UNIVERSITY OF LONDON

IMPERIAL COLLEGE OF SCIENCE AND TECHNOLOGY

Department of Management Science

THE QUADRATIC ASSIGNMENT PROBLEM AND PLANT LAYOUT

by

Michael John Gerrard

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

November 1975.

ABSTRACT

This research is concerned with methods for finding optimal solutions to the general quadratic assignment problem. Plant layout is an application of this problem and is used as an example throughout this thesis.

The literature over the last decade has been largely concerned with sub-optimal procedures, but some tree-search algorithms have been proposed. Few empirical results have been reported for these methods and so a computer program was written to rectify this situation; the program has solved significantly larger problems than reported elsewhere.

Several alternative lower bounds have been examined, both theoretically and practically. In particular, the two best-known bounds can be interpreted graphically and thereby shown to be just two elements of a large class of bounds. All the other potentially useful bounds in this class are enumerated.

Integer quadratic programming provides a different approach to the problem and this has been examined in detail. The resultant program cannot guarantee an optimal assignment, but it does find good assignments extremely quickly.

Neither approach has proved very satisfactory for large general quadratic assignment problems, but most practical problems have a specialised structure which may facilitate solution. Various such structures were examined and efficient techniques have been

developed to solve some of them.

The thesis concludes with an empirical comparison of the procedures proposed here with those reported in the literature.

ACKNOWLEDGEMENTS

I wish to thank my supervisor, Dr. N. Christofides, for his advice and guidance throughout this research. I also wish to thank Dr. R. Flavell and Mr. S. Mitra for their help with the IBM 360 computer and its quadratic programming package.

CONTENTS

CHAPTER 1

INTRODUCTION

1.1 ASSIGNMENT WITH QUADRATIC COSTS

The cost function for a wide range of assignment situations is quadratic in nature. This arises when a pair of facilities contributes a cost which depends on the product of the distance between the locations to which these facilities are assigned and the level of interaction of the facilities.

For example, a company may acquire several new premises in different towns and must decide which section of its business to locate in each town. If the sections are all independent of each other then the problem of minimising the operating cost can be formulated as an ordinary assignment problem and easily solved (31,36,54). Generally, however, labour and materials must be transported regularly from one section to another and the cost of this movement depends on the distance between the towns.

Quadratic assignment problems exist on many scales and computer design involves one much-studied situation (18,37,59,60,65). The central processor of a modern computer consists of many modules, each having many terminals which must be connected by wires to particular terminals on other modules. The aim is to locate the modules so as to minimise the total length of wire used.

1.2 PLANT LAYOUT AS A QUADRATIC ASSIGNMENT PROBLEM

As has been indicated, many assignment problems have quadratic costs and chapter 2 discusses some further situations. All these problems have the same mathematical form, and the material in this thesis could be presented in terms of this abstract mathematical form, but it is more convenient to use the nomenclature of a particular practical application. The plant layout problem has been chosen for this purpose.

Plant layout involves a factory with several vacant locations which are to be occupied by several new machines. The locations are considered indivisible so that only one machine may occupy each site. It is convenient to assume that there are the same number of machines as locations; if this is not so, artificial machines with zero costs may be introduced to regularise the problem. The factory may or may not already have existing machines in fixed locations.

Different assignments of machines to locations will result in different levels of efficiency which can be expressed in terms of operating cost. Plant layout involves two types of costs which can be termed linear and quadratic. The best assignment is one which minimises the total operating cost in so far as this depends on the locations of the machines.

Linear costs are those dependent on the locations of individual machines. For example, the amount of artificial lighting required for

a machine may depend on its location. Also the cost of transporting semi-finished goods between a new machine and other existing machines or store-rooms depends on the location of the new machine.

Quadratic costs are due to the interaction of machines and are dependent on the locations of pairs of machines. The cost of transporting materials between two machines is assumed to be proportional to the distance between the machines and so the locations of both machines are needed to determine the cost. As well as materials, employees and information must flow from one machine to another and the cost per unit distance of the total flow between two machines is a measure of the level of interaction for the pair of machines.

The validity of this cost structure has been accepted by most researchers, at least as a very good approximation to reality. Buffa and Vollmann (4) have discussed the underlying assumptions in detail.

1.3 NOTATION

The new machines are labelled alphabetically $A, B, C, \ldots, \alpha, \ldots$ and the locations numerically $1, 2, 3, \ldots, i, \ldots$. The number of both machines and locations is n. An assignment, ρ, is a function mapping each machine, α, to a different location, i.

The linear cost of assigning machine α to location i is denoted by $c_{\alpha i}$. Hence the linear cost of an assignment, ρ, is $\sum_{\alpha} c_{\alpha \rho(\alpha)}$.

The level of interaction of two machines, α and β, is denoted

by $f_{\alpha\beta}$; i.e. $f_{\alpha\beta}$ is the cost per unit distance of the total flow of men, materials and information between machines α and β. By definition $f_{\beta\alpha} = f_{\alpha\beta}$. The distance from location i to location j is denoted by d_{ij}. It is assumed that $d_{ji} = d_{ij}$. The quadratic cost of assigning machine α to location i and machine β to location j is $f_{\alpha\beta} d_{ij}$. Hence the quadratic cost of an assignment, ρ, is

$$\sum_{\alpha} \sum_{\beta>\alpha} f_{\alpha\beta} d_{\rho(\alpha)\rho(\beta)} .$$

The total operating cost of an assignment is denoted by $Z(\rho)$.

Clearly, $Z(\rho) = \sum_{\alpha} c_{\alpha\rho(\alpha)} + \sum_{\alpha} \sum_{\beta>\alpha} f_{\alpha\beta} d_{\rho(\alpha)\rho(\beta)}$.

The plant layout problem is to discover, from the n! possible assignments, one which minimises $Z(\rho)$.

1.4 OUTLINE OF THESIS

The following chapter considers the research on the quadratic assignment problem which has been reported in the literature. The theoretical properties of the problem and its relation to other problems are considered as well as methods of solution. The various heuristics are shown to yield rather poor assignments whilst algorithms which find the optimum can only handle small problems.

Chapter 3 formulates the problem as an integer quadratic program and attempts to modify it so that it can be solved using a standard mathematical programming package.

The next chapter presents a tree-search algorithm and proposes several alternate lower bounds for it. The empirical results for this computer program are superior to others found in the literature.

Chapter 5 is a theoretical study of lower bounds which could be used in a tree-search algorithm.

There has been some research on specific situations for which the distances conform to a restricted structure. Chapter 6 considers several cases for which the flows are restricted.

Chapter 7 makes a detailed computational comparison of the various methods proposed both in the literature and in this thesis.

The final chapter is forced to the conclusion that, in spite of considerable effort by many people, there is still no really satisfactory solution to the general quadratic assignment problem. One of the contributions of this thesis is to partly explain why the problem has caused so much difficulty.

CHAPTER 2

LITERATURE SURVEY

2.1 EARLY HISTORY

For almost as long as factories have been used it has been recognised that the relative locations of machines have a significant effect on efficiency. At first the manager relied on common sense and experience to design a layout, but then factories grew larger and so specialised heuristic techniques were developed. Many textbooks have been written on this topic (17,49,55).

Travel Charting

One analytic tool described in most of these textbooks is the "travel chart" or "from-to chart". This is a matrix in which each row and column corresponds to a machine and the entries give the flow of materials or "travelling" from each machine to every other machine; the diagonal entries are not defined.

