addition of p_{ij} to the cost of the network would not violate the budget constraint the quantity

$$(d_{ij}-l_{ij})*f_{ij}/p_{ij} \quad ,$$

where d_{ij} denotes the distance between i and j in the network, is calculated. The pair (i,j) for which this expression is positive and has its highest value is called the <u>focus</u> at the current node. If no such pair exists then a terminal node of the search tree has been reached.

At such a non-terminal node a <u>focal operator list</u> is constructed. This consists of the set of links with the property that the addition to the current network of any one of them would shorten the path from i to j without violating the budget constraint. To make the search a complete search of the state-space an additional operator, the <u>focus banning operator</u>, must be added to the focal operator list. When the focus banning operator is applied, the generated successor state consists of the same network, but with the additional specification that when the focus of this or any descendant state is being determined (i,j) is not a candidate as focus. In practice, however, the focus banning operator can be dispensed with.

Let L_{gh} be a link in a focal operator list. Define

v_{gh} by

$$v_{gh} = (d_{gh} - l_{gh}) * f_{gh} \quad .$$

The links in the focal operator list are ordered in descending order of v_{gh}.

The development of the search tree is based not on an evaluation function but on the use of heuristic information embodied in the ordering of the operators in the operator lists. The form of an iteration in the development of the search tree is:

i) Find the non-terminal node with lowest cost (an undeveloped node has cost 0). This is the current node.

ii) The first operator in the operator list for the current node becomes the current operator and is deleted from the operator list. Add 10 to the cost of the current node.

iii) Construct the new node resulting from the application of the current operator at the current node.

iv) If the new node is terminal, compare the user costs for it with those of the best solution found so far. If they are lower, note the new node as being the best solution found so far. If the new node is not terminal, construct its operator list.

v) If there are fewer than 500 nodes on the tree, return to (i).

For the range of problem sizes considered, experience indicates that after 500 nodes have been constructed a solution will have been found that is either optimal or very nearly optimal. It is to be expected that if larger

problems were considered a larger number of iterations would be necessary.

5.4 Branch-and-bound

The problem also admits a branch-and-bound formulation. Each node of the branch-and-bound tree, other than the root node, is labelled with the inclusion or exclusion of a link, L_{ij}, that might occur in the solution network. Associated with each node is a network class. This is the set of networks which contain the links whose inclusions label nodes on the path from the root node to the given node, and do not contain the links whose exclusions label nodes on the path from the root node to the given node, and also satisfy the budget constraint. At a terminal node the network class has the property that it has a member of which all the members are sub-graphs. That member is the feasible solution of the problem defined by that terminal node.

It is found that solutions to the problem can be efficiently determined by the use of a go-right strategy for the development of the branch-and-bound tree. There are three aspects to the strategy: the branching rule, considerations of connectivity, and the calculation of the bound.

The branching rule uses heuristic information about the problem to select at each iteration the link, whose status (included or excluded) is undecided, that is thought most likely to occur in the optimal solution. Corresponding to each link, L_{ij}, there is a distance, m_{ij}, which is the shortest distance between i and j in the network consisting of all the links except L_{ij}. Let

$\epsilon_{ij} = \max(0, m_{ij} - l_{ij})$. Then for each pair (i,j) $\delta_{ij} = (t_{ij} + t_{ji}) * \epsilon_{ij} / p_{ij}$ is calculated. At each iteration the link with undecided status for which δ_{ij} is largest is selected by the branching rule. Note that the calculation of the δ_{ij}'s need only be done once, at the start of the construction of the tree.

For a given network class, consider the graph formed by the links that are definitely included. This graph may or may not be connected. If it is not, then it consists of a number of separate connected components. A feasible solution is a connected graph, so it is known that any feasible solution descendant from such a node must have added to the list of included links a set of links that connect the presently separate components. Using a result of Kruskal (1956) on minimal spanning trees, it is possible to calculate the minimum possible cost of such a set of links. If this cost added to the cost of the definitely included links exceeds the budget, then the node with which the network class is associated can be marked as undevelopable. This analysis is done for a node when it is being considered for development.

Let x be a network class. Let E_x, I_x, U_x denote respectively the set of links that are definitely excluded in x, the set that are definitely included, and the set that are undecided. If W is a set of links then let C(W) be the user costs for the network they form. When a node labelled with an exclusion and associated with a network class, y, is added to the tree, its lower bound is calculated as $C(I_y \cup U_y)$. When a node labelled with an inclusion and associated with a network class, z,

is added to the tree, the calculation is first done of $r_z = B - \sum_{L_{ij} \in I_z} p_{ij}$, that is, that part of the budget not yet committed. If there is some link L_{gh} in U such that $p_{gh} > r_z$, then the set $A = \{L_{gh} \mid L_{gh} \in U_z, p_{gh} \leq r_z\}$ is determined and the lower bound of the node is calculated as $C(I_z \cup A_z)$.

Let s_{gh} be the shortest distance from g to h in the network formed by $(I_z \cup U_z) - \{L_{gh}\}$. Let $\eta_{gh} = \max(0, s_{gh} - l_{gh})$. Let V be any subset of U_z, then

$$C(I_z \cup V) \geq C(I_z \cup U_z) + \sum_{L_{gh} \in U_z - V} (t_{gh} + t_{hg}) \eta_{gh}$$

since at least the additional costs over $C(I_z \cup U_z)$ are incurred that traffic between g and h must follow some path other than the direct link between them. This inequality can be rewritten as

$$\text{(I)} \quad C(I_z \cup V) \geq C(I_z \cup U_z) + \sum_{L_{gh} \in U_z} (t_{gh} + t_{hg}) \eta_{gh} - \sum_{L_{gh} \in V} (t_{gh} + t_{hg}) \eta_{gh}$$

Now, any terminal node descendant from the node with which z is associated will define a network whose set of links is $I_z \cup V_q$ where $\sum_{L_{gh} \in V_q} p_{gh} \leq r_z$ and $V_q \subseteq U_z$. So for the node with which z is associated

$$\min_{\{V_q \mid \sum_{L_{gh} \in V_q} p_{gh} \leq r_z\}} C(I_z \cup V_q)$$

is the best possible lower bound. Using inequality I above it can be seen that a weaker lower bound can be calculated as

$$\{V_q \mid \sum_{L_{gh} \in V_q}^{\min} p_{gh} \leq r_z\} \left[C(I_z \cup U_z) + \sum_{L_{gh} \in U_z} (t_{gh} + t_{hg}) \eta_{gh} - \sum_{L_{gh} \in V_q} (t_{gh} + t_{hg}) \eta_{gh} \right]$$

This can be rewritten as

$$C(I_z \cup U_z) + \sum_{L_{gh} \in U_z} (t_{gh} + t_{hg}) \eta_{gh} - \{V_q \mid \sum_{L_{gh} \in V_q}^{\max} p_{gh} \leq r_z\} \sum_{L_{gh} \in V_q} (t_{gh} + t_{hg}) \eta_{gh}$$

The finding of an upper bound for this maximum can be done by solving a simple knapsack problem (Garfinkel and Nemhauser, 1972). Using this upper bound in place of the maximum gives a lower bound for the node. As the η_{gh}'s, and the related quantities whose values are needed in the solution of the knapsack problem, calculated at one node will be the same for the successor node labelled with an inclusion, this lower bound can be economically calculated for nodes labelled with inclusions, and this is in fact done for nodes other than those the calculations of whose lower bound has already been described.

5.5 Results

Programs using the two strategies were written in Algol 68-R (Woodward and Bond, 1972) and tested against the four sets of data shown in table 5.5.1. BFW531 is the graph described by Boyce, Farhi and Weischedel (1973); it has all the t_{ij}'s equal to 1 and the p_{ij}'s equal to the l_{ij}'s. BFWD10 has the same l_{ij}'s, but the t_{ij}'s are random integers in the range 1 to 10, and

$p_{ij}=(10+\rho_{ij})*l_{ij}$ where the ρ_{ij}'s are random integers in the range 0 to 9. ND101 is another 10-vertex graph, based on a different set of random points, with the t_{ij}'s and p_{ij}'s derived as before. ND201 is a similarly constructed 20-vertex graph.

The results of these tests are shown in table 5.5.2. The column "budget fraction" contains the budget constraint expressed as a fraction (approximate in the case of BFW531) of the total cost of the links of the graph. "Budget" contains the budget constraint, "solution" the best solution found by the program, "solution node no." the number of nodes on the tree at the time the best solution was found, and "total no. of nodes" the number of nodes on the tree at the time the program terminated or was abandoned.

The column "proved?" for the branch-and bound method contains "Y" if the program successfully terminated, "N" if the program was abandoned, and "NN" if the best solution that had been produced by the program at the time it was abandoned is known not to be optimal. Abandonment occurred if the job containing the program had consumed more than 30 minutes CPU time. Asterisks occur against the number of nodes for the cases where abandonment occurred when other programs were run in the same job.

The column "diff from B&B" contains the difference between the best solution found by the state-space search and the best solution found by the branch-and-bound method, expressed in absolute terms and as a percentage. For comparison the results obtained by Boyce, Farhi, and

BFW531/BFWD10

arc lengths

i \ j	1	2	3	4	5	6	7	8	9	10
1	0	470	519	236	369	94	383	371	300	547
2	470	0	982	281	143	488	851	98	296	393
3	519	982	0	750	864	539	204	885	801	934
4	236	281	750	0	246	220	594	191	81	523
5	369	143	864	246	0	410	750	95	298	297
6	94	488	539	220	410	0	374	391	262	622
7	383	851	204	594	750	374	0	752	633	881
8	371	98	885	191	95	391	752	0	224	389
9	300	296	801	81	298	262	633	224	0	588
10	547	393	934	523	297	622	881	389	588	0

traffic flows

i \ j	1	2	3	4	5	6	7	8	9	10
1	0	10	1	7	4	2	2	8	2	3
2	7	0	4	9	10	4	4	9	10	3
3	9	4	0	10	4	9	3	1	3	8
4	2	2	1	0	7	8	1	5	6	9
5	9	3	2	9	0	2	1	8	2	7
6	4	5	10	4	7	0	9	7	3	6
7	4	4	9	1	9	9	0	9	8	9
8	1	6	2	3	9	1	10	0	1	9
9	10	4	5	10	6	5	10	4	0	5
10	3	2	7	10	1	7	7	2	10	0

project costs

i \ j	1	2	3	4	5	6	7	8	9	10
1	0	4700	8304	2832	5904	1034	3830	6678	5700	5470
2	4700	0	12766	2810	1573	8296	19212	1274	3552	6681
3	8304	12766	0	11250	13824	7546	3876	12390	15219	9340
4	2832	2810	11250	0	3690	3740	8316	2674	1134	9414
5	5904	1573	13824	3690	0	6970	10500	1235	5066	3564
6	1034	8296	7546	3740	6970	0	5610	4692	2620	7464
7	3830	10212	3876	8316	10500	5610	0	12032	7596	10572
8	6678	1274	12390	2674	1235	4692	12032	0	3136	5057
9	5700	3552	15219	1134	5066	2620	7596	3136	0	10584
10	5470	6681	9340	9414	3564	7464	10572	5057	10584	0

Table 5.5.1

Test data

ND101

arc lengths

i \ j	1	2	3	4	5	6	7	8	9	10
1	0	505	716	927	615	441	674	106	848	480
2	505	0	213	438	160	541	441	542	403	474
3	716	213	0	238	151	680	461	747	257	593
4	927	438	238	0	312	798	490	942	153	699
5	615	160	151	312	0	530	326	630	247	442
6	441	541	680	798	530	0	356	361	665	103
7	674	441	461	490	326	356	0	636	341	255
8	106	542	747	942	630	361	636	0	849	416
9	848	403	257	153	247	665	341	849	0	563
10	480	474	593	699	442	103	255	416	563	0

traffic flows

i \ j	1	2	3	4	5	6	7	8	9	10
1	0	5	8	5	4	8	9	7	2	7
2	7	0	6	2	5	7	6	6	7	2
3	3	9	0	5	9	10	4	2	5	8
4	10	2	10	0	9	8	2	4	10	5
5	4	5	5	8	0	10	3	7	8	9
6	6	9	8	8	2	0	2	1	10	5
7	1	9	3	5	2	8	0	3	3	9
8	3	6	10	10	2	5	9	0	8	6
9	1	4	10	10	2	8	2	8	0	5
10	10	1	8	8	1	1	8	9	4	0

project costs

i \ j	1	2	3	4	5	6	7	8	9	10
1	0	8080	10024	12051	7380	4410	9436	1378	11024	5280
2	8080	0	4047	4818	1920	9738	6615	8672	6045	6636
3	10024	4047	0	4284	2114	8840	8759	11952	3598	8895
4	12051	4818	4284	0	4368	12768	7350	10362	2907	11883
5	7380	1920	2114	4368	0	6890	5542	8190	3211	6188
6	4410	9738	8840	12768	6890	0	3916	5054	12635	1236
7	9436	6615	8759	7350	5542	3916	0	8904	3410	3825
8	1378	8672	11952	10362	8190	5054	8904	0	13584	7904
9	11024	6045	3598	2907	3211	12635	3410	13584	0	9008
10	5280	6636	8895	11883	6188	1236	3825	7904	9008	0

ND201

arc lengths

j	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	0	729	1978	953	401	1054	988	481	810	73	783	149	744	652	530	1148	391	922	1006	969
2	729	0	407	744	438	500	259	345	205	801	178	629	184	102	206	541	673	196	588	380
3	1078	407	0	652	842	214	257	751	521	1145	514	952	334	442	560	171	895	321	398	160
4	953	744	652	0	979	451	824	940	948	990	922	806	562	672	693	554	587	836	263	501
5	401	438	842	979	0	896	679	100	443	470	422	389	544	409	314	960	594	601	923	784
6	1054	500	214	451	896	0	444	818	665	1114	649	914	352	490	585	114	802	488	190	120
7	988	259	257	824	679	444	0	580	285	1060	286	886	322	345	462	427	902	81	601	341
8	481	345	751	940	100	818	580	0	344	552	322	444	467	328	252	876	616	502	860	703
9	810	205	521	948	443	665	285	344	0	883	32	741	386	294	354	681	837	206	779	546
10	73	801	1145	990	470	1114	1060	552	883	0	856	200	811	722	600	1210	413	994	1058	1032
11	783	178	514	922	422	649	286	322	32	856	0	712	361	264	323	670	805	206	758	530
12	149	629	952	806	389	914	886	444	741	200	712	0	619	542	424	1011	260	826	860	835
13	744	184	334	562	544	352	322	467	386	811	361	619	0	140	237	421	589	301	410	242
14	652	102	442	672	409	490	345	328	294	722	264	542	140	0	122	552	571	292	546	376
15	530	206	560	693	314	585	462	252	354	600	323	424	237	122	0	658	483	403	610	479
16	1148	541	171	554	960	114	427	876	681	1210	670	1011	421	552	658	0	911	487	292	180
17	391	673	895	587	594	802	902	616	837	413	805	260	589	571	483	911	0	860	700	752
18	922	196	321	836	601	488	81	502	206	994	206	826	301	292	403	487	860	0	630	376
19	1006	588	398	263	923	190	601	860	779	1058	758	860	410	546	610	292	700	630	0	262
20	969	380	160	501	784	120	341	703	546	1032	530	835	242	376	479	180	752	376	262	0

ND201

traffic flows

i \ j	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	0	3	6	4	5	1	4	6	10	6	19	9	4	7	10	8	5	5	2	9
2	10	0	2	3	8	2	4	9	8	4	3	1	6	5	1	6	10	9	1	1
3	10	9	0	3	6	10	7	10	8	3	4	2	5	6	4	3	7	4	7	6
4	4	8	6	0	1	9	10	7	4	8	4	3	10	2	4	8	5	2	8	3
5	3	3	7	5	0	5	9	9	7	5	1	6	8	5	9	2	9	8	2	3
6	3	4	10	8	1	0	3	6	9	1	7	3	9	5	8	2	2	8	6	6
7	10	9	1	2	8	2	0	7	4	10	1	5	5	9	3	3	5	5	2	10
8	4	5	7	2	6	10	4	0	7	8	4	4	2	5	3	6	9	5	10	2
9	7	2	2	10	8	7	1	5	0	10	8	2	8	9	10	9	2	3	8	4
10	3	5	3	4	10	5	1	9	7	0	3	10	2	5	1	6	2	6	7	4
11	4	1	3	5	7	7	1	3	6	3	0	7	4	1	8	8	8	10	7	3
12	2	2	5	3	10	1	4	6	6	10	8	0	9	10	3	7	1	8	6	2
13	2	7	5	8	6	10	3	2	8	2	8	8	0	8	4	10	2	6	7	7
14	3	2	10	7	2	3	3	2	8	2	6	2	8	0	9	9	4	5	10	1
15	4	6	4	7	4	2	2	6	8	9	5	9	9	10	0	6	10	9	6	4
16	2	10	3	8	3	7	9	7	5	1	6	3	7	8	4	0	9	3	9	7
17	3	9	3	10	6	6	8	6	5	8	3	5	9	4	6	4	0	9	1	6
18	7	5	7	9	6	2	5	4	7	10	7	5	7	8	1	8	10	0	9	5
19	4	10	2	6	5	1	8	5	2	8	6	2	8	4	5	10	7	7	0	9
20	8	7	2	3	8	2	4	10	6	4	4	2	4	7	7	5	6	10	4	0

ND201

project costs

j / i	1	2	3	4	5	6	7	8	9	10
1	0	9477	17248	9530	6817	20026	10868	6253	8910	1022
2	9477	0	6512	12648	5256	9000	4662	5865	3075	8010
3	17248	6512	0	9780	9262	2782	3855	8261	5731	14885
4	9530	12648	9780	0	16643	5412	12360	16920	16116	11880
5	6817	5256	9262	16643	0	14336	9506	1700	4873	7520
6	20026	9000	2782	5412	14336	0	7548	8180	9975	13368
7	10868	4662	3855	12360	9506	7548	0	5800	5415	14840
8	6253	5865	8261	16920	1700	8180	5800	0	5160	8832
9	8910	3075	5731	16116	4873	9975	5415	5160	0	10596
10	1022	8010	14885	11880	7520	13368	14840	8832	10596	0
11	7830	2314	5654	15674	6752	8437	5434	5796	352	8560
12	1937	11322	11424	12896	3890	10054	11518	5328	12597	3200
13	10416	2024	5678	7306	7072	6336	3542	5137	5404	12976
14	6520	1020	7956	7392	6135	7840	5175	5248	5586	13718
15	5830	2678	9520	9009	3454	7020	7392	4032	6372	10200
16	18638	10279	2736	9972	17280	2166	5551	8760	11577	22990
17	3910	7403	16110	9392	8910	12832	17138	8008	14229	5369
18	11064	3136	4494	15048	10217	5856	891	6526	3502	10934
19	10060	6820	5970	2893	16614	3610	9616	16340	9348	15870
20	11628	6840	1920	9018	7840	1320	5115	12654	6552	12384

ND201

project costs

i \ j	11	12	13	14	15	16	17	18	19	20
1	7830	1937	10416	6520	5830	18368	3910	11064	10060	11628
2	2314	11322	2024	1020	2678	10279	7403	3136	8820	6840
3	5654	11424	5678	7956	9520	2736	16110	4494	5970	1920
4	15674	12896	7306	7392	9009	9972	9392	15048	2893	9018
5	6752	3890	7072	6135	3454	17280	8910	10217	16614	7840
6	8437	10054	6336	7840	7020	2166	12832	5856	3610	1320
7	5434	11518	3542	5175	7392	5551	17138	891	9616	5115
8	5796	5328	5137	5248	4032	8760	8008	6526	16340	12654
9	352	12597	5404	5586	6372	11577	14229	3502	9348	6552
10	8560	3200	12976	13718	10200	22990	5369	10934	15870	12384
11	0	12816	6498	3696	6137	8040	8855	3708	8338	6360
12	12816	0	9904	5962	7208	12132	2860	8260	9460	14195
13	6498	9904	0	2660	4266	7578	10602	3010	4510	4114
14	3696	5962	2660	0	1342	6072	7423	3504	7098	5264
15	6137	7208	4266	1342	0	10528	6279	4433	11590	8622
16	8040	12132	7578	6072	10528	0	10932	8279	3796	2700
17	8855	2860	10602	7423	6279	10932	0	14620	9100	9776
18	3708	8260	3010	3504	4433	8279	14620	0	8190	4512
19	8338	9460	4510	7098	11590	3796	9100	8190	0	3144
20	6360	14195	4114	5264	8622	2700	9776	4512	3144	0

Table 5.5.2

Results of test runs

BFW531

budget fraction	budget	solution	branch-and-bound solution node no.	branch-and-bound total no. of nodes	proved?	state-space search solution	state-space search solution node no.	state-space search diff from B&B
0.9	18000	41234	82	82	Y	41234	41	0
0.8	16000	41242	76	78	Y	41242	38	0
0.7	14000	41262	78	106	Y	41266	35	4=.01%
0.6	12000	41306	64	98	Y	41306	32	0
0.5	10000	41408	58	158	Y	41408	194	0
0.4	8000	41632	72	252	Y	41660	161	28=.07%
0.3	6000	42144	40	1964	Y	42144	131	0
0.2	4000	43300	1182	11938*	N	43300	122	0
0.1	2000	47708	74	8702	Y	47708	145	0

Boyce's method

budget fraction	budget	solution	solution iteration no.	no. of iterations	proved?
0.9	18000	41234	1	1	Y
0.8	16000	41242	1	4	Y
0.7	14000	41262	1	12	Y
0.6	12000	41306	1	21	Y
0.5	10000	41408	1	138	Y
0.4	8000	41632	2	1329	Y
0.3	6000	42144	1	6000	N
0.2	4000	43300	3541	6000	N
0.1	2000	47708	1	8000	N

BFWD10

budget fraction	budget	solution	branch-and-bound solution node no.	branch-and-bound total no. of nodes	branch-and-bound proved?	state-space search solution	state-space search solution node no.	state-space search diff from B&B
0.9	261655	226069	84	84	Y	226069	42	0
0.8	232583	226086	78	82	Y	226086	291	0
0.7	203511	226194	78	130	Y	226215	37	21=.01%
0.6	174439	226357	66	120	Y	226357	302	0
0.5	145367	226694	60	124	Y	226694	271	0
0.4	116295	227760	132	398	Y	227760	225	0
0.3	87223	230208	68	1476	Y	230382	485	174=.08%
0.2	58151	237193	4656	13412*	N	237883	142	690=.29%
0.1	29074	259156	528	15203	N	259156	255	0

ND101

budget fraction	budget	solution	branch-and-bound solution node no.	branch-and-bound total no. of nodes	branch-and-bound proved?	state-space search solution	state-space search solution node no.	state-space search diff from B&B
0.9	283618	259356	84	88	Y	259356	41	0
0.8	252105	259416	82	104	Y	259424	38	8=.003%
0.7	220592	259550	70	96	Y	259550	35	0
0.6	189079	259854	62	90	Y	259854	31	0
0.5	157566	261106	502	862	Y	261159	58	53=.02%
0.4	126053	262851	690	2656	Y	262851	25	0
0.3	94540	267119	6856	10946*	N	267810	21	691=.26%
0.2	63027	-	-	-	-	282516	482	-
0.1	31514	331167	1906	16316	N	331167	64	0

ND201

budget fraction	budget	branch-and-bound solution	branch-and-bound solution node no.	branch-and-bound total no. of nodes	branch-and-bound proved?	state-space search solution	state-space search solution node no.	diff from B&B
0.9	1379795	1184864	352	352	Y	1184864	169	0
0.8	1226485	1184885	336	438	Y	1184983	165	8=.0007%
0.7	1073175	1185085	496	3403*	N	1185112	151	27=.002%
0.6	919865	-	-	-	-	1185465	136	-
0.5	766555	-	-	-	-	1186294	120	-
0.4	613245	-	-	-	-	1188062	311	-
0.3	459935	-	-	-	-	1192647	254	-
0.2	306225	-	-	-	-	1204208	197	-
0.1	153315	1268663	1572	4142	NN	1251141	280	-17522

Weischedel (1973) for graph BFW531 are included in the tabulation of results for this graph.