Two other charts are often used, one giving the distance between each pair of locations and the other giving the product of flow and distance for each pair of machines for a particular assignment. The sum of all the entries in this product matrix is the total quadratic cost of the assignment. Figure 2-1 shows the three charts for a hypothetical problem with four machines. Also shown is the flow matrix defined in chapter 1 ; each entry is simply the sum of the corresponding entries in the travel chart and its transpose. This symmetric matrix can be used to calculate a triangular quadratic cost matrix which is equivalent to the product matrix.

	A	B	C	D
A	-	8	4	2
B	0	-	0	7
C	4	7	-	0
D	1	5	0	-

Travel Chart

	1	2	3	4
1	-	3	4	5
2	3	-	4	6
3	4	4	-	5
4	5	6	5	-

Distance Chart
or Distance Matrix (d_{ij})

	A	B	C	D
A	-	24	16	10
B	0	-	0	42
C	16	28	-	0
D	5	30	0	-

Product Chart for the assignment (A1 B2 C3 D4). Quadratic Cost = 171.

	A	B	C	D
A	-	8	8	3
B	8	-	7	12
C	8	7	-	0
D	3	12	0	-

Flow Matrix $(f_{\alpha\beta})$

	A	B	C
B	24		
C	32	28	
D	15	72	0

Triangular Quadratic Cost Matrix for (A1 B2 C3 D4). Quadratic Cost = 171.

Figure 2-1.

Travel charts and matrices for a 4-machine problem.

The travel charting method begins with the travel and distance charts and then an assignment is proposed. This is evaluated using a product chart. If the quadratic cost seems too large, a new assignment is proposed and evaluated; the process is repeated until a satisfactory layout is found. Inspection of the product chart usually suggests possible improvements to the layout - in figure 2-1, for example, it seems that either B or D should be moved. Moore (49) notes that "the travel-chart technique is highly dependent on the ingenuity of the layout man, since it utilises a trial-and-error technique".

Koopmans and Beckmann's Formulation

The importance of travel charting is not in producing good layouts, but in quantifying the efficiency of layouts. Its underlying assumption is that the cost of moving materials between two machines is proportional to the distance between them. This is the only assumption about costs which is needed to justify the use of travel charts.

Koopmans and Beckmann (34) extended these ideas in two senses. Firstly, they recognised that the same cost assumption was applicable on a larger scale when assigning plants or other economic activities to towns or geographic regions. Secondly, they were able to formulate the quadratic assignment problem as a mathematical program modelled on the ordinary (linear) assignment problem.

Their program defines a matrix of variables, $X = (x_{\alpha i})$ such that $x_{\alpha i}$ has the value 1 if machine α is assigned to location i

and has the value 0 otherwise. The quadratic cost of an assignment can then be expressed as

$$\sum_{\alpha}\sum_{\beta}\sum_{i}\sum_{j} f_{\alpha\beta} d_{ij} x_{\alpha i} x_{\beta j} .$$

(In their formulation $f_{\alpha\beta}$ was only the flow from α to β, not from β to α as well). This function is quadratic in X and that accounts for the name "quadratic assignment problem".

At that time there was no known method for solving quadratic programs but Koopmans and Beckmann were able to transform it into a linear program. Unfortunately this was too large to be practical; and anyway, their research was not specifically concerned with solving the problem. But this thesis is concerned with solution and so chapter 3 considers the quadratic program in detail.

2.2 APPLICATIONS AND RELATED PROBLEMS

There are many applications for the quadratic assignment problem and some which have appeared in the literature are described below in descending order of scale. Some problems do not involve assignment to locations but can still be put in the mathematical form of the quadratic assignment problem. Some generalised and some restricted problems are also mentioned.

At an international level a multi-national corporation may decide, for partly political reasons, to manufacture different components in different countries; at least one computer manufacturer does this.

The purely economic decision of which component is assigned to each country is clearly a quadratic assignment problem if the various factories have any significant interaction.

The same situation obviously arises within a country or region, but there can be a different emphasis in centrally controlled and mixed economies. All economic activity within a country interacts and could, in theory, be modelled by a huge quadratic assignment formulation. If the government has direct control over industry it should try to implement a solution to the quadratic assignment problem. If it has only indirect control, such as by fixing rents or taxes in different areas, then it should try to adjust these pricing mechanisms so that the market forces of private enterprise automatically select an optimal assignment. Koopmans and Beckmann studied these price mechanisms, but recently Artle and Varaiya (2) have shown that, in general, there are no prices which will ensure an optimal assignment.

Plant layout is one problem at the scale of a building or group of buildings. Another is the assignment of employees to rooms in an office building; the aim is to minimise the walking between offices (42). Two algorithms have been proposed for the one-dimensional problem of assigning rooms along a corridor, one being a tree-search formulation (64) and the other dynamic programming (40) - neither could deal with as many as 20 rooms.

The layout problem for a reference library is very similar

to plant layout. Very few readers walk from books about computers to medieval literary criticism and so these do not need to be near each other, but computing should be near mathematics for example. It would also be convenient, however, to have computing near accounting, and accounting near law, and law near sociology, etc. Compromise is necessary and the best compromise is an optimal solution to the quadratic assignment problem which minimises total walking.

Computing design has already been mentioned and there are numerous publications on this topic (18,24,37,59,60,65). Computer operation also provides an example (52). A computer tape or disc may contain many records which are accessed in different orders by various users. The "distance" between two locations on a tape or disc is the time needed to move the tape or reading head from one location (record) to the other; the "flow" between two records is the frequency with which they are accessed consecutively.

The travelling salesman problem can be forced into quadratic assignment form by defining $f_{\alpha\beta}$ to be zero except when $\beta = \alpha + 1$ - then it takes the value 1. Lawler (39) has extended this to a multi-salesman problem in which the salesmen must also pay to communicate with each other. Maxwell (47) has formulated a machine sequencing problem in a similar way, although it requires a more complex flow matrix.

Bowman, Pierce and Ramsey (3) have formulated an economic

problem of input-output analysis in quadratic assignment form. Their formulation is not quite compatible with the definition used in this thesis because their distance matrix is not symmetric - in fact, it is an upper triangular matrix of 1's. They convert the problem to a linear program with which they are able to solve moderately large problems.

Other Related Problems

Lawler (39) has defined a generalised quadratic assignment problem for which the objective function is

$$Z(\rho) = \sum_{\alpha} \sum_{\beta} a_{\alpha\beta} \rho(\alpha)\rho(\beta)$$

where $a_{\alpha\beta ij}$ is the cost of interaction of machines α and β when they are located at i and j respectively. Note that even a 10-machine problem requires almost 10,000 items of data and so all practical situations must be special cases in some sense. As one special case he proposes the multi-commodity problem for which $a_{\alpha\beta ij} = \sum_{k} f^{(k)}_{\alpha\beta} d^{(k)}_{ij}$. This assumes that different commodities (distinguished by superscript, k) must use different routes of different lengths between pairs of locations.

Several papers have been published on the related bottleneck or minimax problem (5,6,7,14,44). This aims to minimise $Z(\rho) = \max (f_{\alpha\beta} d_{\rho(\alpha)\rho(\beta)})$ and has proved slightly easier than the quadratic assignment problem (5,7), especially when distances are rectilinear (14).

Another related problem which has received much attention allows the new machines to be placed anywhere in 2 or 3 dimensions (9,12,45,46,51,69). Again the rectilinear problem is easier (9,12,51,69).

2.3 MATHEMATICAL PROPERTIES

The first step in solving many problems is to transform them into other problems. This section discusses transformations which leave optimal assignments unchanged.