5.6 Conclusions

The methods described here represent a substantial improvement on those previously available in terms of efficiency. The state-space method, although not guaranteeing the optimality of the solution produced, for the problems considered always yielded solutions that are nearly optimal, the maximum error being .29%. There are some indications (see the 0.1 level result for ND201) that when the complexity of the problem causes the branch-and-bound method to break down the state-space method will continue to produce acceptable results.

It is of interest to note that the difference between the user costs at the 0.1 level and the 0.9 level ranges from 28% (ND101) down to 5% (ND201). This suggests that dense networks are not cost effective and that in practical problems methods appropriate to the finding of optimal sparse networks would be important.

Chapter 6 An abstract 2-dimensional trim-loss problem

6.1 Introduction

6.1.1 Statement of the problem

Material is held in rectangular stock sheets from which an order list of smaller rectangular pieces must be cut. All the stock sheets have the same dimensions. The dimension of the longer side is called the length and that of the shorter side the breadth. A similar terminology is applied to the pieces. All dimensions are small integers. For each piece there are three integers, the length, the breadth, and the number ordered. The totality of these integers defines the order list.

A cutting pattern is a set of instructions as to how a sheet is to be cut. There is no restriction on the type of cutting that may be specified. However, it has been found that for the type of data under consideration satisfactory solutions can be found in which much of the cutting is guillotine cutting.

6.1.2 Choice of method

There is a resemblance in this problem to the abstract 1½-dimensional trim-loss problem, and an attempt was made to apply a similar ordered operator search as a solution technique. This attempt, however, revealed that a more significant feature of the geometry of the present problem is the large degree of repetition in the order list. The implication of this repetition is that a number of regular features will recur during the construction of a solution. Consideration of these features suggests a classification of the types of subgoal that would be created if problem

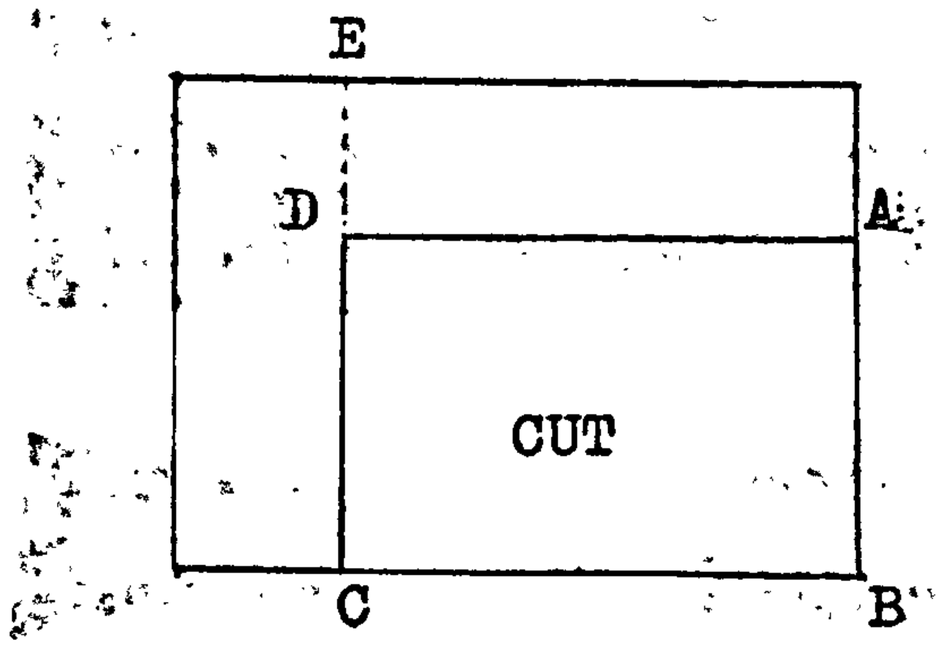
reduction were used for the solution of the problem.

The regular features are:

i) Given an order list, a type of piece can be identified as being the one of which it is most important that instructions to cut at least one should be included in the instructions for cutting the next sheet.

ii) A number of such pieces, similarly oriented and juxtaposed, can be cut from a stock sheet either leaving an L-shaped or rectangular fragment of the sheet uncut, or consuming the entire sheet.

iii) An L-shaped fragment can be divided into two rectangular fragments (see figure 6.1.1).



Division of an L-shaped fragment

Figure 6.1.1

iv) A number of similarly oriented and juxtaposed pieces can be cut from a rectangular fragment, either consuming it completely, or leaving an L-shaped or rectangular fragment uncut.

The related types of subgoal are:

i) specify the set of juxtaposed and similarly oriented most important pieces that is to be cut from the next stock sheet,

ii) specify the way in which an L-shaped fragment is to

be divided into rectangular fragments,

iii) specify the way in which a rectangular fragment is to be divided into pieces.

The decision having been made to make use of a problem reduction method, the process of forming a cutting pattern under it can be regarded as a search in the subgoal tree for a set of subgoal solutions which together define a cutting pattern. A simplified version of the subgoal tree that will be set up under this formulation is shown in figure 6.1.2.

The geometry of the problem suggests both the possible reductions of the subgoals and an ordering on these reductions. It is apparent that the overall goal of constructing a set of cutting patterns must be reduced into a sequence of subgoals of designing cutting patterns in which each pattern is appropriate to an order list produced from the original order list by the removal of orders for pieces whose cutting is specified by preceding patterns in the sequence. Preferred reduction within non-backtracking search is therefore indicated as a suitable method.

The basic method can, for this problem, be enhanced by carrying forward information accumulated during attempts at the solution of one subgoal for use in attempts at the solution of others. This is done by maintaining lists describing the action taken when an L-shaped or rectangular fragment was encountered during the preparation of earlier cutting patterns, or earlier in the preparation of the current cutting pattern.

The cutting patterns produced by application of the

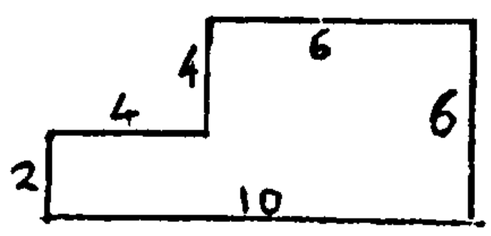
Initial order list
10 of 4 x 4
25 of 10 x 1
25 of 8 x 1
15 of 6 x 1

Sheet size 10 x 6

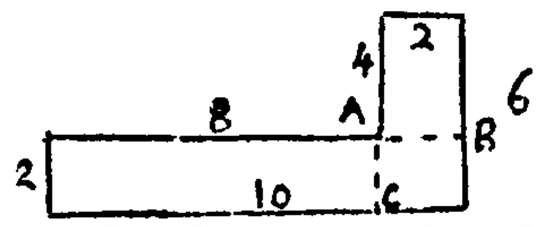
Notation: Q-definition of set of subgoals, solution of one or more of which is necessary for solution of current goal
A- alternative selected from Q of parent
S-explicit solution of subgoal

Q: Which sub-tessellation with 4 x 4 pieces should be used?

A: One 4 x 4 piece, leaving L-shape



A: Two 4 x 4 pieces, leaving L-shape



Q: Should L-shape be divided along AB or AC?

A: Along AB
Q: How may 10 x 2 fragment be divided and how may 4 x 2 fragment be divided?

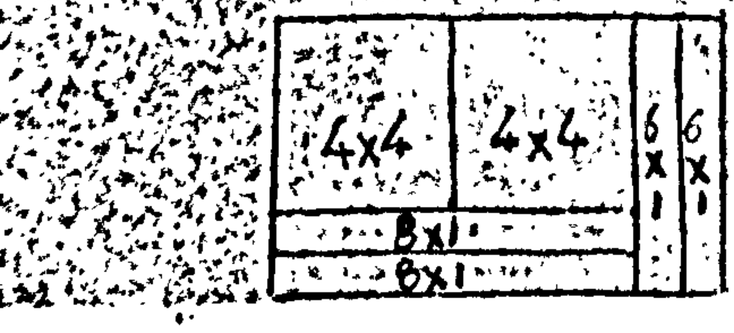
A: 10x2 fragment
S: Two 10x1 pieces

A: 4x2 fragment
S: NO SOLUTION

A: Along AC
Q: How may 8 x 2 fragment be divided and how may 6 x 2 fragment be divided?

A:8x2 fragment
S: Two 8x1 pieces

A:6x2 fragment
S: Two 6x1 pieces



Resultant cutting pattern, used 5 times leaving order list
0 of 4 x 4
25 of 10 x 1
15 of 8 x 1
5 of 6 x 1

Simplified subgoal tree

Figure 6.1.2

problem reduction method will always be ones in which all the cutting is guillotine cutting. There remain, however, a small number of instances where it is desirable to search for patterns involving non-guillotine cutting and here ordered operator search is appropriate.

6.1.3 Structure of the description of the solution method

The method of solution adopted will be described in terms of a program that produces instructions as to how a required set of pieces should be cut from a set of stock sheets. The program has two main routines, REGULAR which uses problem reduction based on the formulation above to produce cutting patterns in which all the cutting is guillotine cutting, and IRREGULAR which uses an ordered operator search based on the method applied to the abstract 1½-dimensional trim-loss problem to produce cutting patterns that may involve non-guillotine cutting.

The way in which these routines are connected is described in section 6.2. They both operate on an order list on which a sequencing has been imposed. The way in which this sequencing is determined is described in sub-section 6.2.1. Control must be passed between the routines, and decisions made as to the maximum level of trim-loss (scrap) which they should be allowed to include in a cutting pattern at a given time. These matters are discussed in sub-section 6.2.2.

Section 6.3 describes the routine REGULAR, sub-section 6.3.1 its general structure, 6.3.2 the way in which the development of its subgoal tree is managed, 6.3.3 routines for handling "most important pieces", and 6.3.4 the "learning lists" that accumulate information

acquired by the routines for handling rectangular and L-shaped fragments described in 6.3.5.

Section 6.4 describes the routine IRREGULAR, sub-section 6.4.1 its general structure, and 6.4.2 the way in which it determines sub-patterns with a "spiral" structure. The results produced by the program are presented in section 6.5.

6.1.4 Definitions

A piece is tessellating if its length divides the length of the sheet and its breadth divides the breadth of the sheet, or if its breadth divides the length of the sheet and its length divides the breadth of the sheet.

A piece is semi-tessellating if it is not tessellating and one of its dimensions divides one of the dimensions of the sheet.

A piece is non-tessellating if it is neither tessellating nor semi-tessellating.

A sub-tessellation is the juxtaposition of a number of similarly oriented identical pieces to form a rectangular area that can be included in a sheet. A sub-tessellation is maximal if no larger sub-tessellation could be formed with the given pieces in the given area. Figure 6.2.1 shows a maximal sub-tessellation of a 20x20 sheet consisting of 7x4 pieces.

A partial cutting pattern is an incomplete set of instructions as to how a sheet is to be cut. A global step is the design of a cutting pattern.

6.2 General strategy

6.2.1 Urgency of pieces

In order to keep the resources required by the

program to an acceptable level the degree of backtracking is restricted. Once a usable cutting pattern has been designed the maximum number of times it can be used is determined. The total numbers of pieces resulting from such cuttings are then subtracted from the order list and the resultant order list used in the next global step.

The pieces can be ranked in terms of the urgency of finding a cutting pattern that includes them. The urgency of disposing of a piece that is not tessellating is greater than that for a tessellating piece, since if only one type of piece is left at the last global step waste is unavoidable if it is not tessellating. The problem of using the area remaining in a sheet after a sub-tessellation with a semi-tessellating piece has been cut from it can be more easily solved than the corresponding problem for a non-tessellating piece. So the urgency of disposing of the semi-tessellating piece is less. Within each category, the smaller the piece the less the urgency of disposing of it.

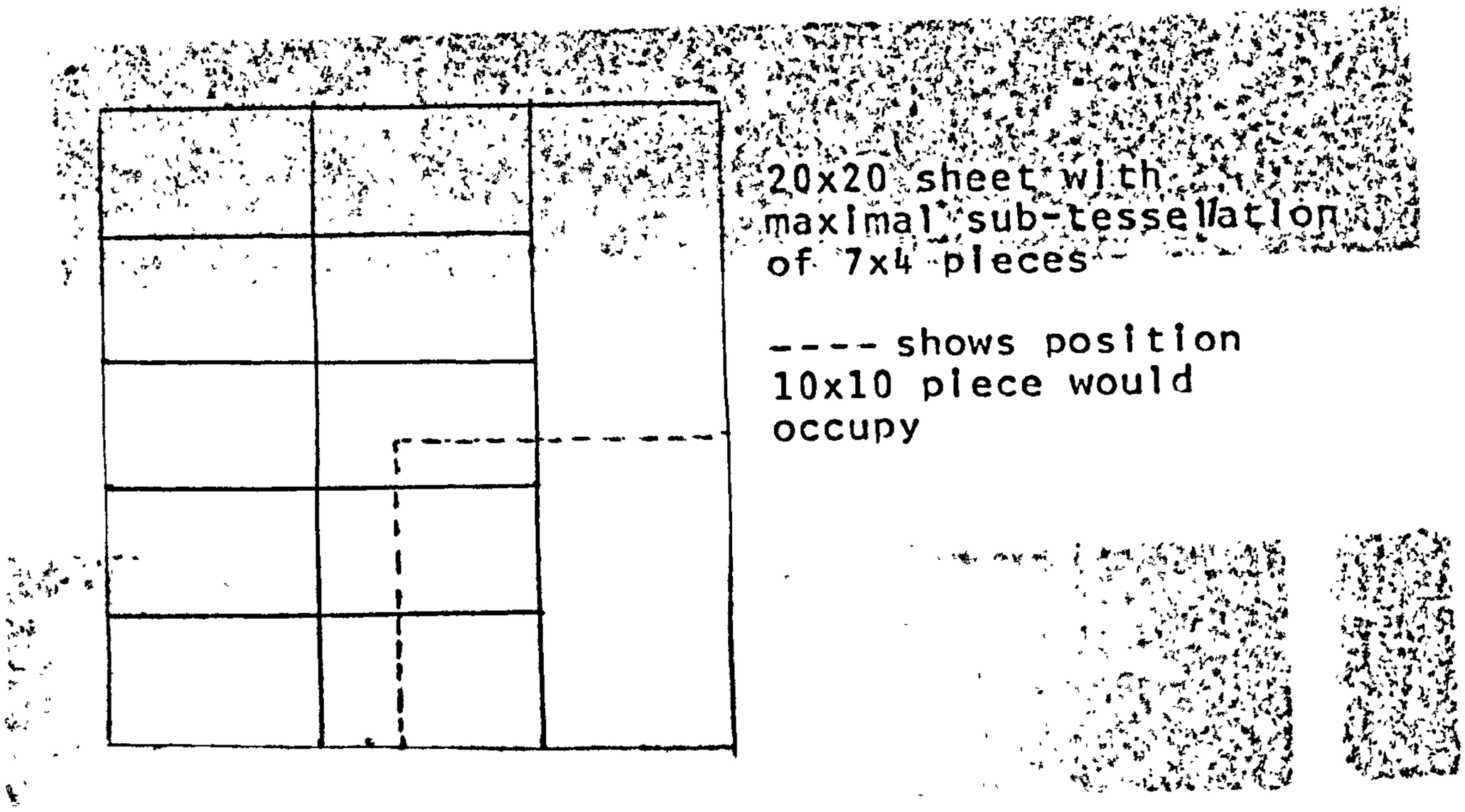
When the problem of using the area remaining in a sheet after a sub-tessellation with a piece that is not tessellating is being considered, it is in general not the case that it will be specified that a piece with area equal to a quarter of the area of the sheet should be cut from it. In figure 6.2.1 it can be seen that the specification of sub-tessellations including large numbers of 7x4 pieces precludes the subsequent cutting of a 10x10 piece, and also that the fragment remaining after the cutting of a smaller sub-tessellation of 7x4 pieces and a 10x10 piece is potentially difficult to divide.

Suppose instructions to cut one or more pieces of quarter sheet area were included in a cutting pattern that would result in trim-loss. Then it is in general the case that the amount of trim-loss per unit area of pieces that are not tessellating in that cutting pattern will be higher than that for a cutting pattern that does not include instructions to cut pieces of quarter sheet area whilst still including instructions to cut the pieces that are not tessellating. For example, the maximum number of 7x4 pieces that can be cut from a 20x20 sheet with guillotine cutting is 12 (the sub-tessellation of 10 shown in figure 6.2.1 plus 2 with the other orientation) giving a trim-loss per unit area of pieces that are not tessellating of .19, whereas if a 10x10 piece is cut from the sheet the maximum number of 7x4 pieces that can be cut from the remainder is 8 (the 7 in figure 6.2.1 not overlapped by the 10x10 piece plus 1 with the other orientation) giving a trim-loss per unit area of pieces that are not tessellating of .53.

For these reasons the urgency of a piece P is calculated as:

```
area(P)+if area(P)*4 = sheet area then 500
       elseif non-tessellating(P) then 500
       elseif semi-tessellating(P) then 250
       else 0
       close
```

The order list is sequenced in descending order of the urgencies to which each order relates.



Effect of a quarter sheet piece

Figure 6.2.1

At each global step the goal is to find a cutting pattern which includes pieces of a specified type. Call this type T*. At nearly every global step T* will be the highest sequenced type still in the order list. The exceptional case occurs when the total area of pieces that are not tessellating and are still to be cut is less than the area of a sheet and there is at least one tessellating type such that the total area of pieces of that type not yet cut is not at least equal to the area of a sheet. In this case T* is the first such tessellating type in the order list.

The reason for introducing the special case in the choice of T* relates to the special condition that holds when the last sheets are cut. It is known that the last cutting pattern will normally involve trim-loss, since normally the total area of the remaining orders is not equal to the area of a sheet. The cutting of the last

pieces that are not tessellating is likely to involve trim-loss. The aim is to defer this cutting until use can be made of the known necessary trim-loss in the end condition.

6.2.2 Cost control

Two types of cost are incurred by the program, the cost of the computation and the cost of the trim-loss resulting from the set of cutting patterns produced. The routines do not attempt to determine the minimum amount of scrap that is essential for the next cutting pattern to be formed. Instead they operate on an order list to which a number of pieces of unit area, which will be scrap in the determined cutting pattern, have been added. The number of such pieces in the order list will be the maximum amount of scrap per sheet that is to be considered allowable in the current search for a cutting pattern. If a cutting pattern has not been formed by a routine after it has used 3 minutes of CPU time it will report failure. When a routine reports failure the alternatives available are:

i) invoke the other routine,

ii) increase the amount of scrap per sheet allowable,

iii) change the urgencies of the pieces,

iv) some combination of (i) - (iii).

The choice from these alternatives is made in such a way as to provide the balance between the two types of cost.

An explanation is necessary for (iii) above. The urgencies of the pieces affect the decisions taken inside REGULAR. Once it is clear that scrap will be necessary the justification for assigning a low urgency to

tessellating pieces no longer holds, and the change in urgencies may allow REGULAR to find patterns, possibly not involving scrap, that it would not otherwise determine.

In certain circumstances the need for the introduction of scrap can be easily established. Three such cases are identified:

i) If the total area remaining to be cut is less than that of one sheet then sufficient scrap must be introduced to make the total area up to that of one sheet.

ii) If the total area of pieces that are not tessellating is less than that of one sheet and this total is not divisible by the highest common factor of the areas of the uncut tessellating pieces then sufficient scrap must be introduced to give the total this divisibility property.

iii) If the sheet area is not divisible by the highest common factor of the areas of the uncut pieces then an amount of scrap equal to the remainder of this division must be introduced.

The function ESTABLISH-SCRAP-PARAMETERS returns two values which are used when additional scrap is introduced into the order list. INITIAL-SCRAP is the amount of scrap to be introduced initially. It is calculated in the following way:

i) Initialize INITIAL-SCRAP to the area of a sheet. Look at the order for T*.

ii) Subtract from INITIAL-SCRAP the area of the largest possible number of pieces of the current type. This

is either the largest number which does not give a negative result to the subtraction, or, if it is smaller, the number remaining to be cut.

iii) Look at the next order that has not been completed (number required not zero) and that is for pieces whose area is not greater than the current value of INITIAL-SCRAP. If any such order exists, go to (ii).

iv) Find the smallest number of sheets whose total area is not less than the total area of the pieces still to be cut and the difference between these two totals. Hence calculate the average scrap per sheet that is inevitable. INITIAL-SCRAP is the greater of this value and the one previously calculated.

SUBSEQUENT-SCRAP parameterizes the amount of scrap to be introduced subsequently. It is the larger of the area of a piece of the smallest area for which the number required is not zero and the result of dividing the area of a sheet by the number of sheets that must necessarily be cut.

It is now possible to consider how the control function of the program invokes REGULAR and IRREGULAR, and the action it takes when they report failure. The sequence of events is:

i) Establish the order list with urgencies calculated as in 6.2.1.

ii) Determine T* and set TOTAL-SCRAP, the amount of scrap to be added to the order list, to the amount it is known must be introduced.

iii) If TOTAL-SCRAP is not zero, determine SUBSEQUENT-SCRAP and go to (vi).

iv) Call REGULAR. If a cutting pattern is found, go to (xi).

v) Call ESTABLISH-SCRAP-PARAMETERS. If INITIAL-SCRAP is not greater than TOTAL-SCRAP, go to (ix), otherwise make TOTAL-SCRAP equal to INITIAL-SCRAP.

vi) Make the urgency of each piece equal to 250 + its area.

vii) Call REGULAR. If a cutting pattern is found, go to (xi).

viii) Call IRREGULAR. If a cutting pattern is found, go to (xi).

ix) If SUBSEQUENT-SCRAP is greater than TOTAL-SCRAP, make TOTAL-SCRAP equal to SUBSEQUENT-SCRAP and go to (vii).

x) Double SUBSEQUENT-SCRAP. Go to (ix).

xi) Determine the maximum number of times the cutting pattern can be used without causing an excess quantity of some piece to be cut. Print cutting instructions and update the order list.

xii) Determine T* and the amount of scrap (MAXIMUM-SCRAP) that must be introduced for the revised order list. If all the orders have been satisfied, stop.

xiii) If the new T* differs from the old one then:

a) if MAXIMUM-SCRAP is zero and TOTAL-SCRAP is not zero, restore the urgencies of the pieces to their original values,

b) set TOTAL-SCRAP equal to MAXIMUM-SCRAP and go to (iii).

xiv) The amount of scrap in the order list will be the larger of MAXIMUM-SCRAP and the amount of scrap in the previously generated cutting pattern.

xv) If this amount is zero and TOTAL-SCRAP is not zero, restore the urgencies of the pieces to their original values. Set TOTAL-SCRAP to this amount.

xvi) If the last cutting pattern was found by IRREGULAR, go to (viii). Otherwise go to (iii).

6.3 The routine REGULAR

6.3.1 Structure of the routine

The principal data structure maintained by the routine is a tree of subgoals. The label at each node includes:

i) The name of a function that will either solve the subgoal or set up additional subgoals.

ii) A marker that indicates the status of the node as "active", "suspended", or "inactive".

iii) A cost used to determine which subgoal should be considered next.

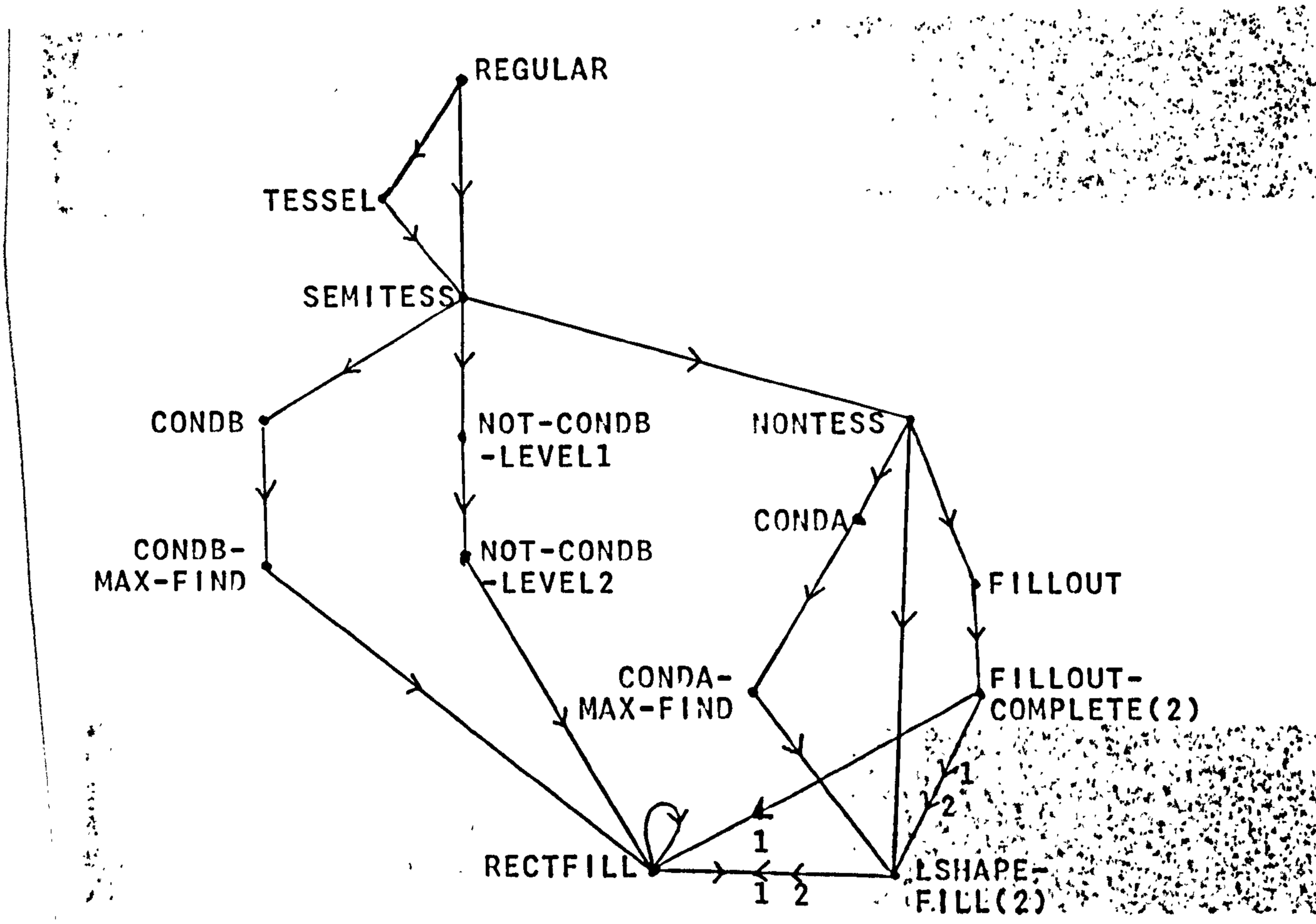
The functions whose names occur in node labels can be divided into two sets. Those in the first set are concerned specifically with the cutting of pieces of type T*; they can be further divided according to T*:

i) T* tessellating: TESSEL

ii) T* semi-tessellating: SEMITESS, CONDB, CONDB-MAX-FIND, NOT-CONDB-LEVEL1, NOT-CONDB-LEVEL2

iii) T* non-tessellating: NONTESS, CONDA, CONDA-MAX-FIND, FILLOUT, FILLOUT-COMPLETE .

The second set consists of two functions which attempt to find a way in which a fragment of a sheet having a certain shape can be divided into pieces. RECTFILL handles rectangular fragments, whilst LSHAPE-FILL handles L-shaped fragments.



Relationships between reduction functions

Figure 6.3.1

The relationships between all of these functions are outlined diagrammatically in figure 6.3.1. An arrow pointing from one function name to another indicates that the first function can set up subgoals which the second function will be invoked to solve. In most cases the combination of the subgoal solution with information "memorized" by the function that set up the subgoal will

result in a solution of the subgoal this latter function was required to solve. In certain cases (those in which "(2)" follows the function name in the diagram), however, the solution of a subgoal must be combined not only with the memorized information, but also with the solution of a second subgoal, which will not be set up until a solution of the first subgoal is found. In these cases arrows pointing to functions that may be invoked to solve the first subgoal of a pair are labelled "1", and those pointing to functions invoked to solve the second are labelled "2".

Three lists, RECT-OCCURS, RECT-PENDING and RECT-SUCCESS are used to hold the information "learnt" by RECTFILL as to how rectangular fragments may be cut into pieces. The list LSHAPE-OCCURS holds similar information learnt by LSHAPE-FILL.

6.3.2 Search strategy

A search is being conducted in a problem reduction tree for a set of subgoal solutions which together define a usable cutting pattern. A step in this search may result in the solution of a subgoal. First, however, the case in which the previous step did not result in such a solution will be considered. In this case a search is made for the node which is active and has the lowest cost in its label. The function in this label is then executed with the values of its local variables set from information in the label. The function will execute until one of three things happens:

i) it finds a solution of the subgoal,

ii) it creates a new subgoal,

iii) it terminates having failed to do either of these things.

If its action is (i) or (ii) it may add to the costs of any of the present node, the parent of the present node, and the grand-parent of the present node. The size of the increment reflects the number of alternative reductions of the subgoal at the node at which it is made that it is thought should be attempted. The larger the increment the less frequently attempts will be made to reduce the subgoal. The cost of an ancestor is incremented when the fact that the reduction of the descendant has been made makes it less desirable to perform alternative reductions of the ancestor. The node associated with a new subgoal will have initial cost zero and will be marked active, unless information on one of the learning lists relating to it indicates that this should not be case. If the action of the function was (i) or (iii) the current node is marked inactive.

If, on the other hand, the previous step did result in the solution of a subgoal, the parent node of that subgoal is considered. The function in its label is reinstated with its local variables set from information in the subgoal solution. The function will either report the solution of the current subgoal, in which case the node is marked inactive, or create a new subgoal.

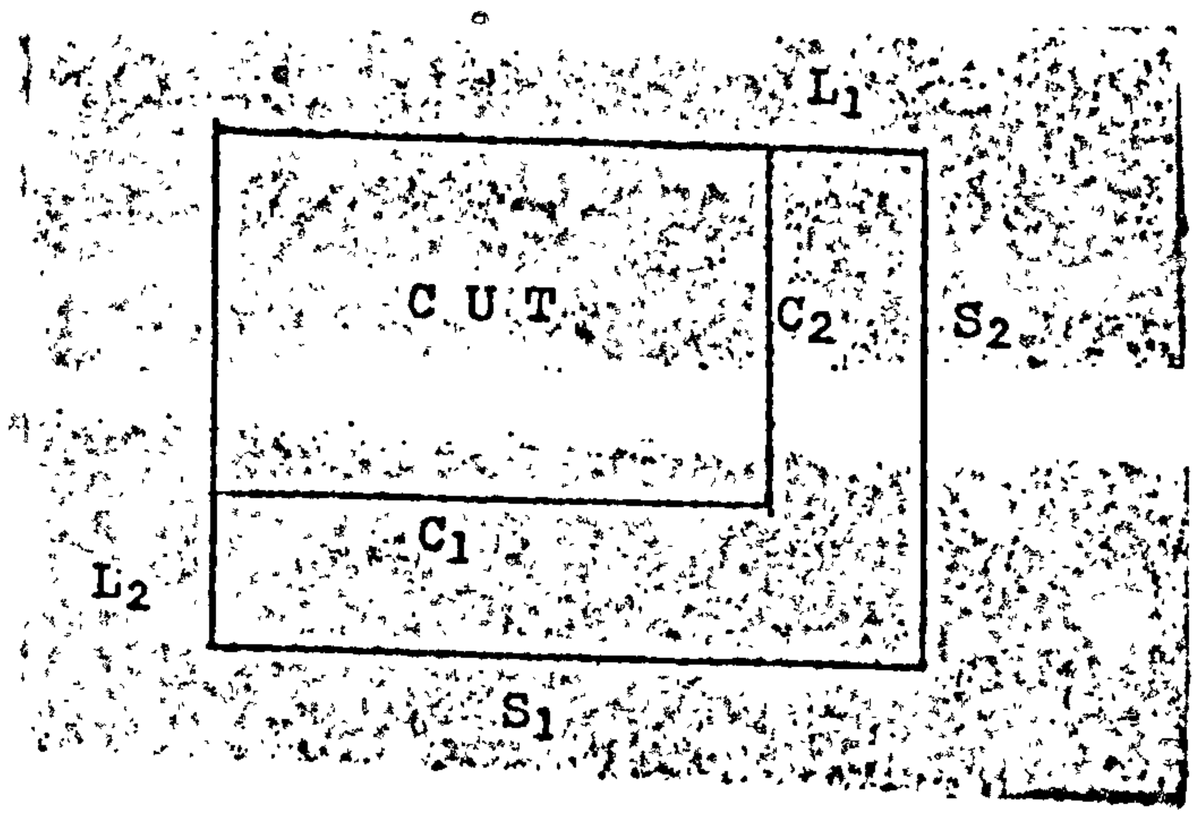
6.3.3 Functions associated with T*

For each of the three categories of T*, non-tessellating, semi-tessellating, and tessellating, there is a sequence of subgoals that can be set up, each

subgoal being the completion of a partial cutting pattern that includes pieces of type T*. Sub-sections 6.3.3.1 to 6.3.3.3 describe these sequences of subgoals and the operation of the functions that create them.

6.3.3.1 T* non-tessellating

Any sub-tessellation starting from the north-west corner of the sheet will result in a partial cutting pattern of the form shown in figure 6.3.2. All the subgoals are completions of such cutting patterns. Note that, depending on whether the sheet or T* are square, up to 4 pairs of values of L_1 and L_2 can result from sub-tessellations with m pieces in one direction and n pieces in the other.



Sheet with sub-tessellation cut

Figure 6.3.2

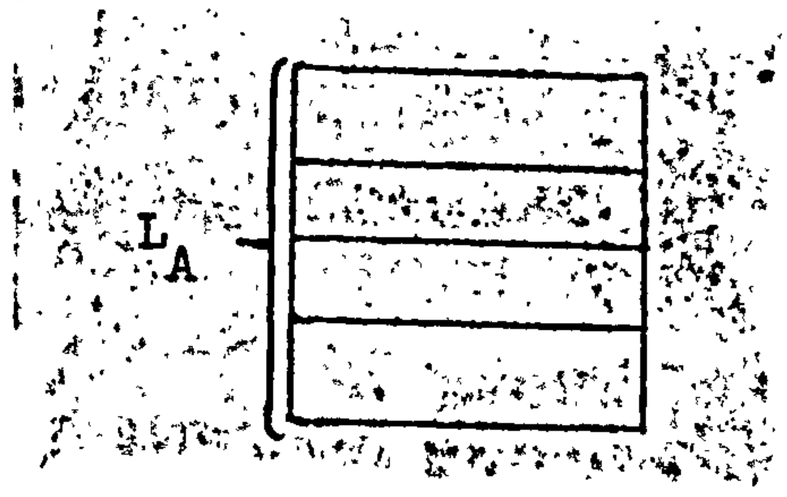
There may be a number of ways in which a value of L_1 or L_2 is divisible by the length of a side of a piece with urgency not less than 250. The function BEST-REMAINDER-FIND is used to evaluate the alternatives for values of L_1 and L_2 resulting from a given orientation of the pieces of type T* in the sub-tessellation. The evaluation for the L_1 case will be

described. C_2 is held constant at its largest possible value. The initial value of L_1 is the smallest possible consistent with the requirement that the sub-tessellation should not include more pieces of type T* than remain in the order list. The pieces that remain to be cut are considered in the sequence in which they occur in the order list. For each such piece both possible orientations are considered. A check is first made that there is not some piece of higher urgency for which the divisibility property has been found. Suppose there is not, that side length P_1 of the piece divides L_1, and that the other side of the piece has length P_2. If P_2 divides C_2 or S_2 and the number of pieces of this type remaining in the order list is sufficient to form a sub-tessellation with dimensions L_1 and C_2 or S_2 (as appropriate) then BEST-REMAINDER-FIND exits with a value of 1000 + urgency of this piece. Otherwise, if the number of pieces of this type remaining is at least L_1/P_1 the details of the piece and orientation are noted. If these are the details finally returned by BEST-REMAINDER-FIND the value will be the urgency of this piece. When all pieces have been considered for a particular value of L_1, the next largest value of L_1 is considered. The evaluation for L_2 is similar, for a description simply interchange the 1's and 2's in the subscripts in the above.

If one of the evaluations of BEST-REMAINDER-FIND identifies such a divisibility condition then condition A is said to hold. If both evaluations identify such divisibility conditions, condition A refers to the one

with the higher value. The aim of establishing condition A is to devise partial cutting patterns which, when completed, would produce cutting patterns that, when used, would simultaneously reduce the number of pieces uncut for two pieces, whilst making the completion of such partial cutting patterns a relatively easy problem.

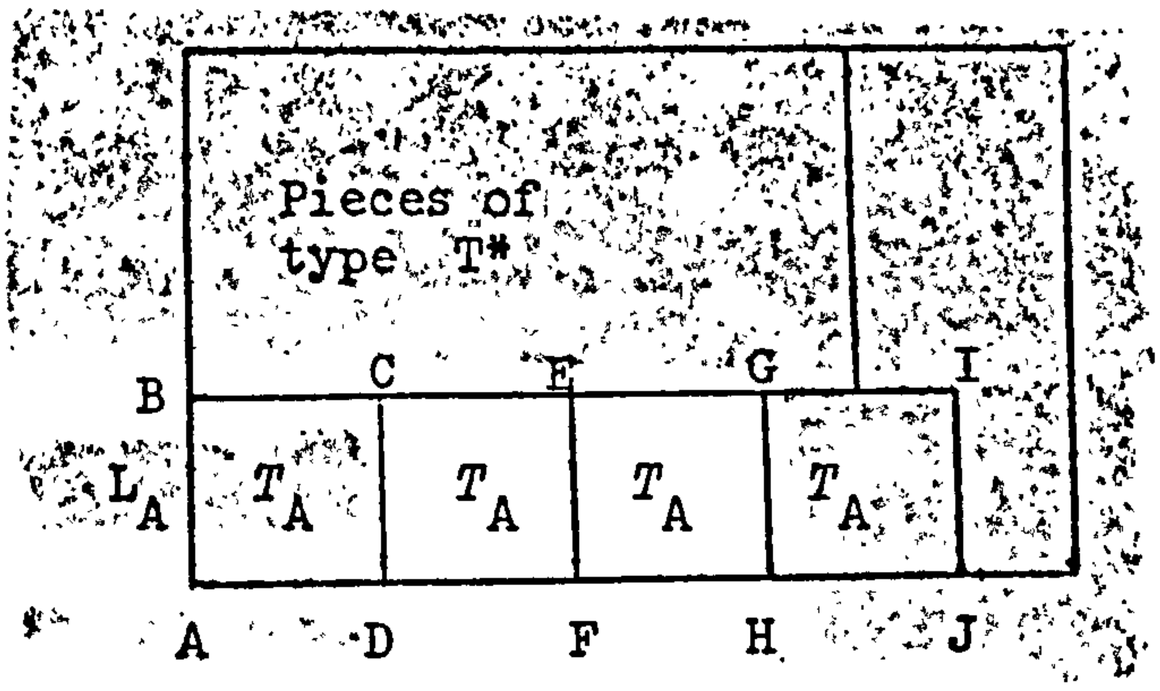
Let L_A denote the value of L_1 or L_2 specified by condition A. Let T_A denote a sub-tessellation consisting of a set of the specified pieces with the specified orientation juxtaposed so that the sides whose lengths divide L_A together form a side of the sub-tessellation of length L_A and the other side length of the sub-tessellation is equal to the other side length of the piece. An example of such a sub-tessellation is shown in figure 6.3.3.



T_A sub-tessellation

Figure 6.3.3

CONDA-MAX-FIND is invoked if it has been established that condition A holds. It first determines the largest sub-tessellation using sub-tessellations of type T_A with their sides of length L_A juxtaposed that can be formed in the remainder of the sheet starting at the edge of length L_A and with a sub-tessellation side of length L_A along this edge. A possible form of this sub-tessellation is shown in figure 6.3.4.



State generated by CONDA-MAX-FIND

Figure 6.3.4

The first subgoal set up by CONDA-MAX-FIND is the completion of this cutting pattern. Subsequent subgoals are to complete a cutting pattern of this form in which the number of T_A sub-tessellations in this sub-tessellation is reduced by 1 each time until it reaches 1. In terms of figure 6.3.4, in the creation of subsequent subgoals the sub-tessellation is successively ABGH, ABEF, and ABCD. Each time it sets up a subgoal CONDA-MAX-FIND adds 1 to the cost of the current node, 1 to the cost of the parent node, and 1 to the cost of the grandparent node.

CONDA is given an orientation of T* and determines whether condition A holds for this orientation. If it does, CONDA sets up a subgoal to be solved by CONDA-MAX-FIND and adds 1 to the cost of the current node.

The first goal set up by NONTESS is for CONDA to establish whether condition A holds with the breadth of T* parallel to the breadth of the sheet. If neither the sheet nor T* are square the next subgoal set up is for

CONDA to establish whether condition A holds for the other orientation of T*.

NONTESS next determines the upper bounds for M and N, the number of pieces in each row and column of a sub-tessellation using pieces of type T*. For descending values of M+N, for descending values of M within M+N, and for both orientations of T* within N when neither the sheet nor T* are square, subgoals are set up, at successive entries to NONTESS, to complete a partial cutting pattern that starts with an M by N sub-tessellation using pieces of type T*, where MxN is not greater than the number of pieces of type T* still in the order list.

The last subgoal set up by NONTESS has FILLOUT as the function in the label of the node. The strategy controlled by FILLOUT is as follows:

i) Consider a sub-tessellation using all the remaining pieces of type T*, if such a sub-tessellation exists.

ii) Consider a type occurring later in the order list than T* for which pieces remain to be cut.

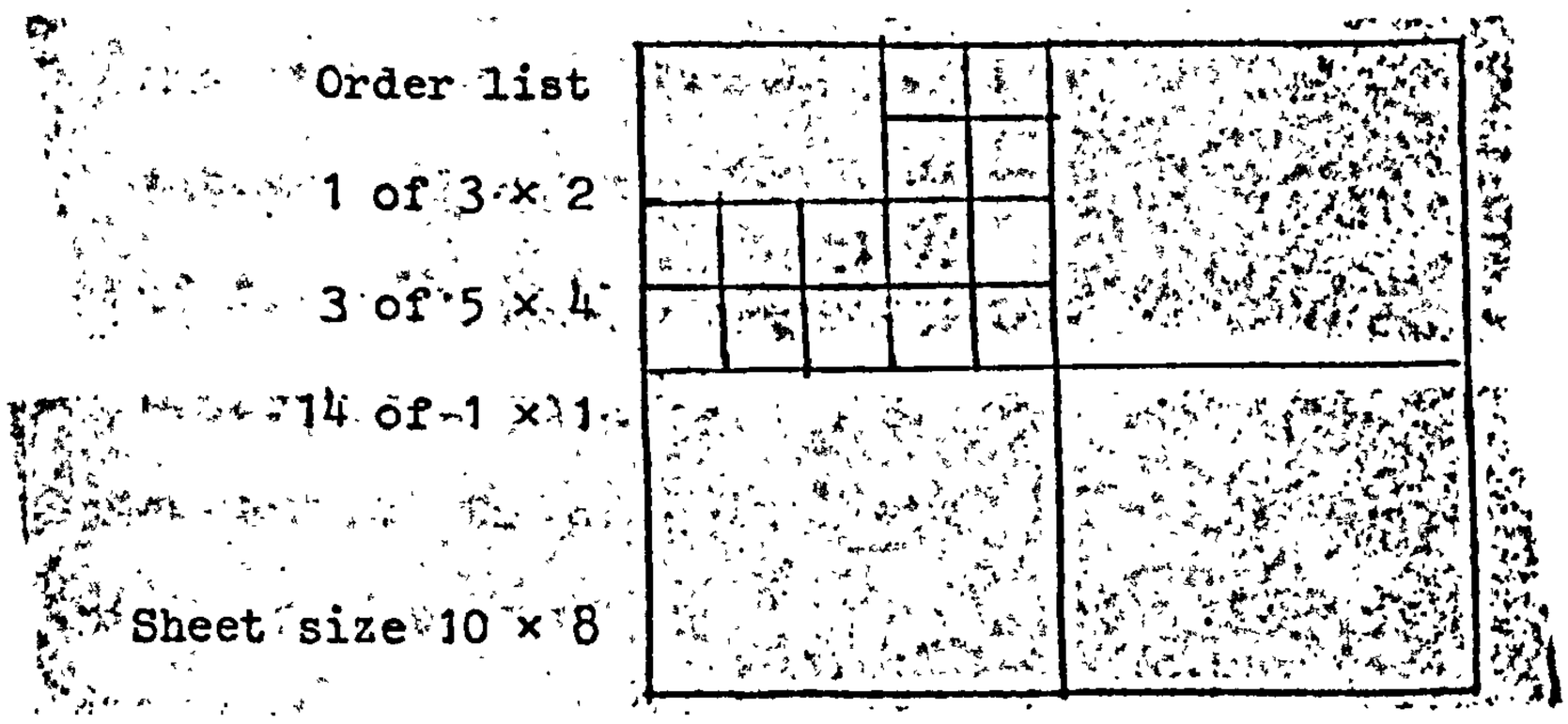
iii) Determine the minimal sub-tessellation with pieces of this type such that the lengths of both sides of the sub-tessellation are not less than those of the corresponding sides of the sub-tessellation in (i).

iv) If the sub-tessellation in (iii) would leave an L-shaped fragment uncut, determine a partial cutting pattern that includes the sub-tessellation in (i) and defines the cutting of a rectangle whose sides are equal to the sides of the sub-tessellation in

(iii).

v) Determine a way in which the remaining L-shaped fragment may be cut.