As defined in chapter 1, the cost of an assignment, ρ, is $Z(\rho) = \sum_{\alpha} c_{\alpha\rho(\alpha)} + \sum_{\alpha}\sum_{\beta>\alpha} f_{\alpha\beta} d_{\rho(\alpha)\rho(\beta)}$. The first term of this expression is the same as for the linear assignment problem for which a well-known transformation subtracts a constant from each entry in a particular row of the cost matrix. This is still valid: i.e. if $c_{\alpha i}$ is reduced by γ for a particular machine, α, and every location, i, the total cost of every assignment is thereby reduced by γ and so any optimal assignment for the transformed problem is also optimal for the original problem.

Transformation of the flow and distance matrices is more complex. If all the flows from a particular machine are reduced then that machine becomes less critical to the plant and so is likely to be put in an isolated corner - hence this transformation affects the optimality of assignments. However, if flows between all pairs of machines are reduced by the same amount, the costs of all assignments

also change by a fixed amount.

Burkard (5,6) has shown that flows from individual machines can be reduced if the linear costs are simultaneously increased. Specifically, if $f_{\alpha\beta}$ and $f_{\beta\alpha}$ are decreased by γ for a particular machine, α, and every other machine, β, and $c_{\alpha i}$ is increased by $\gamma\sum_j d_{ij}$ for every location, i, then the costs of all assignments remain unchanged.

An analogous transformation exists for the distance matrix. This means that at least one zero entry can be forced into each row and column of the flow and distance matrices. The resulting shift in emphasis from the quadratic to the linear term was expected to assist the search for optimal assignments, but Burkard's success was largely restricted to the bottleneck problem defined in the previous section.

2.4 CONSTRUCTIVE HEURISTICS

Procedures which calculate good assignments may be classified as "algorithms" if they guarantee to find an optimal assignment or as "heuristics" if they only aim to find a good sub-optimal assignment. Nugent, Vollmann and Ruml (57) further classify heuristics as being either "constructive" or "improvement". A constructive heuristic considers one machine at a time and selects the location to which it will be assigned whereas an improvement heuristic attempts to modify a given layout to decrease its cost.

Gilmore (21) proposed the first effective constructive heuristics and most other constructive methods are based on his concepts. Both his techniques can be extended to tree-search algorithms which eventually find an optimal assignment (see section 2.7).

Gilmore's basic idea is to approximate the quadratic costs by linear costs. Consider the quadratic costs incurred by assigning machine α to location i. There are n-1 flows, $f_{\alpha\beta}$, which must be assigned to n-1 distances, d_{ij}, in some order, and the cost of this is the sum of the pairwise products of the corresponding flows and distances. This cost is minimised by assigning the largest flow to the smallest distances, the next-to-largest flow to the next-to-smallest distance and so on. This gives a lower bound on the quadratic costs of assigning α to i and, when added to the known linear cost, $c_{\alpha i}$, it gives an indication of the actual cost of assigning α to i.

These approximate costs are calculated for all n^2 possible assignments and then the most critical assignment is selected according to a heuristic rule. Gilmore's two procedures differ only in their rules for this choice. His "n^4" heuristic finds the minimal approximate cost for each machine and location and then selects the machine-location pair which yields the largest minimum. The "n^5" heuristic considers the nxn matrix of approximate costs as data for a linear assignment problem and selects the largest cost which appears in that problem's solution. The names "n^4" and "n^5" refer to the order of computational

effort required.

Now that one machine has been assigned to a location the size of the problem has effectively been reduced from n to n-1 machines. Approximate costs are recalculated for the reduced problem and another machine assigned. This continues until all the machines have been assigned.

Other Constructive Methods

Hillier and Connors (30) used the same process to calculate the matrix of approximate costs, but a "difference" or "penalty" concept for selecting the machine-location pair. For each row and column of the matrix they calculated the difference between the smallest and next-to-smallest entries, and then selected the smallest entry in the row or column which had the largest difference. This new rule produced slightly better solutions than either of Gilmore's rules when applied to a computer design problem with 34 modules proposed by Steinberg (65).

Heider (29) proposed two alternative methods of calculating approximate costs and his selection rule was to choose the machine-location pair giving the smallest cost. His more successful method calculated the mean value of all complete assignments which included the specified machine-location pair. An algebraic expression for this mean was derived by Graves and Whinston (22) whose own method, although a heuristic, is best considered as an incomplete tree-search algorithm and so is discussed in section 2.7. Heider claims that his

heuristic is marginally better than the earlier construction methods.

Recently Neghabat (56) has proposed an interesting constructive heuristic for the special case of rectilinear distances. He partially decomposes the problem into 2 one-dimensional problems. The solution of either of these new problems defines an assignment, but the two solutions may not be compatible. This difficulty is resolved by alternately solving one-dimensional problems in each dimension until convergence to a good assignment is achieved. The quality of his solutions is very slightly worse than others, but the computation is extremely fast.

2.5 IMPROVEMENT HEURISTICS

Just as all constructive heuristics (excluding Neghabat's specialised method) are based on Gilmore's original concept, so all the improvement heuristics are based on the concept of exchanging two machines if so doing reduces the total cost of the layout. The various heuristics differ only in the order in which they consider machines and in the basic assumptions made about the layout.

Armour and Buffa seem to have been the first to formalise this idea and program it for a computer (1,4). They assume that different machines may be of different sizes and so only exchanges between adjacent machines or machines of the same size can be considered. They calculate the cost-improvement (positive or negative) resulting from each feasible exchange and, if any improvements are positive,

the exchange giving the largest improvement is made. The process is repeated until no further improvement is possible. This heuristic has become known as CRAFT - Computerised Relative Allocation of Facilities Technique.

Hillier (29) independently programmed a restricted form of CRAFT which assumes equal-sized machines to be located on the lattice points of a rectangular grid. Since he only considers exchanges between machines which are orthogonally or diagonally adjacent, the resulting assignments are not as good as CRAFT's although much less computation is involved.

An improved version (30) allows some non-adjacent exchanges. It also ranks the desirability of moving each machine in each direction so that the most propitious exchanges may be considered first. This heuristic is very much faster than CRAFT for large problems and its layouts appear to be only marginally worse.

The above procedures are deterministic in that each produces a specific final assignment from a given initial assignment. Nugent et al. (57) proposed a stochastic procedure which they called biased sampling. This differs from CRAFT in that all exchanges giving a positive improvement are considered and the probability of a particular exchange being made is a function of its improvement.

Nugent et al. also programmed the earlier heuristics and compared their performance on eight problems with from 5 to 30 machines. These have since become standard test problems and are

reproduced in appendix A. They claim that biased sampling yields slightly better assignments than CRAFT, although at the expense of vastly more computation.

Garside and Nicholson (18), Edwards, Gillet and Hale (13), Khalil (33), Munita (53) and Hitchings (32) have all tested other minor modifications of the CRAFT principle and met with minor success. Munita notes that CRAFT is an application of the "r-opt" concept developed by Lin for the travelling salesman problem (43) - here r = 2.

2.6 COMPUTER-AIDED HUMAN TECHNIQUE

Scriabin and Vergin (63) have recently noted that several authors (24,57) have compared the relative effectiveness of various computerised heuristics, but that the traditional methods were being ignored. Therefore they wrote a simple, interactive APL program which calculates the cost of a proposed layout and generally facilitates the travel charting method described in section 2.1.

Seventy four students used this program to solve nine layout problems ranging in size from 5 to 20 machines. Each student sat at a remote typewriter terminal and entered a suggested layout for a problem. The computer then calculated the cost of the layout and, on request, also printed travel charts of the type shown earlier in figure 2-1. Using these charts the student could suggest new layouts until he was satisfied with the result.