Note that in (i) it is required that all remaining pieces of type T* be used and that in (iv) it is required that the remaining fragment be L-shaped. This strategy is specifically designed to "embed" the remaining pieces of type T* in a sub-tessellation with another type to produce a solution that would not otherwise be found. Figure 6.3.5 shows how this strategy might generate a solution.



Strategy of FILLOUT

Figure 6.3.5

If the piece into a sub-tessellation with which the remaining pieces of type T* are to be embedded is not square there are two distinct subgoals for creating a partial cutting pattern of the type specified in (iv) above. These will be created on successive entries to FILLOUT-COMPLETE that are not consequential upon the solution of a subgoal having been found. If the piece is square only one subgoal is set up. If FILLOUT-COMPLETE is entered following the solution of such a subgoal it sets up the subgoal of completing the resultant partial

cutting pattern.

Successive entries to FILLOUT set up subgoals to be solved by FILLOUT-COMPLETE in the fiollowing order:

i) Consider in turn each type following T* on the order list for which pieces remain to be cut.

ii) For each type in (i) consider firstly those sub-tessellations in which the breadth of the pieces of type T* is parallel to the breadth of the sheet and the length of the sub-tessellation is successively 1,2,.... pieces, where such sub-tessellations exist. Secondly, if neither T* nor the sheet are square, consider in turn the sub-tessellations in which the length of the pieces of type T* is parallel to the breadth of the sheet.

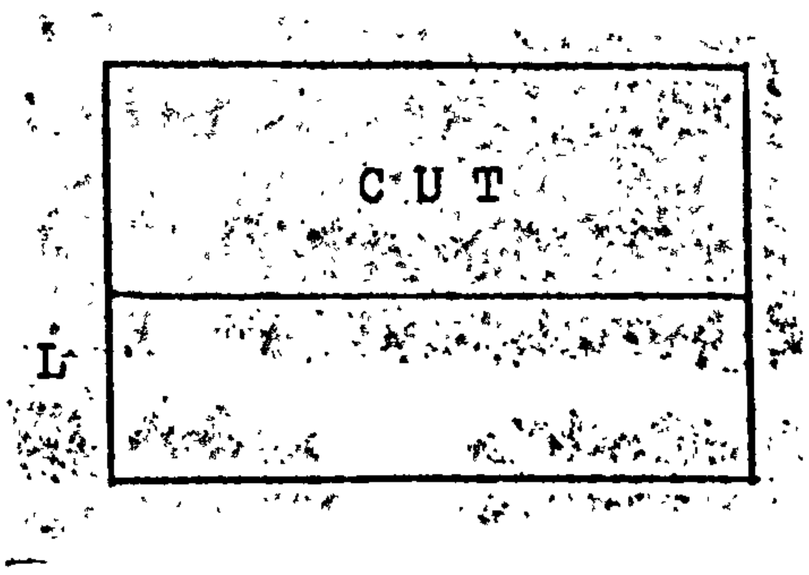
Each time FILLOUT-COMPLETE sets up a subgoal not consequential upon the solving of a previous subgoal it adds 1 to the value of the current node. Each time FILLOUT sets up a subgoal it adds 1 to the value of the current node. Each time NONTESS sets up a subgoal it adds 6 to the cost of the current node.

6.3.3.2 T* semi-tessellating

Suppose that sheet dimension S_1 is divisible by dimension P_1 of type T*, so that $S_1/P_1 = R_1$. Then a sub-tessellation of R pieces of type T* with their sides of length P_1 along the sheet side of length S_1 can be formed. Call such a sub-tessellation a row. Note that either of the piece dimensions might divide either of the sheet dimensions, so it may be possible to form more than one type of row.

When a sub-tessellation is formed by juxtaposing

several rows, the uncut fragment of the sheet will be rectangular, as in figure 6.3.6. There may be a number of ways in which L is divisible by the length of a side of a piece with urgency not less than 250. The function BEST-REMAINDER-FIND is used to evaluate these alternatives; in the account in section 6.3.3.1, C_2 is equal to S_2 and L_1 denotes L. If BEST-REMAINDER-FIND identifies a divisibility condition, condition B is said to hold.

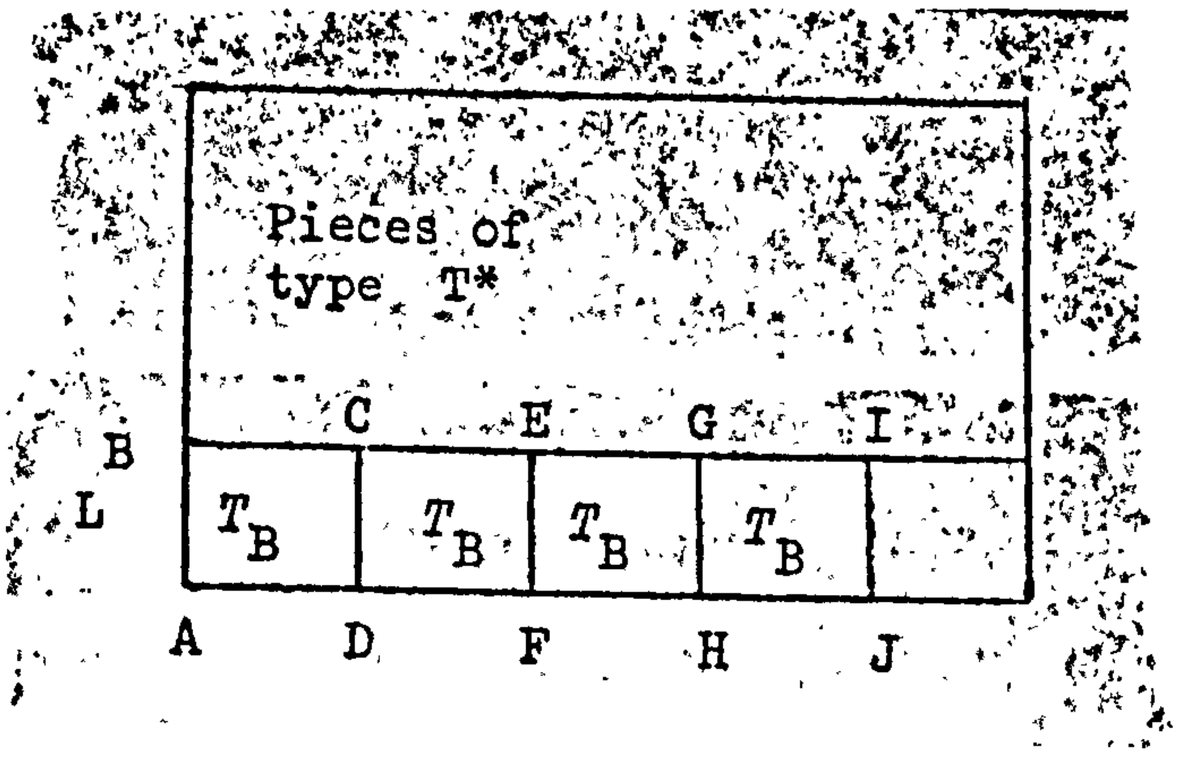


Sub-tessellation with semi-tessellating pieces

Figure 6.3.6

Let L_B denote the value of L specified by condition B and let T_B denote a sub-tessellation analogous to T_A in section 6.3.3.1.

CONDB-MAX-FIND is invoked if it has been established that condition B holds. It first determines the largest sub-tessellation using sub-tessellations of type T_B with their sides of length L_B juxtaposed that can be formed in the remainder of the sheet starting at a side of length L_B and with a sub-tessellation of length L_B along this side. A possible form of this sub-tessellation is shown in figure 6.3.7.



State generated by CONDB-MAX-FIND

Figure 6.3.7

The first subgoal set up by CONDB-MAX-FIND is the completion of this cutting pattern. Subsequent subgoals are to complete a cutting pattern of this form in which the number of T_B's in this sub-tessellation is reduced by 1 each time until it reaches 1. In terms of figure 6.3.7, in the creation of subsequent subgoals the sub-tessellation is successively ABGH, ABEF, and ABCD. Each time it sets up a subgoal CONDB-MAX-FIND adds 1 to the cost of the current node, 1 to the cost of the parent node, and 1 to the cost of the grandparent node.

CONDB is given an orientation of T* and a direction in which rows can be formed given this orientation of T*, and determines whether condition B holds in this case. If it does, CONDB sets up a subgoal to be solved by CONDB-MAX-FIND and adds 1 to the cost of the current node.

NOT-CONDB-LEVEL2 is given an orientation of T*, a direction in which rows can be formed given this orientation of T*, and a number of rows. It checks that the number of pieces of type T* still in the order list

is not less than that required to form the requested number of rows. If this is case, it sets up the subgoal of completing the partial cutting pattern that starts with the given sub-tessellation with pieces of type T* and adds 6 to the cost of the current node.

NOT-CONDB-LEVEL1 determines the upper bound on MR, the number of rows that can be formed given either orientation of T* and either direction of the sheet. For descending values of MR, and for both orientations and both directions within MR, where these would result in distinct subgoals, subgoals are set up, at successive entries to NOT-CONDB-LEVEL1, for a partial cutting pattern that starts with MR rows of T* with the given orientation and direction to be considered by NOT-CONDB-LEVEL2. Each time a subgoal is set up, 6 is added to the cost of the current node and 6 to the cost of the parent node.

The first sequence of subgoals created by SEMITESS is for CONDB to establish whether condition B holds for each of the possible rows. Rows whose shapes are the same as those already considered, because T* or the sheet are square, are ignored. Each time such a subgoal is created, 1 is added to the cost of the present node. The next subgoal set up is for NOT-CONDB-LEVEL1 to create a cutting pattern The last subgoal that can be created is for NONTESS to create a cutting pattern. The sub-tessellations with pieces of type T* that are examined by NONTESS are distinct from those that have previously been considered; also, in this case, NONTESS will not investigate the possibility of condition A

holding.

6.3.3.3 T* tessellating

If the total area of pieces of type T* still in the order list exceeds the area of a sheet, TESSEL can immediately create a cutting pattern. Otherwise a subgoal for SEMITESS to create a cutting pattern is set up.

6.3.4 The learning lists

The assumptions on which the maintenance of the list LSHAPE-OCCURS is based are:

i) if the subgoal of dividing an L-shaped fragment of certain dimensions into pieces has been set up a number of times but never solved, then the greater the number of times it has been set up the less the likelihood that a solution will ever be found,

ii) if in a global step a number of subgoals of dividing an L-shaped fragment of certain dimensions have been set up and no solution has yet been found to any such subgoal, only one such subgoal should be active at one time, as this will avoid the parallel pursuit of solutions to a possibly insoluble problem,

iii) if in a global step a number of subgoals of dividing an L-shaped fragment of certain dimensions have been set up and a solution has been found to one such subgoal, then all such subgoals should be active, since it is likely that all the subgoals have solutions.

Each element of LSHAPE-OCCURS has three parts, the dimensions of an L-shaped fragment, a count, and a flag. Whenever a subgoal to be solved by LSHAPE-FILL is set up, this list is inspected. If there is not an element

corresponding to the L-shape to be handled in this subgoal, an element consisting of the dimensions of this L-shape, a count of 1, and a set flag is added to LSHAPE-OCCURS. If there is an element corresponding to the L-shape then the cost of the node is made equal to twice the value of the count and 1 is added to the count. If the flag was set the node is marked as suspended, otherwise the flag is set. When LSHAPE-FILL solves a subgoal the corresponding element is removed from LSHAPE-OCCURS, and all subgoals for the handling of L-shapes of those dimensions currently marked suspended are re-marked active. If LSHAPE-FILL is executed and terminates without solving the current subgoal or setting up a new subgoal, the current node therefore being marked inactive, one subgoal for the handling of an L-shape of those dimensions currently marked suspended is re-marked active. At the end of each global step all flags are cleared. If the amount of scrap allowable is increased the existing elements of LSHAPE-OCCURS are discarded, since it may now be possible to generate instructions to divide a fragment that could not be divided into pieces when less scrap was allowable.

The elements of the list RECT-SUCCESS are sets of instructions for dividing rectangular fragments into pieces. When such a set is found it is added to RECT-SUCCESS. When instructions for dividing a rectangular fragment are sought, RECT-SUCCESS is inspected to determine whether it includes instructions for dividing a fragment of the dimensions of the one being considered. If it does, and if the use of these

instructions would not result in the cutting of pieces in excess of the order quantities, the instructions provide a solution of the subgoal. Retrieving of subgoal solutions in this way not only reduces search time, but also makes it possible for solutions found when one set of urgency values was in use to be used when the other set of urgency values is in use. In this case the search might not otherwise reach a solution before the search time expired. At each global step all sets of instructions that would necessarily result in the cutting of pieces in excess of the order quantities are removed from RECT-SUCCESS.

When the subgoal under consideration is the cutting of a rectangular fragment into pieces, there will be a sequence of alternative actions, both the setting up of subgoals and the generation of cutting instructions, that can be taken. This sequence may be of considerable length. If each of a number of subgoals, all for the cutting of rectangular fragments of a certain size, are treated independently, it may be expected that the same sequence of events will occur for each of them, resulting in the same (possibly abortive) search pattern. The list RECT-PENDING is used for avoidance of this situation.

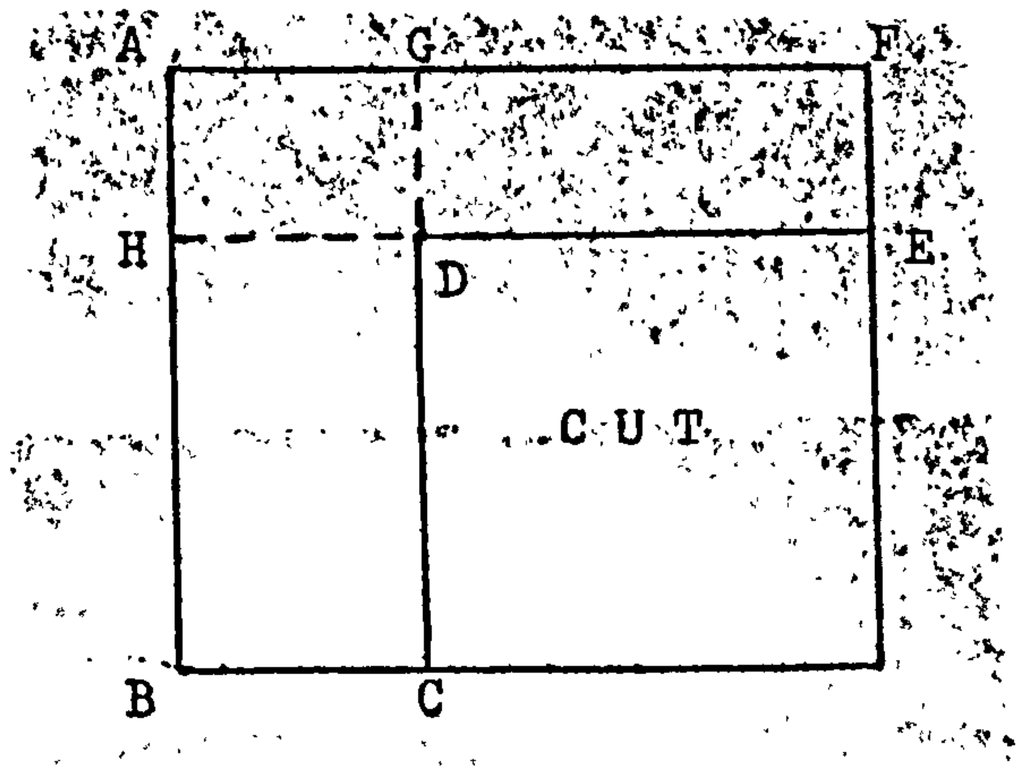
If some, but not all, of the possible actions for a fragment of a particular size have been considered, details of the last action taken, which will have resulted in the setting up of a subgoal, are stored in RECT-PENDING. On the next occasion that the subgoal of dividing a fragment of that size is encountered, these details will be used to determine the action to be taken,

this being the next after the one whose details were found in RECT-PENDING, if any such action exists. RECT-PENDING is retained from one invocation of REGULAR to the next, unless the values of the urgencies are changed, in which case the order in which the alternatives are to be considered will be changed. This means that actions for a subgoal of dividing a rectangular fragment of a given size that have not been considered in a previous global step can be preferred to those that have.

Where all possible actions for the subgoal of dividing a rectangular fragment of a given size have been considered, an entry is made in the list RECT-OCCURS. RECT-OCCURS is used similarly to LSHAPE-OCCURS, to allow only one node relating to a fragment of a given shape to be active at any one time and to weight such nodes according to the frequency with which they occur. In the case of rectangular fragments, the cost of a new node relating to a rectangle is made equal to the count for that rectangle.

6.3.5 Rectangular and L-shaped fragments

A check is made at the time that a subgoal is created that the smallest dimension of a rectangular or L-shaped fragment that is to be divided into pieces is not less than the smallest dimension of any order piece. Subgoals in which this condition would be violated are not set up.



Alternatives for LSHAPE-FILL

Figure 6.3.8

The division of L-shaped fragments into pieces is handled by the function LSHAPE-FILL. In figure 6.3.8, EF is less than BC. The first entry to LSHAPE-FILL for this L-shape would set up the subgoal of dividing the rectangle DEFG into pieces. An entry to LSHAPE-FILL consequent upon the solution of this subgoal will set up the subgoal of dividing the rectangle ABCG. When this is solved the entire subgoal for this L-shape is soved. The second entry to LSHAPE-FILL not consequent upon the solution of a subgoal sets up the subgoal of dividing the rectangle AFEH. An entry to LSHAPE-FILL consequent upon the solution of this subgoal sets up the subgoal of dividing the rectangle BCDH. For L-shapes in which AF is equal to AB and BC is equal to EF, this second case does not occur. When a subgoal is set up as the result of an entry not consequent upon the solution of a subgoal, 1 is added to the cost of the current node.

The division of rectangular fragments into pieces is handled by the function RECTFILL. On the first entry to RECTFILL a check is made that the tota area of pieces

with breadth not greater than the breadth of the fragment is at least equa to the area of the fragment. If this is not the case, the insolubility of the subgoal is reported.

Each entry to RECTFILL causes the specification of a sub-tessellation to be cut from the rectangular fragment. If the dimensions of the sub-tessellation are equal to those of the fragment then a solution to the subgoal has been found. Otherwise a subgoal is set up for the division of the remaining fragment, which may be either rectangular or L-shaped, into pieces.

For each piece with which a sub-tessellation that could be cut from the fragment can be formed, there is a preferred orientation. If the maximal sub-tessellation formed with the pieces having one orientation contains more than that with the pieces having the other, the orientaion which allows the larger number of pieces in the maximal sub-tessellation is preferred. Otherwise, if the orientation with the breadth of the piece parallel to the breadth of the fragment results in one of the fragment dimensions being divisible by the corresponding piece dimension then this orientation is preferred, as a subgoal can be created for the cutting of a rectangular fragment. Otherwise the other orientation is preferred.

Let M and N be the number of pieces in each row and column of a maximal sub-tessellation for a given piece and orientation. Then a sequence of sub-tessellations with that piece and orientation can be defined by taking descending values of M+N and descending values of M within M+N. Only those sub-tessellations in which M*N is

not greater than the total number of pieces remaining to be cut are "usable".

The sequence of sub-tessellations that can be generated by RECTFILL is:

i) in order of urgency of types for which pieces remain to be cut and sub-tessellations can be formed that can be cut from the fragment, the first usable sub-tessellations with the preferred orientation,

ii) in the same order, the first usable sub-tessellations, if any, with the other orientation,

iii) in the same order, the next usable sub-tessellations, if any, with the preferred orientation,

iv) and so on, until there are no usable sub-tessellations with either orientation.

Whenever RECTFILL creates a new subgoal, 1 is added to the cost of the current node.

6.4 The routine IRREGULAR

6.4.1 Structure of the routine

The routine is based on the program for the solution of 1½-dimensional trim-loss problems described in chapter 3. In this routine, however, the successors of the root node of the search tree are specified explicitly. They are:

i) a partial cutting pattern consisting of a piece of type T* with its north-west corner coincident with the north-west corner of the sheet and its breadth parallel to the breadth of the sheet,

ii) if neither T* nor the sheet is square, a partial

cutting pattern consisting of a piece of type T* with its north-west corner coincident with the north-west corner of the sheet and its length parallel to the breadth of the sheet,

iii) partial cutting patterns found by the routines described in section 6.4.2.

This explicit specification ensures that any cutting pattern developed will include at least one piece of type T*.

6.4.2 Spiral configurations

A category of non-guillotine instructions for the cutting of a number of non-square pieces from a sheet can be identified in which the pieces are arranged in a spiral manner. A simple example is shown in figure 6.4.1. Such sets of instructions leave a fragment of the sheet to be divided into other pieces.

Simple spiral arrangement

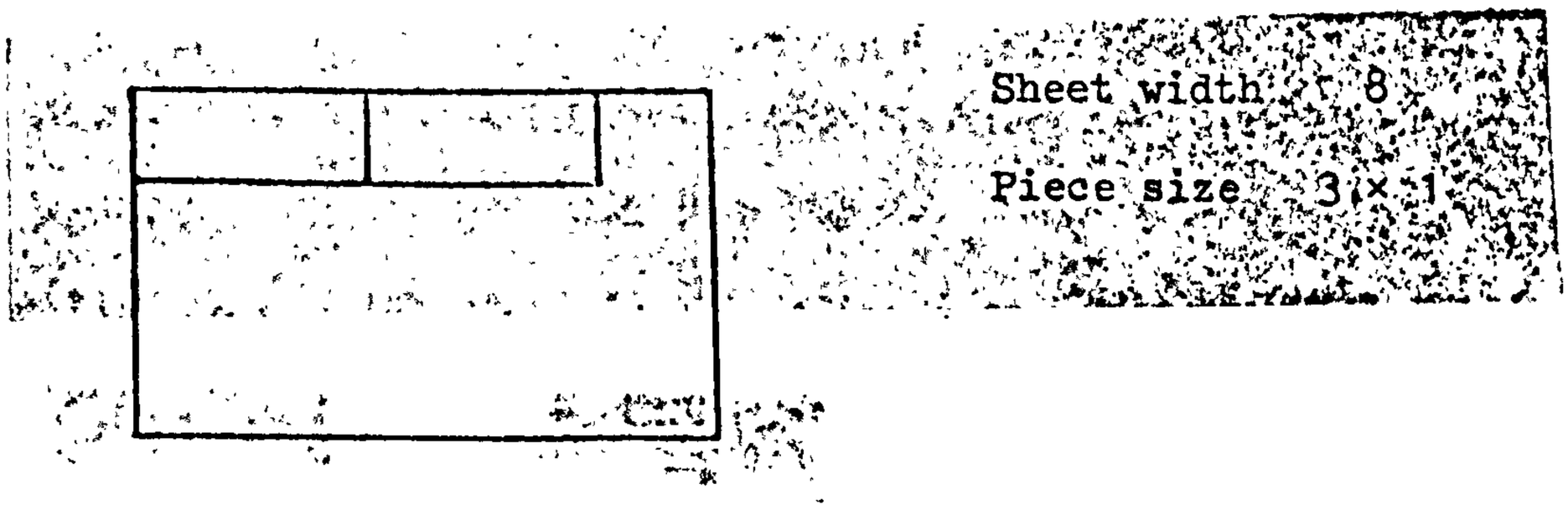
Figure 6.4.1

Several different ways of constructing spiral configurations are possible. The piece in the north-west corner of the sheet may have its length or its breadth parallel to the breadth of the sheet and the spiral may be constructed in a clockwise or anti-clockwise

direction. The algorithm is described in terms of the case where the breadth of the first piece positioned is parallel to the breadth of the sheet and the spiral is constructed in a clockwise direction.

i) Check that the length of the piece does not divide the length of the sheet.

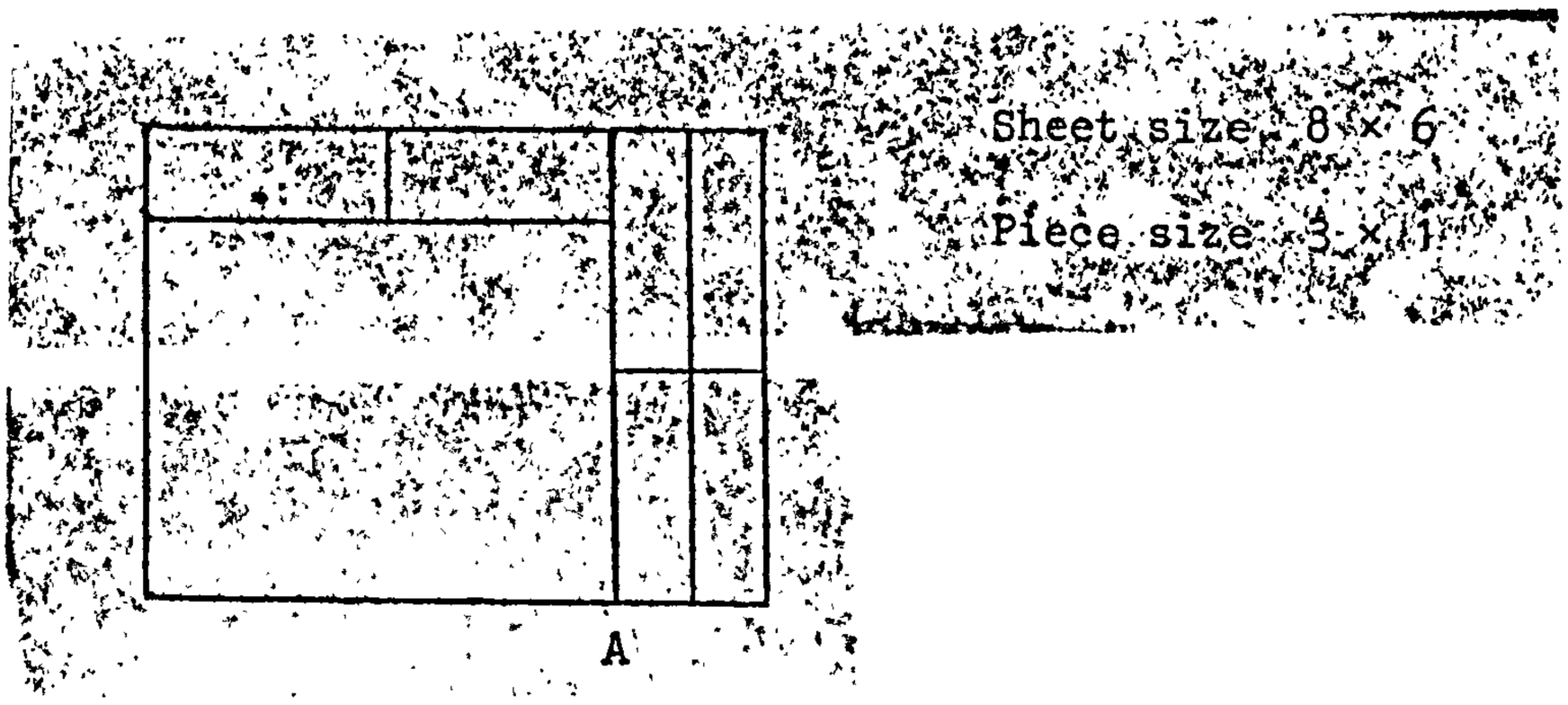
ii) Let PB denote the breadth of a piece, PL its length, SB the breadth of a sheet, and SL its length. Find the largest value of N_1 such that $SL=N_1*PL+M_1*PB$ if any such positive values of M_1 and N_1 exist. Lay out N_1 pieces with their breadth sides adjacent, starting in the north-west corner of the sheet, as in figure 6.4.2.



First stage of spiral

Figure 6.4.2

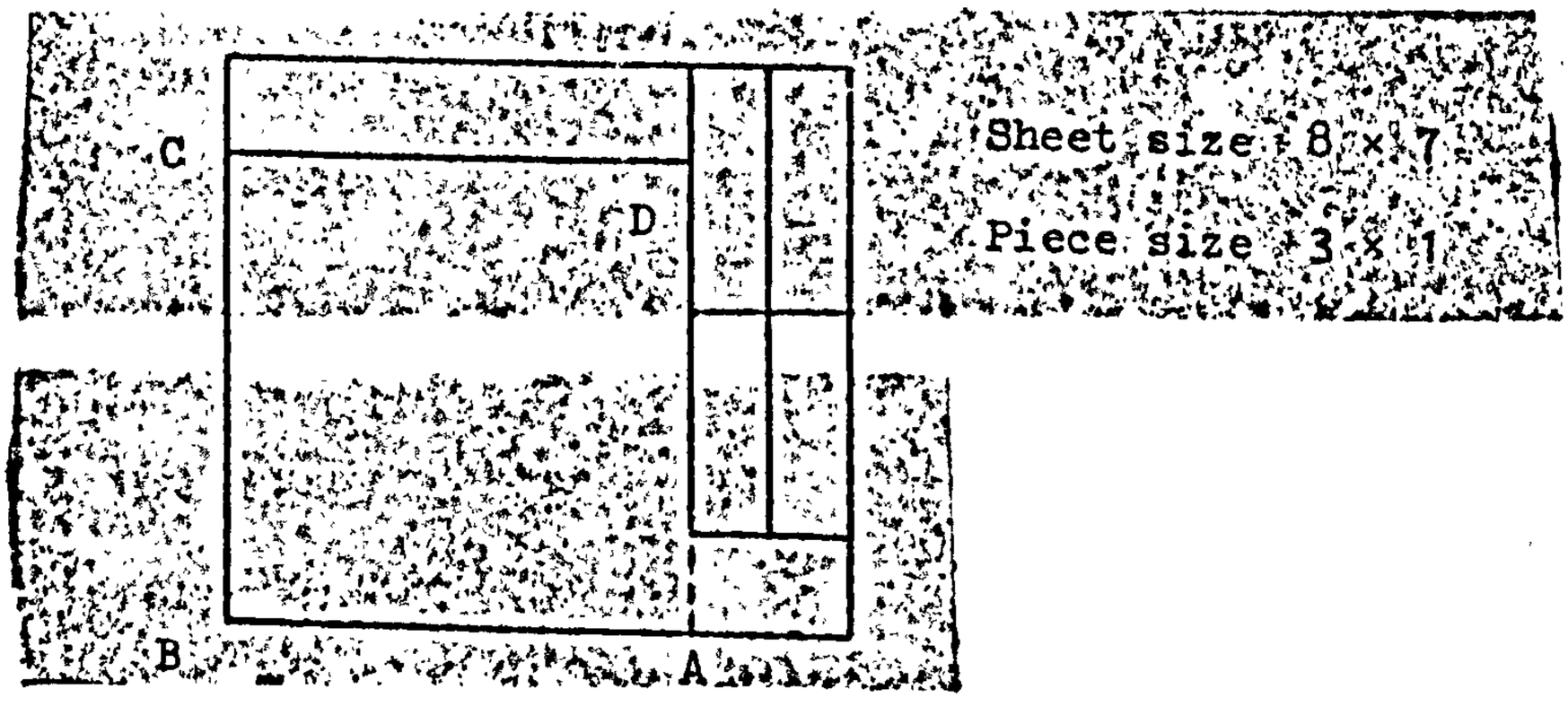
iii) Find the largest value of N_2 such that $SB=N_2*PL+M_2*PB$ if any such positive values of M_2 and N_2 exist. Lay out N_2 sub-tessellations, each consisting of M_2 pieces with their length edges adjacent, starting in the north-east corner of the sheet.



Possible second stage of spiral

Figure 6.4.3

iv) This may leave a rectangular fragment uncut, as in figure 6.4.3. If so, treat the remaining fragment as the original sheet, starting in the corner labelled A. Note that for this fragment it may be the case that SL=N*PL for some N. Laying out pieces in the manner of (ii) will now reduce the fragment to a smaller rectangle and subsequent pieces should be laid out parallel to the previous set.



Alternative second stage of spiral

Figure 6.4.4

v) Alternatively the fragment left uncut may be L-shaped, as in figure 6.4.4. In this case the reduction required is that described in (iii) with the changes necessitated by a rotation through 90°.

Reduction of L-shapes can continue until a rectangular fragment remains unless the rectangular part of the current L-shape (ABCD in figure 6.4.4) could have pieces laid out in it, whilst the rectangular part of the reduced L-shape could not. If this is so, lay out pieces in the rectangular part and exit.

vi) If at any point no appropriate values of N and M in (ii) and (iii) exist, exit.

This technique can be extended to deal with the case in which a number of different types of piece have one dimension in common and the other multiples of one another, e.g. 7x4, 7x2, 7x1. The spiral configuration is generated using the smallest piece, the partial cutting pattern then being transformed by replacing sub-tessellations with the smallest piece with individual larger pieces. A check is necessary here to ascertain that some pieces of type T* are included in the resultant partial cutting pattern.

6.5 Results

A program using the techniques described above has been written in POP-2 (Burstall, Collins and Popplestone, 1971) and tested using the 16 sets of data described in table 6.5.1, on an ICL 4130 using the Multi-Pop system (Dunn, 1972). The results of these tests are shown in table 6.5.2. On 14 sets of data the results clearly cannot be improved upon. In the remaining two cases there is no evidence that better solutions are possible.

In all cases except 5, 7 and C the computation time is less than 15 minutes. In these three cases, however,

the computation time is very large (35 mins, 78 mins and 29 mins), particularly considering that it does not include garbage collection time, which is a system cost under Multi-Pop. Considerable improvement could be expected if the program were rewritten in a compiled, rather than an interpreted, language, and the ICL 4130 is not a fast machine by present-day standards. Nonetheless, these times do indicate that considerable saving could result from the development of heuristics to avoid abortive searches.

Table 6.5.1

Listing of test data

Set 1	Set 2	Set 3
736 of 9 × 3	752 of 9 × 8	945 of 9 × 7
529 of 6 × 3	354 of 8 × 7	119 of 8 × 7
192 of 10 × 8	391 of 10 × 9	574 of 9 × 6
903 of 9 × 5	1353 of 10 × 6	715 of 7 × 7
548 of 8 × 5	447 of 9 × 5	595 of 8 × 6
1242 of 10 × 3	296 of 6 × 4	803 of 7 × 3
711 of 8 × 2	327 of 9 × 2	801 of 6 × 3
248 of 4 × 3	645 of 5 × 3	159 of 10 × 6
214 of 6 × 1	992 of 9 × 1	10 of 9 × 5
304 of 3 × 2	853 of 8 × 1	981 of 8 × 5
235 of 10 × 2	447 of 3 × 2	630 of 9 × 4
607 of 5 × 4	883 of 3 × 1	139 of 7 × 5
55 of 10 × 1	953 of 10 × 5	597 of 8 × 4
1173 of 4 × 2	171 of 10 × 4	514 of 7 × 2
7 of 2 × 1	559 of 5 × 4	1326 of 6 × 1
424 of 1 × 1	721 of 10 × 1	660 of 5 × 4
	699 of 5 × 2	352 of 5 × 1
		592 of 4 × 1

Set 4	Set 5	Set 6
836 of 9 × 8	899 of 10 × 10	278 of 8 × 8
587 of 7 × 6	430 of 9 × 9	306 of 8 × 6
51 of 6 × 3	665 of 9 × 8	657 of 7 × 3
1871 of 10 × 8	94 of 9 × 6	423 of 10 × 8
445 of 8 × 5	734 of 7 × 7	996 of 10 × 6
897 of 8 × 2	963 of 9 × 3	596 of 9 × 5
801 of 9 × 1	482 of 10 × 6	222 of 9 × 4
653 of 8 × 1	526 of 9 × 5	850 of 8 × 4
744 of 3 × 2	807 of 9 × 4	503 of 7 × 4
884 of 6 × 1	1289 of 7 × 5	956 of 8 × 2
466 of 5 × 4	413 of 10 × 3	92 of 7 × 2
642 of 10 × 1	138 of 8 × 2	279 of 3 × 2
820 of 5 × 2	873 of 4 × 3	830 of 10 × 5
333 of 4 × 2	384 of 9 × 1	1174 of 10 × 4
630 of 4 × 1	588 of 10 × 4	996 of 5 × 2
646 of 2 × 1	543 of 10 × 2	439 of 4 × 2
289 of 1 × 1	99 of 4 × 2	440 of 5 × 1

Set 7

230 of 10 × 10
875 of 9 × 9
405 of 8 × 8
200 of 8 × 6
753 of 6 × 6
437 of 10 × 7
591 of 9 × 5
765 of 8 × 5
592 of 9 × 4
940 of 10 × 3
991 of 5 × 3
212 of 8 × 1
757 of 7 × 1
128 of 3 × 2
233 of 5 × 5
546 of 5 × 2
539 of 4 × 2
322 of 5 × 1
498 of 2 × 1

Set 8

246 of 9 × 7
885 of 7 × 7
479 of 6 × 6
180 of 9 × 3
959 of 8 × 3
471 of 7 × 3
218 of 10 × 8
49 of 10 × 7
596 of 8 × 5
967 of 6 × 5
1588 of 5 × 3
155 of 6 × 2
473 of 7 × 1
632 of 6 × 1
245 of 10 × 2
260 of 4 × 1

Set 9

597 of 8 × 8
452 of 8 × 7
136 of 9 × 3
978 of 3 × 3
265 of 10 × 6
990 of 9 × 5
941 of 7 × 4
499 of 5 × 3
395 of 4 × 3
233 of 6 × 2
873 of 9 × 1
604 of 3 × 2
375 of 3 × 1
987 of 10 × 5
42 of 5 × 4
1080 of 4 × 1
265 of 2 × 1

Set A

275 of 9 × 9
178 of 9 × 6
769 of 7 × 3
506 of 10 × 3
274 of 7 × 4
713 of 9 × 2
928 of 6 × 2
948 of 4 × 3
1025 of 9 × 1
504 of 7 × 1
775 of 5 × 5
400 of 5 × 4
887 of 10 × 2
1522 of 4 × 2
1505 of 5 × 1

Set B

81 of 9 × 8
710 of 8 × 7
278 of 7 × 7
1034 of 9 × 3
98 of 8 × 3
93 of 3 × 3
416 of 10 × 9
796 of 10 × 7
863 of 6 × 5
544 of 5 × 3
689 of 4 × 3
414 of 9 × 1
944 of 7 × 1
397 of 6 × 1
463 of 3 × 2
710 of 5 × 4
849 of 5 × 2

Set C

841 of 9 × 6
547 of 7 × 6
1540 of 6 × 3
281 of 10 × 8
424 of 10 × 7
1036 of 7 × 4
278 of 9 × 2
419 of 8 × 2
968 of 7 × 2
61 of 6 × 2
435 of 4 × 3
669 of 9 × 1
1674 of 7 × 1
1303 of 10 × 4
355 of 5 × 5
374 of 10 × 2

Set D

822 of 9 × 6
940 of 8 × 6
1085 of 10 × 9
232 of 10 × 7
67 of 8 × 5
527 of 10 × 3
938 of 9 × 2
173 of 8 × 2
804 of 4 × 3
721 of 8 × 1
836 of 10 × 5
1535 of 10 × 4
366 of 4 × 4
305 of 2 × 2
924 of 2 × 1

Set E

255 of 9 × 8
11 of 8 × 3
661 of 3 × 3
12 of 10 × 9
872 of 8 × 5
620 of 9 × 4
813 of 8 × 4
734 of 6 × 5
804 of 7 × 2
250 of 6 × 2
379 of 4 × 3
727 of 3 × 2
806 of 10 × 2
519 of 10 × 1
509 of 5 × 2
906 of 4 × 2
177 of 4 × 1

Set F

480 of 8 × 8
626 of 9 × 7
687 of 8 × 3
610 of 8 × 5
1018 of 6 × 5
488 of 6 × 4
533 of 9 × 2
13 of 4 × 3
553 of 9 × 1
778 of 8 × 1
871 of 7 × 1
1 of 3 × 1
1146 of 10 × 4
901 of 5 × 1
346 of 2 × 2
692 of 2 × 1
722 of 1 × 1

Set G

422 of 7 × 6
866 of 9 × 3
650 of 10 × 9
393 of 10 × 8
515 of 9 × 4
521 of 6 × 4
212 of 9 × 2
728 of 4 × 3
54 of 6 × 2
529 of 9 × 1
173 of 7 × 1
431 of 6 × 1
720 of 10 × 5
1417 of 10 × 1
910 of 4 × 2
911 of 2 × 1

In all cases the sheet size is 20 × 20.

Table 6.5.2

Results of test runs

Data Set	CPU time in secs	Number of sheets cut	Units of scrap	Number of full sheets waste	Percentage sheets waste
1	224	476	107	0	0
2	253	836	320	0	0
3	355	802	370	0	0
4	204	802	298	0	0
5	2059	1077	688	1	.09
6	286	846	37	0	0
7	4567	845	2369	5	.59
8	387	566	83	0	0
9	477	614	141	0	0
A	276	460	197	0	0
B	710	660	17	0	0
C	1709	737	98	0	0
D	533	923	120	0	0
E	502	472	389	0	0
F	785	585	96	0	0
G	183	608	80	0	0

Chapter 7 A 2-dimensional trim-loss problem with sequencing constraints

7.1 Statement of the problem

The material under consideration is glass. Each order received from a customer consists of a demand for specified numbers (demands) of pieces of each of one or more shapes, a shape being specified by its length (longer dimension) and breadth (shorter dimension) and the order to which it belongs. Each order is designated by a code. If the order is for glass with high quality edges, the first character of the code is "F". These are referred to as type F orders, and the remainder are referred to as type M orders.

The cutting of the stock sheets is a two stage process, and corresponding to each stage of the cutting there is a stripping process, to produce edges of the required quality, that may be manual or automatic. During the first stage of the cutting the sheet is moving in the direction of its longitudinal axis and may be considered to have a "leading" and a "trailing" edge with respect to this motion. The first set of cuts are perpendicular to the direction of motion and are cross cuts. The sub-sheets resulting from the first stage of cutting will be cut into pieces according to the corresponding sub-patterns.

Operational constraints on the design of cutting patterns due to the design of the cutting machinery are:

i) A sub-pattern may contain at most two shapes,

ii) The number of sub-patterns in a pattern may not

exceed five, and if there are five two must be the same,

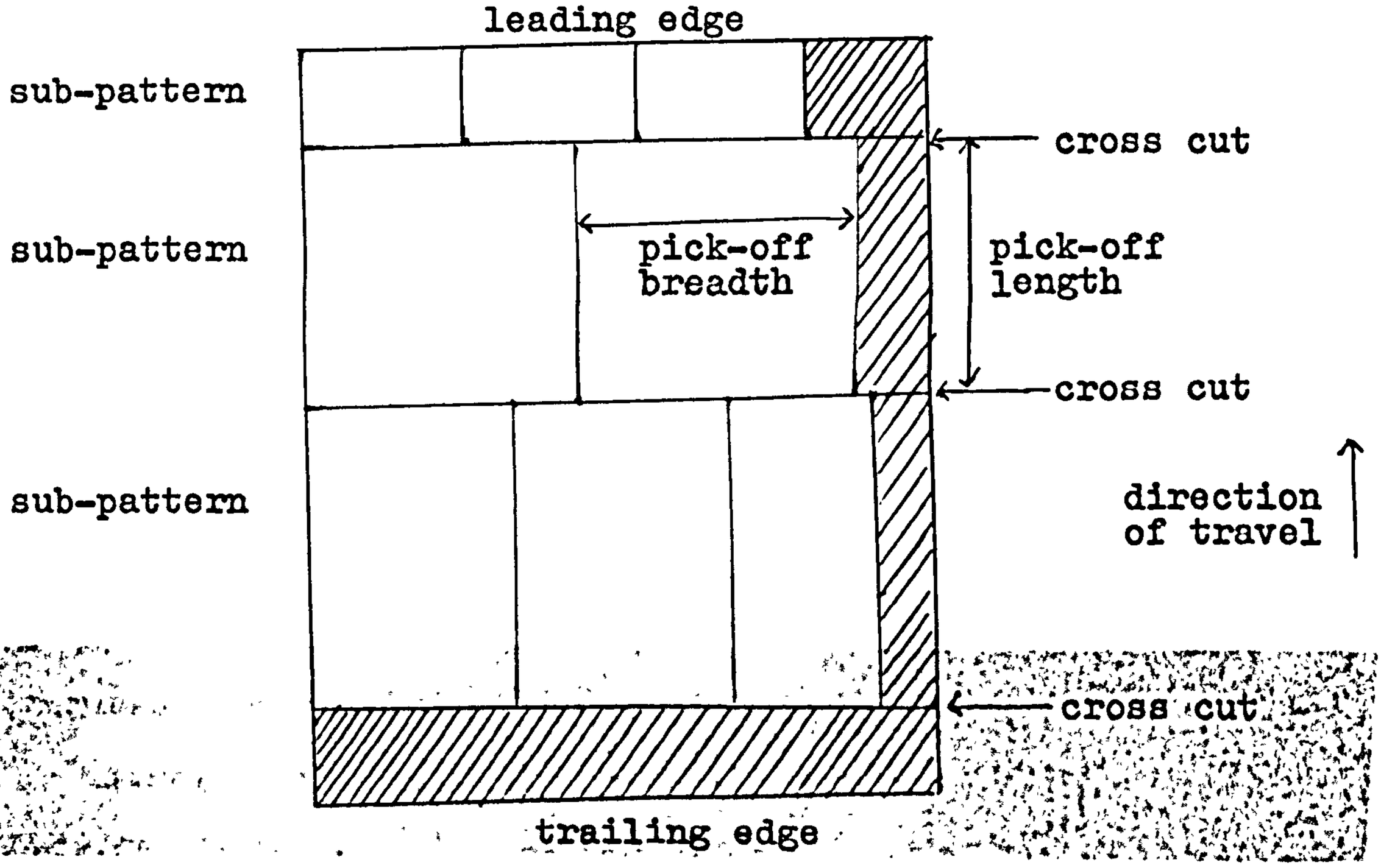
iii) If the leading sub-pattern includes type F pieces then the first cross cut must be at least 626mm from the leading edge of the sheet, and in any case this cut must be at least 432mm from the leading edge of the sheet,

iv) Any sub-pattern other than the leading one must have a length before stripping of at least 813mm,

v) If the trailing sub-pattern includes type F pieces then there must be a cross cut at its trailing edge at least 203mm from the trailing edge of the sheet.