Each problem was given to up to twenty students who independently tried to find good layouts. Their results were compared with CRAFT and Hillier's original improvement heuristic as tested by Grover (23).

For every problem the best "manual" layout was at least as good as the best "computer" layout, and for the largest problem the manual layout was 5.8% cheaper. Also, the median manual layout was significantly better than the median computer layout for most problems.

Perhaps the most important conclusion to be drawn from this experiment is that heuristics do not produce particularly good layouts, especially for larger problems. For the 20-machine problem the best heuristic layout is at least 5.8% worse than the optimum and may be even more inferior.

2.7 TREE-SEARCH ALGORITHMS

Apart from methods for specialised problems which are discussed in the next section, all the algorithms proposed for the quadratic assignment problem have been of the implicit enumeration type known as "branch-and-bound" or "tree-search". Pierce and Crowston (58) have published an excellent review of the various algorithms and there is a detailed analysis in chapter 4 of this thesis; hence only a brief summary is presented here.

In 1962 Gilmore (21) and Lawler (39) simultaneously described equivalent tree-search algorithms although neither reported any computational experience. Gilmore's constructive heuristics were based on his algorithm.

An important part of any tree-search algorithm is the calculation of a lower bound on the costs of all assignments. Gilmore and Lawler both use the matrix of "approximate costs" described in section 2.4. Since these costs are, in fact, lower bounds on the costs of assigning particular machines to particular locations, the solution to the linear assignment problem defined by this matrix is a lower bound on the cost of all layouts.

Gilmore also suggests an alternative which is weaker but much easier to calculate. Since each of the $\frac{1}{2}n(n-1)$ flows must be paired with one of the $\frac{1}{2}n(n-1)$ distances, the quadratic cost is not less than the cost of pairing the smallest flow with the largest distance, the next-to-smallest flow with the next-to-smallest distance and so on. This process gives a lower bound which is readily calculated but can be shown to be lower than the previous bound.

Figure 2-2 indicates how lower bounds are used to develop a tree which partitions all the assignments into disjoint subsets. First a bound is calculated for the original problem (389) and then four 3-machine sub-problems are defined by assuming location 1 is occupied by each of the four machines in turn. Lower bounds are calculated for each sub-problem and the lowest bound (392) determines the next

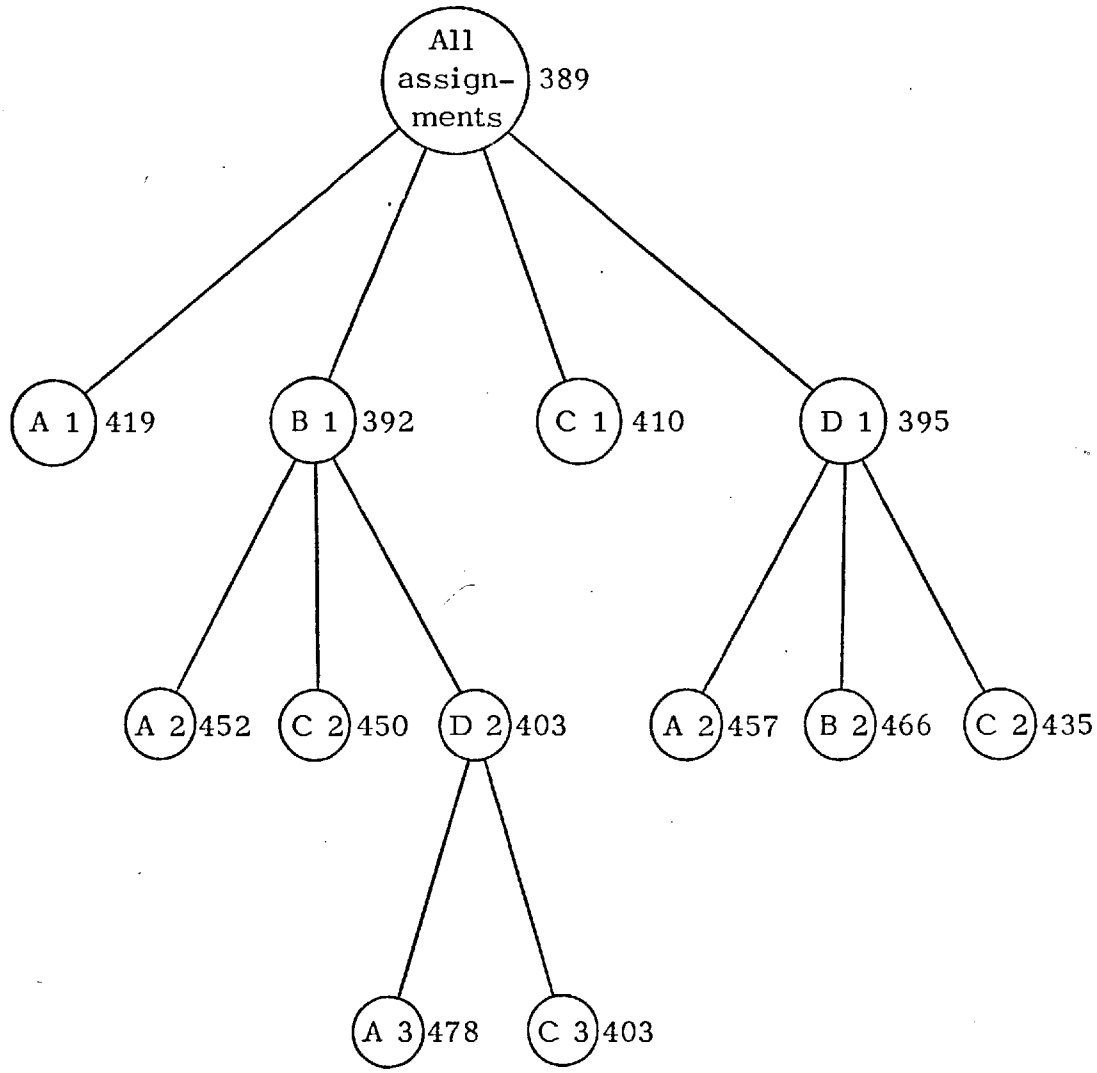


Figure 2-2.

Tree for a 4-machine problem using Gilmore's weak bound.

(Copied from Pierce and Crowston (58)).

branching (B1). Eventually the optimal assignment (A4,B1,C3,D2) is found and its cost, 403, is less than all the other terminal bounds.

Gavett and Plyter (20) and Land (38) chose a different branching strategy. Rather than assigning a machine, α, to a location, i, they assigned a pair of machines, (α,β), to a pair of locations, (i,j), without specifying which machine was assigned to which location. This can be viewed as assigning $f_{\alpha\beta}$ to d_{ij}.

Gilmore's weaker bound is well-suited to this branching strategy and Gavett and Plyter programmed the algorithm for an IBM 7074 computer. The results were disappointing, needing 42 minutes to solve an 8-machine problem.

Graves and Whinston (22) have programmed a heuristic based on the Lawler-Gilmore algorithm. They derived explicit algebraic expressions for the mean and variance of the costs of all n! assignments and were able to use these at each node of the tree to estimate the probability that an optimal solution was included in that node. If this probability is very small they discard the node, even if its bound is less than the cost of the best known assignment.

They have compared their method to several other heuristics and found it produced very good assignments quite quickly.

2.8 ONE-DIMENSIONAL PROBLEMS

For several problems, such as ordering rooms along a corridor or records on a computer tape, the locations lie in a straight line. Simmons (64) and Lawler (40) have both studied this situation.