The pick-off length of a sub-sheet or piece is its dimension in the direction of the sheet from which it was cut, its other dimension being its pick-off breadth (see figure 7.1.1). If the stripping following the first stage cutting is manual the pick-off length of the sub-sheet is reduced by 25mm; if it is automatic the reduction is 76mm. However, if the trailing sub-pattern contains only type M pieces it need not undergo such stripping.

In the stripping following the second-stage cutting the pick-off breadth of a type M piece is reduced by 51mm. The pick-off breadth of a type F piece is reduced by 152mm if it is stripped manually or by 302mm if it is stripped automatically. A piece whose pick-off length is greater than 2540mm must be stripped manually at this stage.



Typical cutting pattern

Figure 7.1.1

The pieces resulting from the cutting and stripping process are picked off for despatch from one of four legs, a leg being a conveyor belt with its associated washing and handling facilities. All the pieces resulting from the cutting of a sub-pattern must be routed to the same leg and the routing of pieces to legs must be consistent with the dimension constraints shown in table 7.1.1, and also with the requirement that type F pieces must be picked off from legs with washing facilities. Hence any type F piece of length exceeding 2540mm must be cut with its length oriented parallel to the sheet length. There is a further restriction that pieces for

not more than three different orders may be picked off any one leg during the cutting of any one pattern.

Table 7.1.1

Maximum dimensions for pick-off

Leg	Length(mm)	Breadth(mm)	Washing facilities
D	2540	1219	Yes
C	2540	1219	Yes
A	3048	3048	No
B	3048	2540	Yes

The restrictions on the design of the individual cutting patterns having been described, it is now necessary to consider how a set of cutting patterns may be arranged to provide instructions for cutting all the orders in an order list.

Initially a cutting pattern is selected. It will specify the size of stock sheet to be used and the pieces to be cut from this sheet. For each shape, s, in the pattern, there is a number n_s, such that if the pattern is repeated n_s times the demand for that shape will be satisfied, whilst if the pattern is only repeated n_s-1 times this will not be the case. The number of stock sheets cut to this pattern will be the smallest of the n_s's.

When this has been done, the original order list will be converted to a new one reflecting the demands that have not yet been satisfied. A new cutting pattern is selected and the process repeated until all the demands have been satisfied.

Operational costs associated with the picking off and packing of orders will affect the choice of the next cutting pattern to be used. The next cutting pattern should include all the shapes in the previous cutting pattern the demands for which were not completed with the use of that pattern. In addition, if there were any shapes, the demands for which were satisfied at this point, belonging to orders which include other shapes, the demands for which have not yet been satisfied, then such a shape from each such order should be included in the new cutting pattern. For each failure to make the appropriate inclusion in the new cutting pattern the sequence break cost is incurred.

There is also an absolute constraint on the sequence of cutting patterns. If a shape belongs to a loose load order, which will have a code starting with the characters "FD", then it must not be cut until the demands for all larger shapes in that order have been satisfied.

Table 7.1.2 shows the dimensions of the stock sheets. A cutting pattern defines implicitly the size of stock sheet on which it is to be used.

Table 7.1.2

Sizes of stock sheets

Length(mm)	Breadth(mm)
4648	2540
4699	2616
4648	3048

7.2 Choice of method

Because of the sequencing constraints it is clear that the pattern sequence must be generated in a stepwise manner. Each pattern is selected as being the best continuation of the sequence in the context of the state of the order list and the sequencing constraints resulting from the use of the previous pattern. Good results were obtained for the abstract 2-dimensional trim-loss problem by applying preferred reduction within non-backtracking search, which generated patterns in a stepwise manner. It therefore appears appropriate to apply this method to the present problem.

In section 7.3 some remarks are made about the representation in terms of a state-space of the stepwise generation of a pattern sequence. Section 7.4 outlines the problem reduction process used to generate a pattern in the sequence. In section 7.5 a program applying these ideas is described. Section 7.6 presents results obtained from this program and in section 7.7 some conclusions are drawn as to the success of the approach taken to the problem.

7.3 The top-level state-space

The process of choosing and utilizing cutting patterns described above can be represented in terms of a state-space. The states are lists of unsatisfied orders, and the operators are cutting patterns together with the number of times they are used. The start state is the original order list and the goal state the empty order list. It is required to find the path between the two for which the combined cost of trim-loss and sequence breaks

is minimal.

In order that the time taken by a program to compute a sequence of cutting patterns may be kept within reasonable bounds, backtracking must be restricted. A certain amount of time is allocated for the generation of possible first patterns. At the end of this time the "best" of the generated patterns is selected. The number of times it is to be used is calculated, the demands in the order list are adjusted to take account of the pieces cut by the use of this pattern, and the adjusted order list together with the first pattern defines the starting point for the generation of possible second patterns. The process is repeated until all the demands in the order list have been satisfied.

This method of working corresponds to the development of a path across the state-space graph without backtracking. The generation of a subset of the possible cutting patterns at each step corresponds to partial development of the corresponding node, and the selection of the "best" pattern represents the application of an evaluation function. The quality of solutions generated will be limited by the correctness of the partial development (whether an operator corresponding to an arc from this node belonging to a solution path is generated) and the accuracy of the evaluation function (whether when generated such an operator is correctly identified).

The generation of the set of cutting patterns from which the "best" is selected is done in the work presented here by a problem reduction technique. This is

described below.

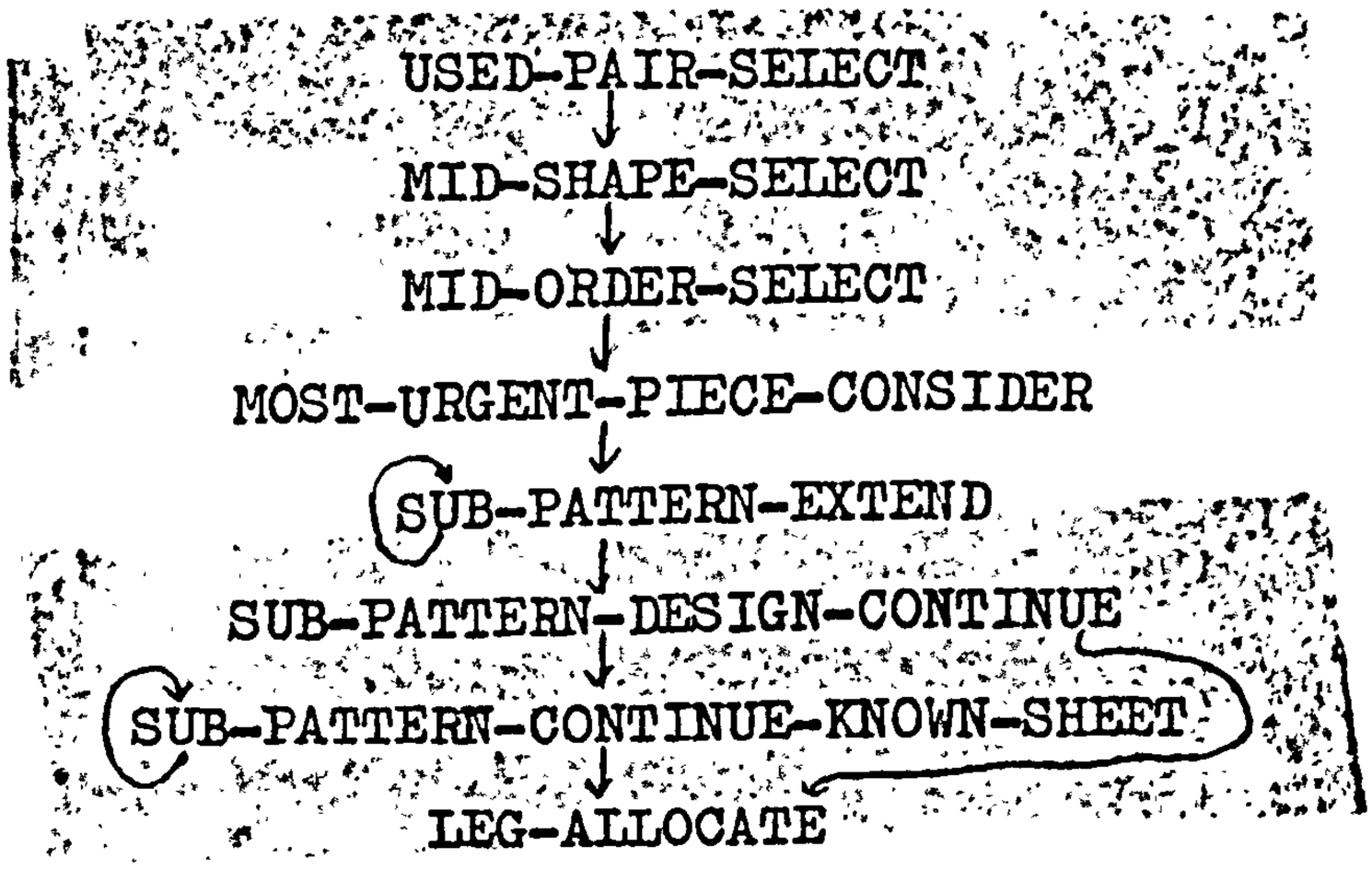
7.4 Structure of the problem reduction

Each node of the problem reduction tree includes in its label a local variable list which is initially a subgoal description and the name of a function that is capable of generating a sequence of alternatives for the reduction of that subgoal. For a primitive subgoal such a reduction is a solution. An occurrence of a function at a node will be referred to as an instance of that function.

In an iteration of the problem reduction process, a node of the problem reduction tree is selected. The function at a node will generate at successive invocations a sequence of alternative reductions. When a node is selected the function named in its label is invoked, and will give the next reduction in the sequence. If the subgoal is not primitive, it may create a new node in whose label are included the subgoal resulting from the reduction and the name of the function to be used for reducing this subgoal, whilst if the subgoal is primitive it may generate a solution of it. Alternatively, in either case it may report that the sequence of reductions has been exhausted. Iterations of the process continue until what is regarded as an adequate number of alternative solutions of the goal have been generated.

It has been said that when a function creates a node it places a function name in its label. Figure 7.4.1 shows the relationships of the subgoal processing functions in this respect. An arrow directed from one function name to another indicates that the label of a

node created by the first may include the name of the second. Where more than one arrow leaves the name of a function, this indicates that a choice of functions is made according to the features of the subgoal set up.



Hierarchy of subgoal processing functions

Figure 7.4.1

Consider the situation that can occur at the generation of any cutting pattern other than the first. The previous cutting pattern may have included one or more sub-patterns that each included two shapes (pairs). If the demand for a shape belonging to a pair has not yet been satisfied, then there is a choice to be made as to whether it should belong to a pair with the same or a related partner in the proposed new pattern. The choices on the retention of pairings are made in USED-PAIR-SELECT.

There may be a number of shapes included in the previous pattern the demands for which have not yet been satisfied. Some of these may have been included in the proposed pattern by choices made in USED-PAIR-SELECT. The choices as to which of the remainder should be included,

and what their orientations should be, in the proposed pattern, are made in MID-SHAPE-SELECT.

There may be one or more shapes, the demands for which were satisfied at the end of the use of the previous cutting pattern, that belong to orders containing shapes the demands for which have not yet been satisfied. In MID-ORDER-SELECT the choices are made as to which shapes, if any, from such orders should be included in the proposed pattern, and what their orientations should be.

For each shape in the order list there is calculated an urgency. When a pattern is being generated there is a shape in the order list, the demand for which has not yet been satisfied, which has maximal urgency. If this shape has not yet been included in the proposed pattern, the choice is made in MOST-URGENT-PIECE-CONSIDER as to whether it should be, and, if so, what its orientation should be.

The choices made in MID-SHAPE-SELECT, MID-ORDER-SELECT and MOST-URGENT-PIECE-CONSIDER are that shapes shall be included in a pattern in separate sub-patterns. Each instance of SUB-PATTERN-EXTEND is concerned with one such sub-pattern and chooses whether it should include one or two shapes. If the choice is for two shapes, the choice of the the second shape is made.

The choice of stock sheet size to be used by the proposed pattern is made in SUB-PATTERN-DESIGN-CONTINUE. A further sub-pattern may also be included in the pattern by this function.

Each instance of SUB-PATTERN-CONTINUE-KNOWN-SHEET

chooses whether a further sub-pattern should be included in a partially completed pattern, and, if so, what this sub-pattern should be.

In LEG-ALLOCATE an allocation of sub-patterns to legs is made if a feasible one exists. If it does not, then the proposed pattern is ruled out of further consideration, otherwise an evaluation is made of the "merit" of the proposed pattern. The allocation of sub-patterns to legs is a primitive subgoal.

7.5 Details of a program

7.5.1 Scrap evaluation

Material stripped from the cut sub-pattern or piece to produce the required edges is not part of the trim-loss, or scrap. If the difference between the dimension of the initial cut and the required final dimension is less than that required for automatic stripping, thus making manual stripping necessary, no trim-loss is deemed to have occurred. Any excess of the difference over that required for automatic stripping is deemed to constitute trim-loss. Trim-loss is measured in square centimetres.

7.5.2 Orientations

A piece being cut from a stock sheet may be oriented in one of two ways. If the length of the piece is parallel to the length of the sheet, the piece is said to be placed along the sheet, otherwise it is said to be across the sheet. The restrictions on cutting pattern design may force a shape always to be placed along the sheet, whilst square pieces always effectively have this orientation. Such pieces are unswingable.

For each shape a preferred orientation can be defined. For an unswingable shape this is necessarily along the sheet. For any other shape all possible sub-patterns that could be cut from any stock sheet and which include this shape with either orientation are considered. The orientation in that sub-pattern which gives least trim-loss within the sub-pattern is the preferred orientation. If the same minimal internal trim-loss would result from two sub-patterns in which the shape has different orientations then the across the sheet orientation is preferred, since placing the shape with this orientation leads to a smaller reduction in the number of choices as to how the remainder of a stock sheet is to be cut.

7.5.3 Urgency and minimum scrap

During the initial processing of the order list two quantities are calculated for each shape. These are referred to as urgency and minimum scrap.

The urgency of a shape reflects the difficulty of designing a cutting pattern including that shape when sequencing desiderata are applied or the order list has been depleted from its initial status. In other words, this is a quantification of the desirability that the shape should feature in the early cutting patterns of the solution sequence.

In the initial calculation of urgency two quantities, LF and BF, are used. These are defined by:

LF = (4445 - minimal length of non-trailing sub-pattern including this shape oriented along

the sheet) ÷ 838

BF = number of pieces of this shape in a sub-pattern consisting only of this shape oriented along the sheet cut from the broadest stock sheet and assuming manual stripping

LF is an estimate of the number of sub-patterns other than that containing the shape under consideration that will occur in patterns containing this shape. "÷" denotes integer division.

Two special cases are identified. If the shape must be cut in a leading sub-pattern then its urgency will be:

(5000 + demand for this shape) ÷ BF

If the shape is type F with a length of more than 2540mm then its urgency will be:

(<u>if</u> LF=1 <u>then</u> 5000 <u>else</u> 20000 <u>close</u> + 100 * demand for this shape) ÷ BF

The first case must be distinguished because if such a shape were left until late in the cutting pattern sequence this might be forced to finish with patterns consisting only of leading sub-patterns, with consequent extremely high trim-loss. The second case is distinguished because of the difficulty of designing sub-patterns including such shapes if sequencing desiderata require the inclusion of several other shapes.

The urgency of any other shapes is based on the amount of trim-loss that would result from the most economic choice of pattern containing only this shape. This is divided by the number of pieces in the pattern. It is further divided by LF*LF*BF if the shape is square and LF*LF*BF*BF otherwise. This division factor reflects

the ease of including the shape in an economic cutting pattern. The resultant quantity is then multiplied by (demand for this shape * 100) to give the urgency. The multiplication factor particularly raises the urgency of shapes that will be relatively difficult to include in an economic cutting pattern and will be continued over several cutting patterns. This means that they are made likely to occur in cutting patterns designed when more of the order list is not yet satisfied and there is therefore a greater range of choices for each of the cutting patterns in which they will be included.

After the initial calculations the urgencies are adjusted to take account of sequencing problems. If the urgency of a shape in a loose load order is less than that of a smaller shape in the same order, the urgency of the larger is made equal to the urgency of the smaller plus 2. This takes account of the fact that the smaller cannot be cut before the larger and therefore its own higher urgency would still not lead it to be introduced into an early cutting pattern. If a shape with urgency greater than 10000 occurs in the same order as a smaller shape, 5000 is subtracted from its urgency. Suppose two shapes each of length L occur in the order list, one in an order by itself and the other in an order with a smaller shape. If the one which occurs by itself is included in a pattern, then the next pattern may include the other together with the other sub-patterns of the previous pattern. If they are taken in the reverse order then sequencing desiderata will lead to the smaller shape of the second order being included in the next pattern

and it is almost certain that other new sub-patterns will be introduced. The need to include these in the following pattern will make it impossible to include the shape of the first order.

For some shapes it will be the case that whatever sub-pattern they are included in there will be trim-loss resulting from the cutting of that sub-pattern. For each shape all possible sub-patterns in which it might be included are considered. For some sub-pattern the quotient of the trim-loss resulting from the cutting of that sub-pattern and the number of pieces of the shape under consideration in the sub-pattern is minimal. This minimal quotient is the minimum scrap for that shape. Note that the sum of the terms (minimum scrap * demand) over all the shapes in an order list provides a lower bound on the trim-loss resulting from the cutting of that order list.

7.5.4 Control structures

The control of generation of possible cutting patterns is based on a representation of the subgoal tree. Each node of the tree has attached to it a label including a cost, the name of a subgoal processing function, an entry point and local variable list for that function, and a flag indicating whether the node is still active.

At each iteration of the generation process the tree is scanned to find the active node with lowest cost. The function attached to this node is then entered at the current entry point, using the current variable list. It will execute until one of three conditions occurs:

i) if the function is LEG-ALLOCATE, then it either reports that there is no possible allocation of sub-patterns to legs for a proposed pattern, or it returns a possible pattern with its merit, otherwise

ii) it sets up a new subgoal, the node representing which will have initial cost 0, or

iii) it reports that no further subgoals can be set up and the current node should be labelled inactive.

An addition, of 10 unless otherwise specified, is then made to the cost of the node and the next iteration commenced. The size of the increment reflects the number of alternative reductions of the subgoal it is thought should be attempted. The larger the increment, the less frequently attempts will be made to reduce the subgoal. The generation process is terminated after 100 iterations if it has not exhausted all alternatives previously.

The merit of a pattern is reported by LEG-ALLOCATE in the form of three integers. SCRAP is the trim-loss per sheet resulting from the use of the pattern, MERIT is an adjustment factor indicating the merit of the pattern sequence that would follow the pattern, and SEQ-BREAK-COUNT is an estimator that, when scaled, indicates the cost of sequence breaks that may result from the use of the pattern. The quantity

SCRAP - MERIT + SEQ-BREAK-COUNT * SEQ-BREAK-WEIGHT ,

where SEQ-BREAK-WEIGHT is set to 1,000,000 to indicate the total undesirability of sequence breaks, is calculated for each proposed pattern. At the end of the generation process the pattern for which it was lowest is

chosen as the next pattern to be included in the sequence.

When a pattern has been selected, the number of times it is to be used is calculated and the order list adjusted accordingly. Three lists are constructed to be used in the application of the sequencing desiderata to the design of the next pattern.

MIDSHAPE is a list of the shapes which occurred in the last cutting pattern the demands for which have not yet been satisfied. It is sorted on the lengths of the shapes, and within this on their breadths. MIDORDER is a list of the codes of shapes, the demand for which was satisfied by the cutting of the last pattern, belonging to orders in which there are other shapes the demand for which has not yet been satisfied.

If the demands for neither of the shapes in a pair in the last pattern have been satisfied, then the pairing of these shapes is recorded in the list USED-PAIRINGS. If the demand for one shape in a pair has been satisfied, the other shapes in the same order are inspected. If there is outstanding demand for one that can be formed into a pair with the other shape in the pair under consideration, then a pairing of these two shapes is recorded in USED-PAIRINGS. If the demands for both shapes in a pair were simultaneously satisfied, the remaining shapes in both orders for which demand has not been satisfied are inspected for possible pairings of one shape from each order. If such a pairing exists, it is recorded in USED-PAIRINGS.

7.5.5 USED-PAIR-SELECT

USED-PAIR-SELECT is the function which is used to reduce the goal of "design a pattern". A selection is made of the pairings in USED-PAIRINGS; these will lead to pairs in the resultant pattern. The first entry to USED-PAIR-SELECT selects all the pairings in USED-PAIRINGS. Subsequent entries will select subsets that get progressivjely smaller. The statement that these pairings shall occur in the pattern is the start of a proposal for the design of a pattern which will be passed to MID-SHAPE-SELECT. After an entry to USED-PAIR-SELECT, 40 is added to the cost in the label of the corresponding node on the subgoal tree.

7.5.6 MID-SHAPE-SELECT

MID-SHAPE-SELECT adds to the proposal statements that shapes occurring in MIDSHAPE and not involved in the included pairings shall occur in separate sub-patterns of the pattern and gives the orientation of these shapes. If MIDORDER is empty then the sequence of alternatives generated in MID-SHAPE-SELECT starts with that including all shapes with their preferred orientations. The generation then gives fewer and fewer of the shapes their preferred orientation, then repeats the sequence for subsets which get progressively smaller. Some of the alternatives generated may violate the constraints on pattern design imposed by the dimensions of the stock sheets. These alternatives are ignored for the purposes of subgoal creation.

If MIDORDER is not empty then an attempt is made to

find orientations of all the shapes which is consistent with some shape from each of the orders in MIDORDER being included in the pattern, each such shape occurring in a distinct sub-pattern. If such a set of orientations is found, it is used as the starting point for the sequence of alternatives, which again includes all orientations of all the pieces before starting with subsets of them.

The ordering of MIDSHAPE means that it is the alternatives with small shapes not having their preferred orientations which are generated first. These alternatives are more likely to produce good pattern designs as large pieces not having their preferred orientations are extremely likely to require unnecessary trim-loss. When a subgoal is created by MID-SHAPE-SELECT, 20 is added to the cost in the label of the corresponding node.

7.5.7 MID-ORDER-SELECT

To the proposal for pattern design received from MID-SHAPE-SELECT, MID-ORDER-SELECT adds statements that shapes belonging to the orders in MIDORDER shall occur in separate sub-patterns and gives the orientation of the shapes. The generation of alternatives starts with the selection of one shape from each order not already represented by a shape included by USED-PAIR-SELECT. The first choice of orientation is that all shapes should have their preferred orientations. Generation proceeds through the choices of orientation for this set of shapes, then through the choices of orientation for other choices of shapes representing the orders, then by a similar process through subsets of the orders.

7.5.8 MOST-URGENT-PIECE-CONSIDER

If the shape with unsatisfied demand for which the urgency is highest has not yet been included in the proposal for pattern design, the decision as to whether it should occur in a separate sub-pattern is made in MOST-URGENT-PIECE-CONSIDER. The order of generation of alternatives is: that it should occur with its preferred orientation, that it should occur with the other orientation, then, finally, that it should not occur.

7.5.9 SUB-PATTERN-EXTEND

Decisions are made in MID-SHAPE-SELECT, MID-ORDER-SELECT and MOST-URGENT-PIECE-CONSIDER that certain shapes should occur in separate sub-patterns in the final pattern. In SUB-PATTERN-EXTEND one such sub-pattern is considered, and the decision made as to whether the shape should occur by itself, or be one of a pair. The first alternative generated is that the shape should occur by itself. The subsequent ones are the feasible pairings, in order of urgency of the second shape to be included in the pair.

If there are other such sub-patterns which have not been considered then the node created on the subgoal tree will have SUB-PATTERN-EXTEND as the function attached to it and the initial local variable list will include an indication of which of the other sub-patterns is to be considered by that instance of SUB-PATTERN-EXTEND. When all such sub-patterns have been considered, the proposal for pattern design is passed to SUB-PATTERN-DESIGN-CONTINUE.

7.5.10 SUB-PATTERN-DESIGN-CONTINUE

The decisions taken in SUB-PATTERN-DESIGN-CONTINUE may be considered to be nested. The most inclusive is that as to the size of stock sheet from which the pattern being designed should be cut. The proposal for pattern design received from SUB-PATTERN-EXTEND states the shapes that shall occur in each of a number of sub-patterns. For each stock sheet size the manner in which each sub-pattern can be constructed so as to include the stated shapes with minimum trim-loss is determined. The aggregates of these minimum trim-losses for each sheet size are calculated and the sheets ranked with the lowest aggregate indicating the most preferred size.

The next level of decision is concerned with the addition, if this is possible, of a further sub-pattern to the partial cutting pattern designed so far. The decision is taken in the form of stating a shape which should occur with a given orientation in that sub-pattern. The order of preference of shapes and orientations is given by estimation of the amount of trim-loss likely to result from the inclusion of each oriented shape, i.e. a shape with its orientation specified, in the next sub-pattern. This estimation is converted into a numeric value, V, of which the highest value denotes the most preferred oriented shape, in the following way:

i) Let V take the value 200,000. If the shape is already included in the partial cutting pattern, add 10 to V. This discriminates in favour of such a shape where another shape of the same dimensions

(belonging to a different order) exists. Inclusion of the second shape would lead to unnecessary complications in the pattern sequencing.

ii) Within any sub-pattern including the given oriented shape, the breadth, T, of the scrap glass can be calculated. Let R be the minimum of the value of T for the sub-pattern including only that oriented shape and T+200 for any sub-pattern in which the oriented shape is one of a pair. Again note is being taken of the fact that the more shapes there are included in this cutting pattern, the more desiderata will be imposed on the design of the next. Let F be the number of pieces of the given shape included in the sub-pattern associated with the minimum value of R. Calculate

(R ÷ 10) * (length of sub-pattern ÷ 10) - F * (minimum scrap for this piece + urgency for this piece)

Subtract the result from V.

iii) Let R be the usable length of that part of the sheet remaining after the sub-patterns already specified and a sub-pattern including the given oriented shape have been cut. Consider the possible lengths of sub-patterns that might be cut from this remaining part. Let T be the shortest possible length of the part that could remain after one had been cut. If T+200 is less than R, replace R by T+200. Calculate

(R ÷ 10) * (width of sheet ÷ 10)

and subtract the result from V. Preference is being given here to the cutting of large shapes. Premature

inclusion of small shapes in cutting patterns causes the sequence to end with patterns that include only large pieces and have a high trim-loss.

The final alternative in this set of decisions is not to cut an additional sub-pattern.

Once it has been decided which oriented shape is to be included in the sub-pattern, the sub-pattern must be designed. The first leg, using the ordering of table 7.1.1, that can be used to pick off the given oriented shape is determined. The sub-pattern including the given oriented shape and not including an oriented shape that cannot be picked off this leg which results in minimum trim-loss is determined. This is the first alternative at this level of decision. The legs later in the ordering are then considered in order. If a more economic sub-pattern would result from assuming pick off from such a leg then that sub-pattern is an alternative.

In the overall generation of alternatives, when all possibilities at a less inclusive level have been considered the next alternative at the next more inclusive level is used. If there is a possibility of adding a further sub-pattern to the existing partial cutting pattern, it is passed to SUB-PATTERN-CONTINUE-KNOWN-SHEET. Otherwise it is passed to LEG-ALLOCATE.

7.5.11 SUB-PATTERN-CONTINUE-KNOWN-SHEET

The generation of alternatives in SUB-PATTERN-CONTINUE-KNOWN-SHEET is similar to that in SUB-PATTERN-DESIGN-CONTINUE, except that the sheet size has already been decided on, so this level of choice does

not exist.

7.5.12 LEG-ALLOCATE

LEG-ALLOCATE receives a cutting pattern. It calculates SEQ-BREAK-COUNT and MERIT (see section 7.5.4) and allocates the sub-patterns to legs.

At the beginning of the calculation of SEQ-BREAK-COUNT it has value 0. For each shape in MIDSHAPE and each order in MIDORDER not represented in the cutting pattern, 2 is added to it. If one shape of a pair in this pattern was cut in the previous pattern then sequencing desiderata mean that both shapes should be picked off the leg from which the first was previously picked off. If any shape belonging to the order that the second shape belongs to cannot be picked off, in either orientation if it is not unswingable, from this leg then there is a possibility of a sequence break at some future time and 1 is added to SEQ-BREAK-COUNT.

Other calculations involving SEQ-BREAK-COUNT and MERIT consider the possibility that the next pattern in the sequence will differ from the present one only in one sub-pattern. For a pair this means that sequencing desiderata require that any remaining shape belonging to one order involved in the pair should be capable of forming a pair with any remaining shape belonging to the other order, the sub-pattern length of any such pair being no greater than that of the present one. If this is not the case, 1 is added to SEQ-BREAK-COUNT. At the same time note is made as to the maximum necessary width of such sub-patterns, and whether any require to be cut at the leading edge of a sheet.

The initial value of MERIT is the sum of the urgency and minimum scrap for every piece in the pattern. An upper bound on the length of the sheet size that would be required to satisfy sequencing desiderata in subsequent patterns is obtained by adding together the lengths of the distinct sub-patterns and any excess that must result from the replacement of a shape occurring by itself in a sub-pattern by another unsatisfied shape in the same order which could only occur in a longer sub-pattern. If there is a shape occurring by itself in a sub-pattern belonging to an order in which another unsatisfied shape can only occur in a sub-pattern that is no longer if it is oriented across the sheet then 5000 is subtracted from MERIT. This reflects the tendency of such orientations to be wasteful and also the loss of flexibility in the design of patterns when orientations of pieces are forced. Upper bounds on the width of the sheet size and on the number of sub-patterns requiring to be cut at the leading edge of a sheet that might occur simultaneously if sequencing desiderata were to be satisfied are calculated at the same time as the upper bound on length and using the same assumptions about sub-pattern design in subsequent patterns.

If the upper bounds on width and length together are not compatible with any stock sheet size, 1 is added to SEQ-BREAK-COUNT, to reflect the possibility of a future sequence break. If they are not compatible with the current sheet size, 5000 is subtracted from MERIT, to reflect the likelihood that subsequent patterns

satisfying the sequencing desiderata would result in high trim-loss. If the upper bound on the number of sub-patterns requiring to be cut from the leading edge of a sheet is greater than 1, the excess over 1 is added to SEQ-BREAK-COUNT.

For each sub-pattern there is a preferred leg from which it should be picked off. For a sub-pattern containing a shape belonging to an order that occurred in the previous pattern, it is the one from which pieces belonging to this order were picked off when that pattern was cut. Otherwise the preferred leg for a sub-pattern is the first leg in the order of table 7.1.1 from which any shape belonging to any order involved in that sub-pattern can be picked off, with either orientation if such a shape is not unswingable. The first attempt to assign sub-patterns to legs assigns each sub-pattern to its preferred leg. If such an assignment is not feasible, the preferences are relaxed, starting with the sub-patterns not involving shapes belonging to orders that occurred in the previous pattern. For each shape belonging to an order assigned to a leg in the previous pattern which is assigned to a different leg in this pattern, 1 is added to SEQ-BREAK-COUNT.

7.6 Results

A program using these techniques has been written in Algol 68-R (Woodward and Bond, 1972) and tested on an ICL 1906S using 16 sets of data (see appendix A). These data sets are random subsets of a set of data supplied by Pilkington Brothers Ltd. The results obtained from these test runs are shown in table 7.6.1. The derivation of

Table 7.6.1

Results of test runs - main problem

1	2	3	4	5	6
Data Set	Lower bound on trim-loss (sq cm x 10^7)	Actual trim-loss (sq cm x 10^7)	Trim-loss as %age of material used	Best known trim-loss (%age)	(difference between 4 & 5)
1	0.5	1.4	8.2	5.6	2.6
2	0.3	1.7	8.9	7.7	1.2
3	0.3	0.7	7.7	7.2	0.5
4	0.6	1.2	7.1	6.6	0.5
5	0.4	1.6	8.9	6.4	2.5
6	0.2	1.4	6.5	6.0	0.5
7	0.8	1.1	7.9	7.1	0.8
8	0.4	0.8	7.9	6.3	1.6
9	0.4	2.0	9.7	7.2	2.5
A	0.4	1.2	8.0	7.6	0.4
B	0.7	1.5	9.2	8.4	0.8
C	0.4	0.9	8.2	6.9	1.3
D	0.6	0.8	7.5	7.2	0.3
E	0.5	0.8	5.5	5.5	0.0
F	0.5	1.6	9.8	9.6	0.2
G	0.2	1.3	6.5	6.5	0.0
		Mean	8.0		1.0

Table 7.6.2

Results of test runs - subsidiary problems

	SEQ-BREAK-WEIGHT=1		Number of orders per leg =2	
Data Set	%age trim-loss	Number of sequence breaks	%age trim-loss	Number of sequence breaks
1	4.8	14	6.0	2
2	6.9	16	11.1	0
3	7.7	15	12.6	1
4	6.1	10	6.9	0
5	5.2	15	9.4	1
6	5.4	19	8.6	0
7	7.2	12	7.9	0
8	6.6	6	8.8	0
9	3.6	16	10.0	0
A	5.7	15	8.2	0
B	8.0	18	9.8	0
C	5.2	9	8.2	2
D	6.9	4	7.5	0
E	6.1	14	5.5	0
F	5.9	15	10.4	1
G	4.8	18	6.9	0

column 2 is given in section 7.5.3. During the development of the program a considerable number of heuristics were experimented with. Some of these produced very good results with some sets of data whilst performing unacceptably badly on others. These results were collated to produce column 5. Column 6 thus shows the known deficiencies of the program. The execution time of the program depends on size of the data set, the format of the output, and whether the object code incorporates run-time checking. However, 4 minutes may reasonably be regarded as an average execution time. No sequence breaks occur in any of these pattern sequences.

Table 7.6.2 shows the results when the program is used to attempt the solution of two related problems. If sequence breaks are considered to be of minor importance, the program can be run with SEQ-BREAK-WEIGHT (see section 7.5.4) set to 1. In the second case the sequencing desiderata of the main problem are imposed, but the number of orders that may be picked off one leg at one time is restricted to 2.

7.7 Conclusions

The program succeeds in producing sequences of cutting patterns with no sequence breaks, which was one of the design aims. Its trim-loss behaviour is known not to be optimal. However, on the test data used the percentage of material lost at scrap never rose to 10% and the average discrepancy between the solutions produced and the known best solutions was 1% of the material used. Whether this behaviour is good enough would depend on details of the industrial environment.

Further development of the technique would have to be closely linked to these.

The heuristic nature of the method used for avoiding sequence breaks is illustrated by their occasional occurrence in the results of the subsidiary problem with the number of orders picked off each leg restricted to 2. Even here they average only 0.5 per sequence and this suggests a superiority to the work of Dyson and Gregory (1974) in this area.

The extremely problem-dependent nature of the heuristics being used is illustrated by a comparison of the results in table 7.6.1 with those under SEQ-BREAK-WEIGHT=1 in table 7.6.2. As a particular example take data set 3, the results from which indicate that in the latter case a different treatment is necessary for pieces that must occur in leading sub-patterns.

Chapter 8 Efficacy of the methods

For convenience of reference in this chapter, the problems considered in the present work will be labelled A-E as follows:

A) abstract 1½-dimensional trim-loss problem (chapter 3),

B) 2-dimensional trim-loss problem with varying stock costs (chapter 4),

C) optimal network problem (chapter 5),

D) abstract 2-dimensional trim-loss problem (chapter 6),

E) 2-dimensional trim-loss problem with sequencing constraints (chapter 7).

The results obtained are good and indicate that heuristic search methods are appropriate to the solution of the problems considered. The quality of solutions obtained by methods of this type depends on the degree of use that can be made of problem-specific information. In the present work this is information about the geometry of the problems.

The geometric information is utilized in three ways. Firstly, it can be used to decide on the search method, as shown for example in section 6.1.2. All the problems required some adaptation of the basic methods of state-space search and problem reduction and some required methods to be combined. These adapted and combined search methods may well have more general applicability.

Secondly, the geometric information may be used to order the alternatives within a search. This ordering is necessary for the use of ordered operator search or

preferred reduction search. Thirdly, the geometric information may be used, as shown for example in section 3.3.3, to eliminate alternatives in a search.

For three of the problems considered (A, C and D) the methods applied produced solutions that were near perfect. For the remaining two (B and E) there was an apparently small, but perceptible, deviation from optimality. At present methods of getting better solutions to these problems are not apparent, and this is clearly an area for further research.

The question may be asked as to whether there is any difference in type between the two groups of problems. The answer would appear to lie in the perceived complexity of the problems. Development of an heuristic solution method is a process of man-machine interaction. Initially a structure is hypothesized for the solution method, a set of plausible heuristics incorporated into it, and the resultant computer program run. The details of the program are reviewed in the light of this run. The review may lead to a complete change in the structure of the program. This happened with problem D; the first solution method tried was an ordered operator search based on the method used on problem A. Certainly the review of results will lead to changes in the heuristics.

Each subsequent run will indicate inadequacies in the existing heuristics. The response to this situation may be to adjust the relative weightings of existing heuristics. On the other hand a completely new heuristic may be introduced. Some salient feature of the problem may be completely overlooked in the creation of the

initial set of heuristics, but its importance can become apparent from the consideration of poor program results.

Similarly, a program containing heuristics which in general perform unacceptably badly, or even logical errors, may produce a good solution for a particular set of data. An analysis of what makes this solution "good" may lead to the revaluing of existing heuristics or the introduction of new ones.

An important limiting factor, then, on the development of heuristic methods is the amount of insight that can be gained into the nature of good solutions. In the group of problems for which near perfect solutions were found are those for which it was possible to perceive the form of good solutions. For the other group it was not possible to specify this form tightly enough. Observe particularly that for problem E two different types of constraint (trim-loss and sequencing) are involved and the difficulty is in determining the nature of their interaction.

In extremis this limitation is in the human being writing the program, but the nature of the computing facilities available for program development will have a significant effect. Most importantly it should be noted that program development will involve a large number of runs to evaluate different combinations of heuristics. If the time taken by a single run is not small then a considerable amount of computer time will be required. The development process is one of man-machine interaction and the computing facilities must allow this in a satisfactory form.

The more information that is available about the intermediate steps performed by the program, the more it is possible to determine how they should be modified. The evaluation of an intermediate step may require considerable computation, and there may also be difficulty in displaying the features of the step. These considerations suggest that the availability of interactive graphics facilities would be useful in the development of the solution methods for complex problems.

Another aspect of the need for graphics facilities arises when the 3-dimensional analogues of the 2-dimensional problems investigated here are considered. The representation of solutions, let alone steps in their creation, raises problems in this area.

It is in the nature of the methods used that the details of the solutions they produce cannot be predicted. There is therefore no possibility of proving such a program correct or exhaustively testing it. In view of this it is even more essential than usual that the programming system being used not only be appropriate to the problem, but also provide adequate diagnostic aids. This point is expanded in appendix B.

In the same way that it is never certain with this type of method that the program implementing it is bug-free, it is never certain that the heuristics are totally stable. By a stable set of heuristics is meant here one which, whatever set of data is presented to the program in which it is embodied, will not produce seriously sub-optimal results. If such a method were

implemented in a practical case, it would be necessary to monitor its behaviour to ascertain whether such events were occurring, and, if so, to make adjustments to it.

References

Abraham,P.M., S.J.Kirby and Y.G.Ng (1976) Production schedule of a float glass process plant. Systems Engineering Project Report. University of Liverpool.

Adamowicz,M. and A.Albano (1972) A two-stage solution of the cutting stock problem, in Information Processing 71 (ed. C.V.Freiman), 1086-1091. Amsterdam: North-Holland.

Adamowicz,M. and A.Albano (1976) A solution of the rectangular cutting stock problem. IEEE Trans. Systems, Man, Cybernetics, SMC-6, 302-310.

Beale,E.M.L. (1970) Selecting an optimum subset, in Integer and Non-Linear Programming (ed. J.Abadie), 451-462. Amsterdam: North-Holland.

Boffey,T.B. and A.I.Hinxman (1976) An algorithm for finding p-medians, in Proceedings of EURO-II. Stockholm.

Boyce,D.E., A.Farhi and R.Weischedel (1973) Optimal network problem: a branch-and-bound algorithm. Environment and Planning 5, 519-533.

Brown,A.R. (1971) Optimum packing and depletion. London: Macdonald.

Burstall,R.M., J.S.Collins and R.J.Popplestone (1971) Programming in POP-2. Edinburgh: Edinburgh University Press.

Chambers,M.L., and R.G.Dyson (1976) The cutting stock problem in the flat glass industry - selection of stock sizes. Opl. Res. Q. 27, 949-957.

Christofides,N. (1977) Personal communication concerning paper in Opns. Res. 25.

Doran,J.E. and D.Michie (1966) Experiments with the graph traverser program. Proc. Roy. Soc. A 294, 235-259.

Dunn,R.D. (1972) POP-2/4100 Users Manual. School of Artificial Intelligence, University of Edinburgh.

Dyson,R.G. and A.S.Gregory (1974) The cutting stock problem in the flat glass industry. Opl. Res. Q. 25, 41-53.

Ernst,G. and A.Newell (1969) GPS: A case study in generality and problem solving. New York: Academic Press (ACM Monograph Series).

Escudero,L.F. and E.Garbayo (1973) The cutting stock problem: application of combinatorial techniques and mixed integer programming. VIII International Symposium on Mathematical Programming (unpublished).

Garfinkel,R.S. and G.L.Nemhauser (1972) Integer Programming. New York: John Wiley and Sons.

Gilmore,P.C. and R.E.Gomory (1961) A linear programming approach to the cutting-stock problem. Opns. Res. 9, 849-859.

Gilmore,P.C. and R.E.Gomory (1963) A linear programming approach to the cutting-stock problem - part II. Opns. Res. 11, 863-888.

Gilmore,P.C. and R.E.Gomory (1965) Multistage cutting stock problems of two and more dimensions. Opns. Res. 13, 94-120.

Gilmore,P.C. and R.E.Gomory (1966) The theory and computation of knapsack functions. Opns. Res. 14, 1045-1074.

Golomb,S.W. (1965) Polyominoes. New York: Charles Scribner's Sons.

Haessler,R.W. (1971) A heuristic programming solution to a non-linear cutting stock problem. Management Science 17, B-793 - B-802.

Haessler,R.W. (1975) Controlling cutting pattern changes in one-dimensional trim problems. Opns. Res. 23, 483-493.

Hahn,S.G. (1968) On the optimal cutting of defective sheets. Opns. Res. 16, 1100-1114.

Haims,M.J. and H.Freeman (1970) A multistage solution of the template layout problem. IEEE Trans. Sys. Sci. Cybernetics SSC-6, 145-151.

Harary,F. (1969) Graph Theory. Reading, Massachusetts: Addison-Wesley.

Harris,L.R. (1973) The bandwidth heuristic search, in Advance papers of 3rd International Joint Conference on Artificial Intelligence (IJCAI-3), 23-29. Stanford, U.S.A.

Hart,P.E., N.J.Nilsson and B.Raphael (1968) A formal basis for the heuristic determination of minimal cost paths. IEEE Trans. Sys. Sci. Cybernetics SSC-4, 100-107.

Herz,J.C. (1972) Recursive computational procedure for two-dimensional stock cutting. IBM J. Res. Develop. 16, 462-469.

Johnson,W.W. and W.E.Story (1879) Notes on the '15' puzzle. Am. J. Math. 2, 397-404.

Kruskal,J.B.,jr. (1956) On the shortest spanning subtree of a graph and the travelling salesman problem. Proc. Amer. Math. Soc. 7, 48-50.

Lawler,E.L. and D.E.Wood (1966) Branch-and-bound methods: a survey. Opns. Res. 14, 699-719.

Lindley,D.V. and J.C.P.Miller (1958) Cambridge Elementary Statistical Tables. Cambridge: Cambridge University Press.

Little,J.D.C., K.G.Murty, D.W.Sweeney and C.Karel (1963) An algorithm for the travelling-salesman problem. Opns. Res. 11, 972-989.

Michie,D. (1967) Strategy-building with the graph traverser, in Machine Intelligence 1 (eds. N.L.Collins and D.Michie), 135-152. Edinburgh: Oliver and Boyd.

Michie,D. and R.Ross (1969) Experiments with the adaptive graph traverser, in Machine Intelligence 5 (eds. B.Meltzer and D.Michie), 301-318. Edinburgh: Edinburgh University Press.

Nilsson,N.J. Searching problem-solving and game-playing trees for minimal cost solutions, in Information Processing 68, (ed. A.J.H.Morrell), vol.2, 1556-1562. Amsterdam: North-Holland.

Nilsson,N.J. (1971) Problem-Solving Methods in Artificial Intelligence. New York: McGraw-Hill.

Pearman,A.D. (1974) Heuristic approaches to road network optimization. Engineering Optimization 1, 37-49.

Pfefferkorn,C.E. (1975) A heuristic problem solving design system for equipment or furniture layouts. C.A.C.M. 18, 286-297.

Pohl,I (1973) The avoidance of (relative) catastrophe, heuristic competence, genuine dynamic weighting and computational issues in heuristic problem solving, in Advance papers of 3rd International Joint Conference on Artificial Intelligence (IJCAI-3), 12-17. Stanford, U.S.A.

Sacerdoti,E.D. (1975) The non-linear nature of plans, in Advance papers of 4th International Joint Conference on Artificial Intelligence (IJCAI-4), 206-214. Tbilisi, U.S.S.R.

Slagle,J. (1963) A heuristic program that solves symbolic integration problems in freshman calculus. J.A.C.M. 10, 507-520.

Stephens,P.D. (1974) The IMP language and compiler. Comp. J. 17, 216-223.

Tait,P.G. (1880) Note on the theory of the '15 puzzle'. Proc. Roy. Soc. Edinb. 10, 664-665.

Woodward,P.M. and S.G.Bond (1972) Algol 68-R Users Guide.
London: H.M.S.O.