Simmons programmed a tree-search algorithm for a closely related problem which is equivalent to the quadratic assignment problem if the distances between adjacent locations are all equal. His program could only handle rather small problems with about 10 machines.

Lawler defined a dynamic programming formulation which would be effective for up to about 16 or 18 machines. It is very similar to Held and Karp's formulation for the travelling salesman problem (26). He also considers several one-dimensional problems with restricted flow patterns such as networks and trees. Some of these cases are easily solved, but some are not.

2.9 CONCLUSIONS

This literature has shown that a great amount of effort by many people has been applied to developing techniques to "solve" the quadratic assignment problem. Francis and Goldstein (16) give a bibliography of 226 papers whilst Moore (50) lists 25 independently developed heuristic programs. And yet the result of all this trouble does not appear impressive.

Ignoring specialised techniques, there are just two basic tree-search algorithms and two heuristics. Nugent et al. (57) have concluded that constructive heuristics are inferior to improvement heuristics, but Scriabin and Vergin (63) have demonstrated that students with a desk calculator are significantly superior to improvement heuristics for large problems.

Computational experience with tree-search algorithms is very limited. The Gavett and Plyter program could not handle more than eight machines and the Lawler-Gilmore algorithm does not appear to have been programmed. Gilmore has estimated that his method could not deal with more than fifteen machines.

Because of the apparent lack of computational experience with algorithms, this thesis investigates these methods which yield optimal assignments. Some specialised problems which can be solved more easily are also studied.

CHAPTER 3

QUADRATIC PROGRAMMING APPROACH

3.1 THE LINEAR ASSIGNMENT PROBLEM

The quadratic assignment problem can be considered as a generalisation of the linear assignment problem. The constraints are identical but the quadratic cost function is more complex. This chapter tries to extend a solution technique from the linear case to the quadratic.

For the linear situation the cost of an assignment, ρ, is

$$Z(\rho) = \sum_{\alpha} c_{\alpha\rho(\alpha)} .$$

This can be formulated as a 0-1 linear program by interpreting the elements of an nxn matrix, X, in the following way:

$$x_{\alpha i} = \begin{cases} 1 & \text{if machine } \alpha \text{ is assigned to location } i, \\ 0 & \text{otherwise.} \end{cases} \qquad 3\text{-}1$$

$$\text{Then } Z(X) = \sum_{\alpha} \sum_{i} c_{\alpha i} x_{\alpha i} . \qquad 3\text{-}2$$

Necessary and sufficient conditions for matrix X to represent an assignment are, in addition to 3-1 :

$$\sum_{i} x_{\alpha i} = 1 \quad \text{for all machines } \alpha, \text{ and}$$
$$\sum_{\alpha} x_{\alpha i} = 1 \quad \text{for all locations } i. \qquad 3\text{-}3$$

If the 0-1 constraint (3-1) is replaced by the weaker inequality

$$0 \leqslant x_{\alpha i} \leqslant 1 \qquad 3\text{-}4$$

then the linear program comprising 3-2, 3-3 and 3-4 can be solved readily using standard computer programs to minimise Z(X). It can be shown (31) that the coefficients of the constraints form a unimodular matrix: hence an optimal basic solution automatically has integer variables and so constraint 3-1 is obeyed implicitly. Thus the linear

assignment problem can be solved effectively using linear programming, although this is not the most efficient technique.

3.2 BASIC QUADRATIC PROGRAMMING FORMULATION

For the quadratic assignment problem the cost of an assignment is $Z(\rho) = \sum_{\alpha} c_{\alpha\rho(\alpha)} + \sum_{\alpha}\sum_{\beta>\alpha} f_{\alpha\beta} d_{\rho(\alpha)\rho(\beta)}$.

Using the 0-1 variables defined in 3-1 this cost can be written as

$$Z(X) = \sum_{\alpha}\sum_{i} c_{\alpha i} x_{\alpha i} + \sum_{\alpha}\sum_{\beta>\alpha}\sum_{i}\sum_{j\neq i} f_{\alpha\beta} d_{ij} x_{\alpha i} x_{\beta j} . \qquad 3\text{-}5$$

Minimising this expression subject to constraints 3-1 and 3-3 would solve the quadratic assignment problem. Koopmans and Beckmann (34) derived an equivalent expression but were unable to optimise it.

There are no general techniques available for solving 0-1 quadratic programs, but there are computer packages available for continuous quadratic programs. It is possible to solve quite large problems of the form :

$$\begin{aligned} &\underset{\sim}{y} \geqslant 0 \\ &\min Z(\underset{\sim}{y}) = \underset{\sim}{r}^{T}\underset{\sim}{y} + \underset{\sim}{y}^{T} Q \underset{\sim}{y} \qquad 3\text{-}6 \\ &\text{subject to } A\underset{\sim}{y} = \underset{\sim}{s} . \end{aligned}$$

where $\underset{\sim}{y}$ is a k-component vector variable,

$\underset{\sim}{r}$ is a k-component vector of linear cost coefficients,

Q is a kxk positive semi-definite cost matrix,

A is an mxk matrix of constraint coefficients, and

$\underset{\sim}{s}$ is an m-component vector of constraint values.

The restriction that Q be positive semi-definite means that $\underset{\sim}{y}^T Q \underset{\sim}{y}$ must be greater than or equal to zero for all real values of $\underset{\sim}{y}$. This ensures that $Z(\underset{\sim}{y})$ is a convex function and hence that any local minimum is, in fact, a global minimum. If Q is not positive semi-definite then the quadratic program may have many local minima with values greater than the true minimum (66). There is a more complete analysis of this difficulty in section 3.6 .

The previous section showed how a linear assignment problem can be solved as a linear program by relaxing the 0-1 constraint (3-1) to a simple inequality (3-4). The same relaxation will now be considered for the quadratic assignment problem. Note first that the constraints 3-3 and $x_{\propto i} \geqslant 0$ automatically force $x_{\propto i} \leqslant 1$ and so this part of constraint 3-4 can conveniently be omitted. This leads to the following basic quadratic programming formulation which conforms to the standard format of 3-6 :

$$
\begin{aligned}
& x_{\propto i} \geqslant 0 \quad \text{for all machines } \propto \text{ and locations } i, \\
& \min Z(X) = \sum_{\propto} \sum_{i} c_{\propto i}\, x_{\propto i} + \sum_{\propto} \sum_{\beta \neq \propto} \sum_{i} \sum_{j \neq i} f_{\propto \beta}\, d_{ij}\, x_{\propto i}\, x_{\beta j} \\
& \sum_{i} x_{\propto i} = 1 \quad \text{for all machines } \propto \qquad \text{3-7} \\
& \sum_{\propto} x_{\propto i} = 1 \quad \text{for all locations } i.
\end{aligned}
$$

In terms of the standard format, k takes the value n^2, m is 2n, $\underset{\sim}{r}$ contains the linear costs $c_{\propto i}$, and $\underset{\sim}{s}$ contains only 1's. The elements of A are 0's and 1's, and the elements of Q are the products of

appropriate flows and distances. Figure 3-1 shows how the 4-machine problem considered by Gavett and Plyter (20) is translated into the quadratic programming format.

3.3 RESULTS FOR THE BASIC FORMULATION

A matrix generator was programmed to produce the standard format for QPS, a quadratic programming package for the IBM 360 series computers. This package was then used to solve the quadratic program 3-7 for several problems involving from 4 to 8 machines.

Figure 3-2 shows that the variables do not automatically take 0-1 values as in the linear case; this should not be surprising. There does, however, appear to be a strong correlation between the fractional solutions and the 0-1 optima which are being sought.

In fact, for NVR8, the 8-machine problem given by Nugent, Vollmann and Ruml (57), a simple rounding procedure provides an optimal assignment; i.e. if the value 1 is given to those variables greater than 0.5 and 0 to those less than 0.5, the resultant solution happens to be both feasible and optimal.

Unfortunately GP4, Gavett and Plyter's 4-machine problem, is less promising. There are only 3 variables greater than 0.5, namely x_{A4}, x_{B3} and x_{D2}. If these are set equal to 1 and the solution made feasible by also increasing x_{C1} from 0.30 to 1, then the cost of the resultant assignment is 419. But if machines B and C are

$(f_{\alpha\beta}) =$

	A	B	C
B	28		
C	25	15	
D	13	4	23

$(d_{ij}) =$

	1	2	3	4
1	–	6	7	2
2	6	–	5	6
3	7	5	–	1
4	2	6	1	–

$c_{\alpha i} = 0$

$$\underset{\sim}{x} \geqslant 0$$

$$Z(\underset{\sim}{x}) = \underset{\sim}{r}^T \underset{\sim}{x} + \underset{\sim}{x}^T Q \underset{\sim}{x}$$

$$A\underset{\sim}{x} = \underset{\sim}{b}$$

$$\underset{\sim}{x}^T = (x_{A1}, x_{A2}, x_{A3}, x_{A4}, x_{B1}, x_{B2}, \ldots, x_{D3}, x_{D4})$$

$$\underset{\sim}{r}^T = (0, 0, 0, 0, 0, 0, \ldots, 0, 0)$$

$$\underset{\sim}{b}^T = (1, 1, 1, 1, 1, 1, 1, 1)$$

$Q =$

		A 1	A 2	A 3	A 4	B 1	B 2	B 3	B 4	C 1	C 2	C 3	C 4	D 1	D 2	D 3	D 4
A	1																
	2																
	3																
	4																
B	1	0	168	196	56												
	2	168	0	140	168												
	3	196	140	0	28												
	4	56	128	28	0												
C	1	0	150	175	50	0	90	105	30								
	2	150	0	125	150	90	0	75	90								
	3	175	125	0	25	105	75	0	15								
	4	50	150	25	0	30	90	15	0								
D	1	0	78	91	26	0	24	28	8	0	138	161	46				
	2	78	0	65	78	24	0	20	24	138	0	115	138				
	3	91	65	0	13	28	20	0	4	161	115	0	23				
	4	26	78	13	0	8	24	4	0	46	138	23	0				

$A =$

		A 1	A 2	A 3	A 4	B 1	B 2	B 3	B 4	C 1	C 2	C 3	C 4	D 1	D 2	D 3	D 4
A		1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
B		0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
C		0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0
D		0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
	1	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0
	2	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0
	3	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0
	4	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1

Figure 3-1.

Basic Quadratic Program for Gavett and Plyter's 4-machine problem.

GP4

Min (BQP) = 341.47 .

Min (QAP) = 403.

	1	2	3	4
A	0	.14	0	.86*
B	.36*	0	.64	0
C	.30	.28	.28*	.14
D	.34	.58*	.08	0

NVR8

	1	2	3	4	5	6	7	8
A	0	0	0	0	.20	.80*	0	0
B	.19	0	0	0	.61*	.20	0	0
C	.81*	0	0	0	.19	0	0	0
D	0	.20	.04	0	0	0	.77*	0
E	0	0	0	.30	0	0	0	.70*
F	0	0	0	.70*	0	0	0	.30
G	0	.20	.80*	0	0	0	0	0
H	0	.60*	.16	0	0	0	.23	0

Min (BQP) = 101.57 . Min (QAP) = 107.

Figure 3-2.

Solutions to the basic quadratic program (BQP) for QP4 and NVR8.

Solutions to the quadratic assignment problem (QAP) are shown by "*".

interchanged the cost of the new assignment is only 403 - in fact this is the optimum. The fractional solution in figure 3-2 gives no hint that this could be so.

A more sophisticated rounding procedure could be based on solving the linear assignment problem defined by the solution matrix, choosing the assignment which maximises the sum of the $x_{\alpha i}$ variables. Each row and column of the solution matrix could first be weighted according to the importance of the appropriate machine or location: i.e. the total of flows or distances from that machine or location. But in the case of GP4 any reasonable weighting scheme would still produce the assignment with cost 419.

These two examples show the limitations of the basic quadratic program. Because the solution can be fractional it may not directly specify an assignment, but it can give a useful indication of a good, if not optimal, assignment. The cost of the fractional solution may also be a lower bound on the cost of the optimal assignment which could be used in a tree-search algorithm.

3.4 BIASED VARIABLES

This section shows how the basic quadratic programming objective function can be modified so that 0-1 results are more likely. The costs of all 0-1 solutions must remain unchanged while the costs of fractional solutions are increased. This concept has been suggested

for other combinatorial problems by Taha (67) and Raghavachari (61); Christofides and Mitra (11) used it with some success on a class of 0-1 problems.

It is important for computational convenience that the bias on the objective function should have either a linear or quadratic character. Consider the bias function $b_{\alpha i} x_{\alpha i}(1-x_{\alpha i})$ where $b_{\alpha i}$ is some non-negative parameter called a bias coefficient; the form of this function is shown in figure 3-3. It is positive for $0 \leqslant x_{\alpha i} \leqslant 1$ and zero for $x_{\alpha i} = 0$ and 1. It is symmetric about $x_{\alpha i} = \frac{1}{2}$ and takes its maximum value of $\frac{1}{4} b_{\alpha i}$ when $x_{\alpha i} = \frac{1}{2}$.

Suppose that a solution has been found for the basic quadratic program in which $x_{\alpha i}$ is fractional for some machine α and some location i. If the bias function with positive $b_{\alpha i}$ is added to the objective function then the value of that particular solution is increased by $b_{\alpha i} x_{\alpha i}(1-x_{\alpha i})$ which is a positive amount. By suitable choice of the bias coefficient the cost may be made arbitrarily large. The cost of all solutions for which $x_{\alpha i}$ is fractional will be increased by adding this bias, but the cost of solutions for which $x_{\alpha i}$ is 0 or 1 will remain unchanged.

The bias functions can be added simultaneously for all variables so that the biased objective function is:

$$Z(X) = \sum_{\alpha} \sum_{i} (c_{\alpha i} + b_{\alpha i}) x_{\alpha i} + \sum_{\alpha} \sum_{i} (-b_{\alpha i} x_{\alpha i}^2 + \sum_{\beta > \alpha} \sum_{j \neq i} f_{\alpha\beta} d_{ij} x_{\alpha i} x_{\beta j})$$