APPENDIX A Test data for 2-dimensional trim-loss problem with sequencing constraints

DATA SET 1

CØDE	LENGTH	WIDTH	DEMAND
BLANK	1219	610	1200
FA3860	3048	2100	54
FA8983	1923	1140	264
FA8983	1923	1440	99
FA8983	1930	840	165
FA9325	2490	1000	50
FA9325	2490	1100	25
FA9325	2490	1200	25
FC8006	720	720	330
FC9010	1200	600	291
FC9010	1520	440	190
FD6459	2540	1500	25
FD6459	2540	1600	25
FD6459	2540	1700	25
FD6459	2540	1800	175
FD8854	2440	1830	250
FD9286	2540	1800	300
FN3584	2490	1500	25
FN3584	3048	1500	20
FN3867	1930	1372	42
FN3867	2540	1524	56
FN3867	2743	2134	20
FN3913	2438	1880	242
FN4004	1219	610	24
FN4004	1219	762	60
FN4004	1219	1219	180
FN4004	1524	1219	214
FN4033	1219	610	1848
FN4033	1219	1219	92
FN4057	2032	914	38
FQ3778	2438	1524	50
ME2443	980	920	100
MF3473	960	720	45
MF3473	1520	920	100
MH0879	1320	700	100
MH2726	1560	620	240
MS3776	2280	2000	21
N3276	1220	915	28
N3283	2480	1500	20
PS1225	1219	610	50
PS1225	1250	457	115
STØCK	2490	1800	112

DATA SET 2

CØDE	LENGTH	WIDTH	DEMAND
BLANK	1219	1219	300
FA3864	3048	1800	66
FA9113	2660	1200	225
FA9418	2490	1000	140
FA9418	2490	1200	245
FA9418	2490	1300	35
FA9418	2490	1400	35
FA9418	2490	1500	140
FC8429	1200	600	488
FC8429	1520	440	295
FC9237	1520	900	96
FD8779	2540	1000	35
FD8779	2540	1200	164
FD8779	2540	1300	35
FD8779	2540	1400	35
FD9088	2540	1000	70
FD9088	2540	1100	35
FD9088	2540	1200	175
FD9088	2540	1400	70
FD9346	2540	1800	300
FN3763	3048	1829	20
FN3763	3048	2134	16
FN3875	2489	2134	18
FN3922	1829	1219	85
FN4006	1219	762	210
FN4006	1524	762	150
FN4006	1524	914	200
FN4034	1219	1219	406
FN4034	1829	1219	250
FP3704	1120	600	149
FP3704	1120	720	198
FP3704	1320	960	127
FP3704	1400	680	105
FP3704	1400	920	78
FQ3824	2134	1219	70
FQ3824	2438	1219	65
ME3520	1300	540	100
MF3485	860	660	129
MH0886	1340	920	60
MH4200	1120	680	39
MR3971	2540	1800	616
MS3929	1320	610	52
N3279	1220	1220	50
PS0107	2490	1800	58
PS1226	838	559	107
PS1226	1046	711	38
PS1226	1098	457	210
PS1226	1250	457	95
PS1226	1250	533	60
PS1226	1250	553	24

DATA SET 3

CØDE	LENGTH	WIDTH	DEMAND
BLANK	1524	914	150
FA9142	2490	1000	25
FA9142	2490	1200	25
FA9142	2490	1600	25
FA9142	2490	1800	25
FA9430	2490	1700	50
FA9430	3048	1600	50
FC8644	1000	620	120
FC9292	980	740	70
FC9292	1040	540	135
FD8780	2540	1500	75
FD8780	2540	1800	50
FD9123	2420	1700	43
FD9671	2540	1400	140
FN3818	2438	2134	30
FN3880	1930	864	84
FN4002	1829	1219	250
FN4007	1930	1168	125
FN4007	2032	813	170
FN4007	2032	914	110
FN4035	1524	914	241
FP3777	1524	914	130
FQ3845	1520	440	528
FQ4076	1524	1219	20
MF2694	1360	560	565
MF4166	840	680	100
MH2692	1220	1060	26
MQ3775	1920	1200	26
MS3325	762	762	1363
N3252	1220	610	20
N3280	1220	610	50
N3280	1220	762	33
N3280	1220	915	68
PS0143	1651	610	25
PS211	1200	600	216

DATA SET 4

CØDE	LENGTH	WIDTH	DEMAND
FA3783	2240	1500	108
FA8779	2240	1300	45
FA9143	2240	1000	30
FA9143	2240	1200	30
FA9143	2240	1600	30
FA9143	2240	1800	30
FC7792	760	600	140
FC7792	1200	900	86
FC8952	1520	440	522
FD6458	2540	1500	25
FD6458	2540	1600	25
FD6458	2540	1700	25
FD6458	2540	1800	175
FD8853	2440	1830	250
FD9152	2440	1830	500
FN3583	3048	1700	85
FN3853	2490	1700	22
FN3905	3048	1829	16
FN4003	1524	610	600
FN4003	1829	610	135
FN4008	2438	610	184
FN4008	2438	762	150
FN4008	2438	914	116
FN4008	2438	1219	95
FN4045	3048	2000	90
FQ3153	2490	1500	6
FQ3778	2438	1524	50
MF2695	900	600	228
MH0875	740	406	344
MH0875	1440	580	1012
MH2715	960	720	786
MH2715	1560	620	364
MH2715	2060	720	347
MQ3943	2520	1500	400
MS3449	1524	508	126
MS3449	1803	660	66
N3272	1800	1070	54
N3281	1830	610	33
N3281	1830	762	27
N3281	1830	915	23
N3281	1830	1220	37
PS1224	838	635	40
PS1224	1046	711	38
STØCK	2490	1500	10

DATA SET 5

CØDE	LENGTH	WIDTH	DEMAND
FN3853	2490	1700	22
FA8983	1923	1140	264
FA8983	1923	1440	99
FA8983	1930	840	165
FA7636	2240	1200	225
FQ3153	2490	1500	6
FD9123	2420	1700	43
PS0107	2490	1800	58
N3280	1220	610	50
N3280	1220	762	33
N3280	1220	915	68
FC9292	980	740	70
FC9292	1040	540	135
FA3860	3048	2100	54
MS3325	762	762	1363
MR3971	2540	1800	616
FC9010	1200	600	291
FC9010	1520	440	190
FQ3778	2438	1524	50
BLANK	1219	1219	300
MR3933	1020	406	520
FN4007	1930	1168	125
FN4007	2032	813	170
FN4007	2032	914	110
FP3777	1524	914	130
FD8780	2540	1500	75
FD8780	2540	1800	50
FN3875	2489	2134	18
FN3867	1930	1372	42
FN3867	2540	1524	56
FN3867	2743	2134	20
FD9346	2540	1800	300
FA9418	2490	1000	140
FA9418	2490	1200	245
FA9418	2490	1300	35
FA9418	2490	1400	35
FA9418	2490	1500	140
PS211	1200	600	216
PS1225	1219	610	50
PS1225	1250	457	115
MS3449	1524	508	126
MS3449	1803	660	66
FC8429	1200	600	488
FC8429	1520	440	295

DATA SET 6

CØDE	LENGTH	WIDTH	DEMAND
FN3818	2438	2134	30
FN3763	3048	1829	20
FN3763	3048	2134	16
ME2443	980	920	100
FD6459	2540	1500	25
FD6459	2540	1600	25
FD6459	2540	1700	25
FD6459	2540	1800	175
FA3864	3048	1800	66
N3283	2480	1500	20
FQ3824	2134	1219	70
FQ3824	2438	1219	65
FC9237	1520	900	96
FN3913	2438	1880	242
MQ3775	1920	1200	26
FA3783	2240	1500	108
FC8952	1520	440	522
FC8006	720	720	330
BLANK	1219	610	1200
MH2715	960	720	786
MH2715	1560	620	364
MH2715	2060	720	347
FN4006	1219	762	210
FN4006	1524	762	150
FN4006	1524	914	200
FP3704	1120	600	149
FP3704	1120	720	198
FP3704	1320	960	127
FP3704	1400	680	105
FP3704	1400	920	78
FN4057	2032	914	38
FN4004	1219	610	24
FN4004	1219	762	60
FN4004	1219	1219	180
FN4004	1524	1219	214
FD8779	2540	1000	35
FD8779	2540	1200	164
FD8779	2540	1300	35
FD8779	2540	1400	35
FN4034	1219	1219	406
FN4034	1829	1219	250
MS3776	2280	2000	21
FA9325	2490	1000	50
FA9325	2490	1100	25
FA9325	2490	1200	25
FN4033	1219	610	1848
FN4033	1219	1219	92
FD9152	2440	1830	500
FA9142	2490	1000	25
FA9142	2490	1200	25
FA9142	2490	1600	25
FA9142	2490	1800	25

DATA SET 7

CØDE	LENGTH	WIDTH	DEMAND
FA9113	2660	1200	225
PS1224	838	635	40
PS1224	1046	711	38
ME2438	1600	640	70
FD6458	2540	1500	25
FD6458	2540	1600	25
FD6458	2540	1700	25
FD6458	2540	1800	175
FN4008	2438	610	184
FN4008	2438	762	150
FN4008	2438	914	116
FN4008	2438	1219	95
N3281	1830	610	33
N3281	1830	762	27
N3281	1830	915	23
N3281	1830	1220	37
FN3922	1829	1219	85
FN3584	2490	1500	25
FN3584	3048	1500	20
MF3473	960	720	45
MF3473	1520	920	100
MH4200	1120	680	39
FN3905	3048	1829	16
N3276	1220	915	28
N3252	1220	610	20
MF2694	1360	560	565
FD9671	2540	1400	140
FD8854	2440	1830	250
FC7792	760	600	140
FC7792	1200	900	86
FN4045	3048	2000	90
FN4035	1524	914	241
STØCK	2490	1800	112
FD9288	2540	1800	300
FN4002	1829	1219	250
FA9143	2240	1000	30
FA9143	2240	1200	30
FA9143	2240	1600	30
FA9143	2240	1800	30
MH0886	1340	920	60
MH0875	740	406	344
MH0875	1440	580	1012

DATA SET 8

CØDE	LENGTH	WIDTH	DEMAND
FA8779	2240	1300	45
MF4166	840	680	100
FQ4076	1524	1219	20
MF3485	860	660	129
PS0143	1651	610	25
FQ3845	1520	440	528
N3279	1220	1220	50
FD9088	2540	1000	70
FD9088	2540	1100	35
FD9088	2540	1200	175
FD9088	2540	1400	70
MF2695	900	600	228
FN3583	3048	1700	85
BLANK	1524	914	150
N3272	1800	1070	54
MH2726	1560	620	240
ME3520	1300	540	100
MQ3943	2520	1500	400
FD8853	2440	1830	250
FA9430	2490	1700	50
FA9430	3048	1600	50
FN3880	1930	864	84
FN4003	1524	610	600
FN4003	1829	610	135
MS3929	1320	610	52
MH2692	1220	1060	26
STØCK	2490	1500	10
PS1226	838	559	107
PS1226	1046	711	38
PS1226	1098	457	210
PS1226	1250	457	95
PS1226	1250	533	60
PS1226	1250	553	24
MH0879	1320	700	100
FC8644	1000	620	120

DATA SET 9

CØDE	LENGTH	WIDTH	DEMAND
FN4057	2032	914	38
FC8429	1200	600	488
FC8429	1520	440	295
MF3485	860	660	129
FD9152	2440	1830	500
FA9418	2490	1000	140
FA9418	2490	1200	245
FA9418	2490	1300	35
FA9418	2490	1400	35
FA9418	2490	1500	140
FC7792	760	600	140
FC7792	1200	900	86
FC9237	1520	900	96
FN3905	3048	1829	16
FN4007	1930	1168	125
FN4007	2032	813	170
FN4007	2032	914	110
PS1225	1219	610	50
PS1225	1250	457	115
N3272	1800	1070	54
PS211	1200	600	216
FD9123	2420	1700	43
N3281	1830	610	33
N3281	1830	762	27
N3281	1830	915	23
N3281	1830	1220	37
FA3783	2240	1500	108
MH0875	740	406	344
MH0875	1440	580	1012
FD6458	2540	1500	25
FD6458	2540	1600	25
FD6458	2540	1700	25
FD6458	2540	1800	175
FN3913	2438	1880	242
FN3853	2490	1700	22
STØCK	2490	1800	112
FD8853	2440	1830	250
FN4034	1219	1219	406
FN4034	1829	1219	250
FA9143	2240	1000	30
FA9143	2240	1200	30
FA9143	2240	1600	30
FA9143	2240	1800	30
MS3325	762	762	1363
FN4002	1829	1219	250
FN4035	1524	914	241

DATA SET A

CØDE	LENGTH	WIDTH	DEMAND
FP3704	1120	600	149
FP3704	1120	720	198
FP3704	1320	960	127
FP3704	1400	680	105
FP3704	1400	920	78
FC8644	1000	620	120
FN4045	3048	2000	90
FC9292	980	740	70
FC9292	1040	540	135
MS3929	1320	610	52
FN4003	1524	610	600
FN4003	1829	610	135
FQ3824	2134	1219	70
FQ3824	2438	1219	65
MH4200	1120	680	39
MS3449	1524	508	126
MS3449	1803	660	66
FD8854	2440	1830	250
BLANK	1524	914	150
FD9288	2540	1800	300
MR3933	1020	406	520
BLANK	1219	1219	300
FQ3153	2490	1500	6
FN4008	2438	610	184
FN4008	2438	762	150
FN4008	2438	914	116
FN4008	2438	1219	95
FC9010	1200	600	291
FC9010	1520	440	190
FA7636	2240	1200	225
ME2438	1600	840	70
PS1226	838	559	107
PS1226	1046	711	38
PS1226	1098	457	210
PS1226	1250	457	95
PS1226	1250	533	60
PS1226	1250	553	24
MS3776	2280	2000	21
FD8779	2540	1000	35
FD8779	2540	1200	164
FD8779	2540	1300	35
FD8779	2540	1400	35
FA9113	2660	1200	225
FA3860	3048	2100	54
FD8780	2540	1500	75
FD8780	2540	1800	50
FN3763	3048	1829	20
FN3763	3048	2134	16

DATA SET B

CØDE	LENGTH	WIDTH	DEMAND
FN4006	1219	762	210
FN4006	1524	762	150
FN4006	1524	914	200
BLANK	1219	610	1200
MH2692	1220	1060	26
N3280	1220	610	50
N3280	1220	762	33
N3280	1220	915	68
FD9346	2540	1800	300
FQ4076	1524	1219	20
FC8006	720	720	330
FA3864	3048	1800	66
FD6459	2540	1500	25
FD6459	2540	1600	25
FD6459	2540	1700	25
FD6459	2540	1800	175
FN3867	1930	1372	42
FN3867	2540	1524	56
FN3867	2743	2134	20
FN3583	3048	1700	85
FN3880	1930	864	84
FC8952	1520	440	522
FQ3778	2438	1524	50
FA8779	2240	1300	45
FD9671	2540	1400	140
MQ3775	1920	1200	26
FA8983	1923	1140	264
FA8983	1923	1440	99
FA8983	1930	840	165
MR3971	2540	1800	616
N3252	1220	610	20
FN3875	2489	2134	18
FN4004	1219	610	24
FN4004	1219	762	60
FN4004	1219	1219	180
FN4004	1524	1219	214
FQ3845	1520	440	528
PS0143	1651	610	25
FP3777	1524	914	130

DATA SET C

CØDE	LENGTH	WIDTH	DEMAND
MH2715	960	720	786
MH2715	1560	620	364
MH2715	2060	720	347
FN3818	2438	2134	30
FA9142	2490	1000	25
FA9142	2490	1200	25
FA9142	2490	1600	25
FA9142	2490	1800	25
FN4033	1219	610	1848
FN4033	1219	1219	92
FA9325	2490	1000	50
FA9325	2490	1100	25
FA9325	2490	1200	25
MF4166	840	680	100
N3283	2480	1500	20
MF3473	960	720	45
MF3473	1520	920	100
FN3584	2490	1500	25
FN3584	3048	1500	20
MH2726	1560	620	240
MF2695	900	600	228
PS0107	2490	1800	58
FN3922	1829	1219	85
FD9088	2540	1000	70
FD9088	2540	1100	35
FD9088	2540	1200	175
FD9088	2540	1400	70
N3279	1220	1220	50
MF2694	1360	560	565
FA9430	2490	1700	50
FA9430	3048	1600	50
MH0879	1320	700	100
PS1224	838	635	40
PS1224	1046	711	38
N3276	1220	915	28
MQ3943	2520	1500	400
MH0886	1340	920	60
STØCK	2490	1500	10
ME2443	980	920	100
ME3520	1300	540	100

DATA SET D

CØDE	LENGTH	WIDTH	DEMAND
MF2695	900	600	228
FN3763	3048	1829	20
FN3763	3048	2134	16
FD8780	2540	1500	75
FD8780	2540	1800	50
FN4035	1524	914	241
MS3929	1320	610	52
BLANK	1219	1219	300
FC9292	980	740	70
FC9292	1040	540	135
FC9010	1200	600	291
FC9010	1520	440	190
FN3818	2438	2134	30
FP3777	1524	914	130
FD9123	2420	1700	43
FQ3153	2490	1500	6
FD9152	2440	1830	500
ME2443	980	920	100
FN3853	2490	1700	22
FD9346	2540	1800	300
N3252	1220	610	20
MH0879	1320	700	100
PS211	1200	600	216
MH0875	740	406	344
MH0875	1440	580	1012
N3272	1800	1070	54
PS1225	1219	610	50
PS1225	1250	457	115
MQ3775	1920	1200	26
FN3905	3048	1829	16
FN3880	1930	864	84
FN3583	3048	1700	85

DATA SET E

CØDE	LENGTH	WIDTH	DEMAND
FN4003	1524	610	600
FN4003	1829	610	135
FC8644	1000	620	120
FA3860	3048	2100	54
MS3776	2280	2000	21
MS3325	762	762	1363
FN4033	1219	610	1848
FN4033	1219	1219	92
FD6459	2540	1500	25
FD6459	2540	1600	25
FD6459	2540	1700	25
FD6459	2540	1800	175
FN4045	3048	2000	90
MR3933	1020	406	520
FA3783	2240	1500	108
FA9143	2240	1000	30
FA9143	2240	1200	30
FA9143	2240	1600	30
FA9143	2240	1800	30
STØCK	2490	1800	112
FC8006	720	720	330
STØCK	2490	1500	10
FQ4076	1524	1219	20
FN3584	2490	1500	25
FN3584	3048	1500	20
MH2692	1220	1060	26
FN3913	2438	1880	242
N3279	1220	1220	50
ME3520	1300	540	100
PS0107	2490	1800	58
FQ3845	1520	440	528
FD9671	2540	1400	140
FQ3778	2438	1524	50
FD8854	2440	1830	250
FN4057	2032	914	38

DATA SET F

CØDE	LENGTH	WIDTH	DEMAND
MF3473	960	720	45
MF3473	1520	920	100
MF4166	840	680	100
FA9113	2660	1200	225
FN4002	1829	1219	250
FA9325	2490	1000	50
FA9325	2490	1100	25
FA9325	2490	1200	25
FN3867	1930	1372	42
FN3867	2540	1524	56
FN3867	2743	2134	20
ME2438	1600	840	70
FA9418	2490	1000	140
FA9418	2490	1200	245
FA9418	2490	1300	35
FA9418	2490	1400	35
FA9418	2490	1500	140
FA3864	3048	1800	66
FN4008	2438	610	184
FN4008	2438	762	150
FN4008	2438	914	116
FN4008	2438	1219	95
FN4034	1219	1219	406
FN4034	1829	1219	250
PS0143	1651	610	25
MF3485	860	660	129
MH0886	1340	920	60
N3276	1220	915	28
FN3875	2489	2134	18
BLANK	1219	610	1200
BLANK	1524	914	150
FA9430	2490	1700	50
FA9430	3048	1600	50
FD9088	2540	1000	70
FD9088	2540	1100	35
FD9088	2540	1200	175
FD9088	2540	1400	70
FN3922	1829	1219	85
FA8983	1923	1140	264
FA8983	1923	1440	99
FA8983	1930	840	165
MH4200	1120	680	39
FQ3824	2134	1219	70
FQ3824	2438	1219	65
FC9237	1520	900	95

DATA SET G

CØDE	LENGTH	WIDTH	DEMAND
N3283	2480	1500	20
FP3704	1120	600	149
FP3704	1120	720	198
FP3704	1320	960	127
FP3704	1400	680	105
FP3704	1400	920	78
FD8779	2540	1000	35
FD8779	2540	1200	164
FD8779	2540	1300	35
FD8779	2540	1400	35
MH2726	1560	620	240
PS1226	838	559	107
PS1226	1046	711	38
PS1226	1098	457	210
PS1226	1250	457	95
PS1226	1250	533	60
PS1226	1250	553	24
FC7792	760	600	140
FC7792	1200	900	86
FA7636	2240	1200	225
FA9142	2490	1000	25
FA9142	2490	1200	25
FA9142	2490	1600	25
FA9142	2490	1800	25
MH2715	960	720	786
MH2715	1560	620	364
MH2715	2060	720	347
N3281	1830	610	33
N3281	1830	762	27
N3281	1830	915	23
N3281	1830	1220	37
FD8853	2440	1830	250
FD9288	2540	1800	300
FC8429	1200	600	488
FC8429	1520	440	295
MQ3943	2520	1500	400
PS1224	838	635	40
PS1224	1046	711	38
N3280	1220	610	50
N3280	1220	762	33
N3280	1220	915	68
MR3971	2540	1800	616
MF2694	1360	560	565
FD6458	2540	1500	25
FD6458	2540	1600	25
FD6458	2540	1700	25
FD6458	2540	1800	175
FN4006	1219	762	210
FN4006	1524	762	150
FN4006	1524	914	200
MS3449	1524	508	126
MS3449	1803	660	66
FN4007	1930	1168	125
FN4007	2032	813	170
FN4007	2032	914	110
FA8779	2240	1300	45
FC8952	1520	440	522
FN4004	1219	610	24
FN4004	1219	762	60
FN4004	1219	1219	180
FN4004	1524	1219	214

Appendix B Some notes on programming

The choice of programming systems made for different parts of the present work was more influenced by availability than desirability. The Multi-Pop system (Dunn, 1972) proved easiest to work with, both in terms of the suitability of the language for the work and of the user interface to the system.

Much of the work on trim-loss problems required the expression in programming terms of the spatial relationships between rectangles and, in one case, the way in which these could be changed. Such ideas proved difficult to express in an efficient and easily manipulable form, and in each case an ad hoc method was adopted. The development of a simple vehicle for these ideas would be an extremely useful piece of work.

Had it not been for contact with Algol 68-R (Woodward and Bond, 1972), the present author would not have thought it necessary to make the observation that a compiler should flag illegal constructs, not simply "compile rubbish". The remainder of these remarks will be concerned with run-time events.

As has been remarked in chapter 8 it must be expected that the programs being developed will contain bugs. It is therefore desirable that as much useful information as possible be available to the programmer on program failure:

i) The reporting of run-time errors should be related as closely as possible to the source text. It is of limited value to know that function A called function

B if it could have done this from any one of half a dozen places. Likewise it is unhelpful if the precise point of failure cannot be diagnosed because at the time of program abort the garbage collector was in execution and it is constructed in such a way as to prevent this information being extracted.

ii) A post-mortem facility is extremely desirable as much useful information will be held in complex data structures. The process of extracting the relevant parts of it is much easier if the post-mortem is interactive.

A separate issue is that of information available to the user about the actions of the garbage collector. It should be possible to collect details of execution times including and excluding the time spent in the garbage collector. It should also be possible to determine the store occupancy of a program at various points in its execution. Only with this information is it possible to determine the real cost of running a program and the nature of possible time/space trade-offs.