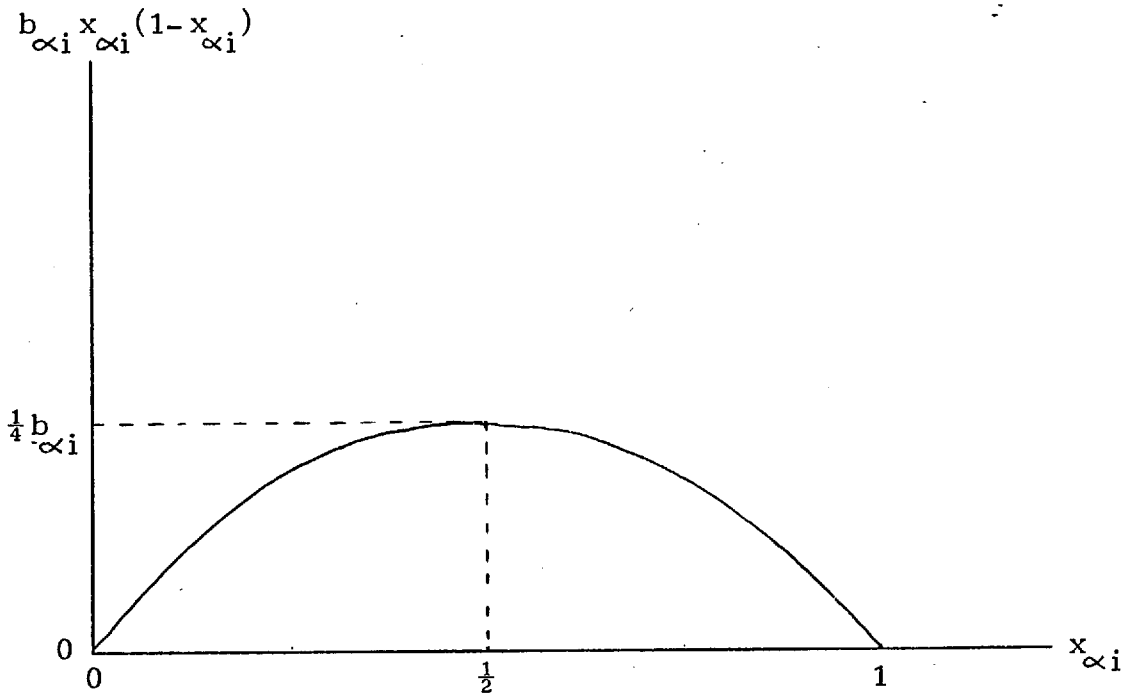


Figure 3-3.

The quadratic bias function.

Choosing sufficiently large values for the bias coefficients can ensure that the minimum value of this expression, subject to constraints 3-3 and 3-4, occurs when all variables are 0 or 1.

The 2-Machine Problem

It is useful to examine in detail the effect of bias on the rather trivial 2-machine quadratic assignment problem. The constraint set 3-3 ensures that $x_{A1} = x_{B2}$ and $x_{A2} = x_{B1} = 1 - x_{A1}$. Hence for all α and i, $x_{\alpha i}(1-x_{\alpha i}) = x_{A1}(1-x_{A1})$. This allows the objective function to be simplified as follows:

$$Z(X) = \sum_{\alpha}\sum_{i} c_{\alpha i} x_{\alpha i} + \sum_{\alpha}\sum_{i}\left(b_{\alpha i} x_{\alpha i}(1-x_{\alpha i}) + \sum_{\beta>\alpha}\sum_{j\neq i} f_{\alpha\beta} d_{ij} x_{\alpha i} x_{\beta j}\right)$$

$$= (c_{A1} - c_{A2} - c_{B1} + c_{B2}) x_{A1} + c_{A2} + c_{B1} + (b_{A1} + b_{A2} + b_{B1} + b_{B2}) x_{A1}(1-x_{A1})$$
$$+ f_{AB} d_{12} x_{A1}^2 + f_{AB} d_{21} (1-x_{A1})^2$$

$$= (c' - c'') x_{A1} + c'' + b x_{A1}(1-x_{A1}) + f_{AB} d_{12}(x_{A1}^2 + (1-x_{A1})^2)$$

where $c' = c_{A1} + c_{B2}$, $c'' = c_{A2} + c_{B1}$

and $b = b_{A1} + b_{A2} + b_{B1} + b_{B2}$.

Further rearrangement yields:

$$Z(X) = (2 f_{AB} d_{12} - b) x_{A1}(x_{A1} - 1) + (c' - c'') x_{A1} + c'' + f_{AB} d_{12}$$

$$= (2 f_{AB} d_{12} - b)\left\{x_{A1} - \tfrac{1}{2} + \frac{c' - c''}{2(2 f_{AB} d_{12} - b)}\right\}^2 +$$

$$+ c'' + f_{AB} d_{12} - \frac{(2 f_{AB} d_{12} - b - c' + c'')^2}{4(2 f_{AB} d_{12} - b)} .$$

This function of x_{A1} is a parabola, symmetric about $x_{A1} = \frac{1}{2} - \frac{c' - c''}{2(2f_{AB}d_{12} - b)}$. It will be convex (concave) if b is less than (greater than) $2f_{AB}d_{12}$. The turning point will lie between 0 and 1 if $b \geqslant 2f_{AB}d_{12} + |c' - c''|$ or $b \leqslant 2f_{AB}d_{12} - |c' - c''|$. These characteristics are summarised in figure 3-4.

It is clear that a quadratic programming technique will always yield the optimal assignment (in this example $x_{A1} = 0$) if the total bias coefficient, b, lies between $2f_{AB}d_{12} - |c' - c''|$ and $2f_{AB}d_{12} + |c' - c''|$. For smaller values of b the result will be fractional; but note that simple rounding to the nearest integer always gives the optimal assignment. Larger values of b allow both assignments to be local minima.

This detailed analysis of the trivial 2-machine problem has shown clearly how biasing may be useful on larger problems. As the bias is increased the variables gradually change from fractional to integral and the nature of the objective function changes from convex to concave. It is to be hoped that integrality can always be achieved before concavity, but it has been demonstrated that slight concavity need not prevent satisfactory solution.

3.5 BIASING IN PRACTICE

The matrix generator program was modified to include biases for all the n^2 variables. The bias coefficients could be set independently

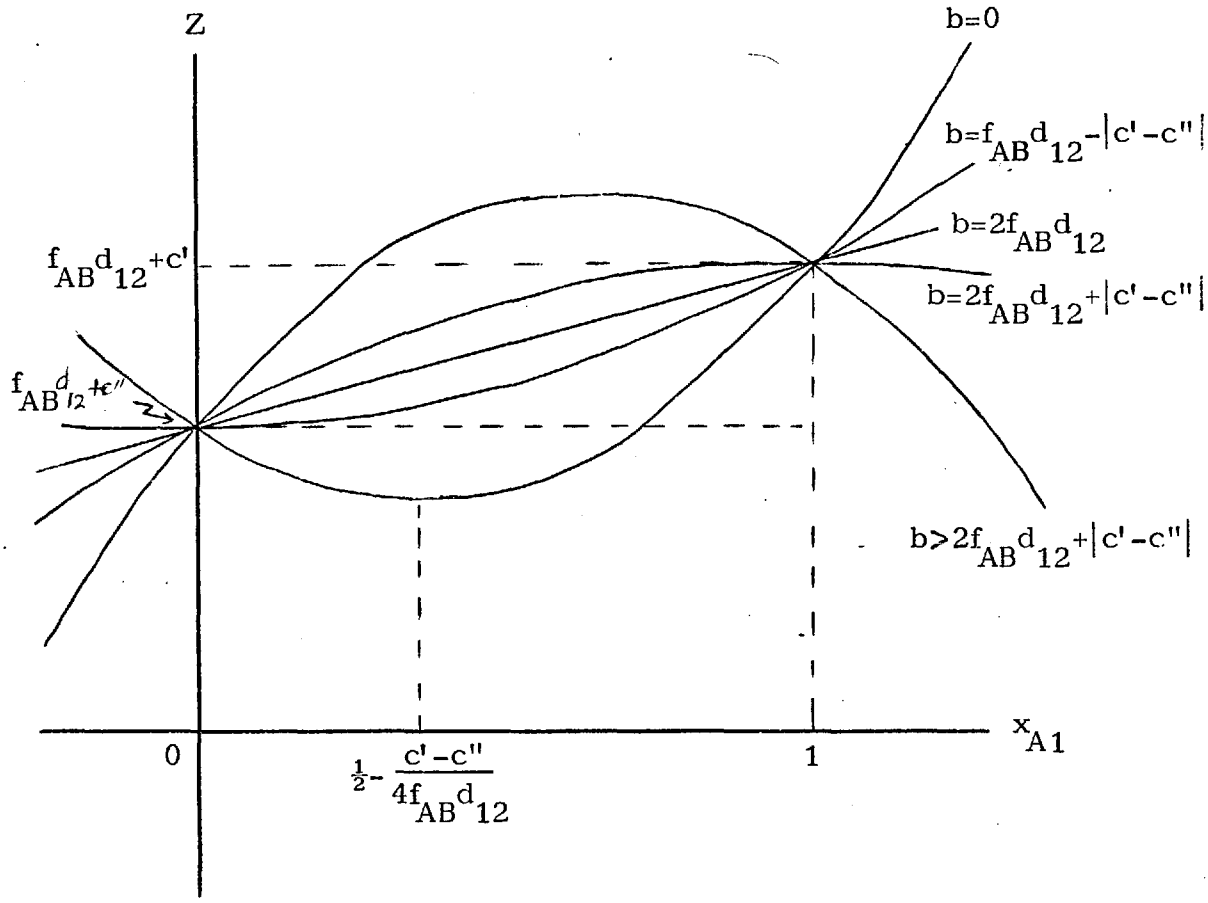


Figure 3-4.

Effect of bias for the 2-machine problem.

of each other by the user. Most problems had to be run several times to achieve optimal, or even good, integer solutions; the coefficients were adjusted after each run, taking account of the information provided by previous results.

Initially the solution to the basic quadratic program with no bias was used. Any variables which were integers already were left unbiased while fractional variables were given a positive bias. It was found best to give all fractional variables the same bias and the value of the coefficients was chosen so that the cost of that particular fractional solution would be increased to marginally more than the cost of an optimal assignment. For the problems studied this optimum was already known; in practice an estimate could be used.

Using these bias coefficients the quadratic program can be solved again. If the solution is still not integral the bias coefficients of the remaining fractional variables can be further increased and the process repeated. This heuristic is expressed more formally in figure 3-5 as a flow-chart. At each iteration the bias and hence the objective function is increased and so an integer solution must be found eventually.

Appendix B shows the detailed working for Nugent's 8-machine problem, NVR8. The procedure worked quite well for this problem, but others needed many more runs to find a set of bias coefficients which produced the optimal assignment. The difficulties encountered with even a 4-machine problem are also shown in appendix B.

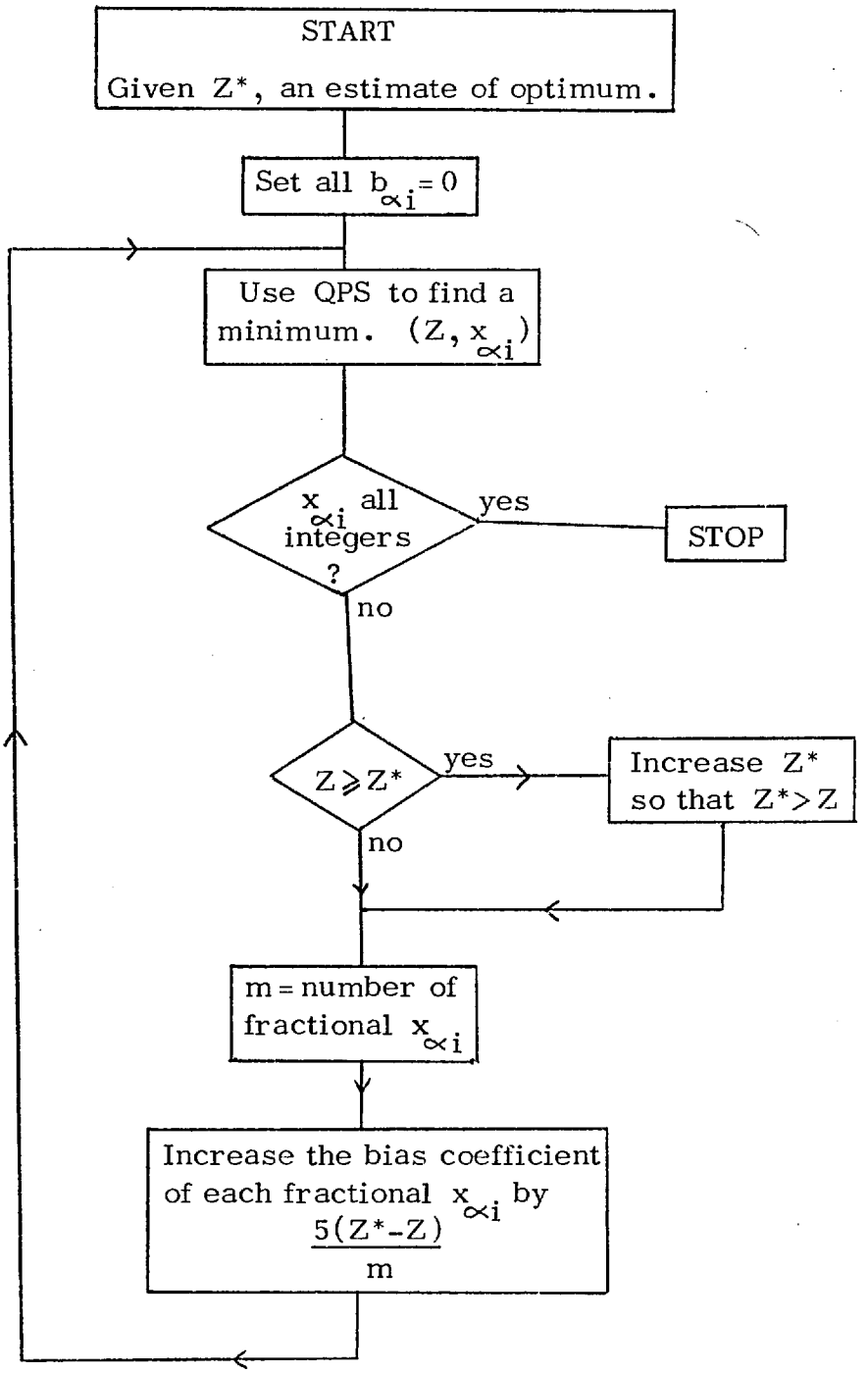


Figure 3-5.

Heuristic procedure for choosing bias coefficients.

An integer solution could sometimes be obtained more quickly by giving all bias coefficients a small positive value and using the solution to this quadratic program as a starting point for the iterative heuristic. Appendix B shows this being sucessfully applied to LA7, Lawler's 7-machine problem.

Judgement and intuition are used in choosing the value of the bias coefficients. If the biasing is too small, many variables will take fractional values. If too large, the objective function will not be convex and it will have many local minima; the QPS package would arbitrarily select any one of these. The remainder of this chapter develops more scientific techniques for choosing bias coefficients.

3.6 CONVEXITY

As stated in section 3.2, the objective function $Z(\underset{\sim}{y}) = \underset{\sim}{r}^T \underset{\sim}{y} + \underset{\sim}{y}^T Q \underset{\sim}{y}$ is only convex if $\underset{\sim}{y}^T Q \underset{\sim}{y}$ is non-negative for all values of $\underset{\sim}{y}$ (66). If Z is non-convex on part of the feasible region defined by the linear constraints then there may be several locally optimal solutions and the QPS package chooses between them arbitrarily. It has been shown how large biases can cause non-convexity, but the unbiased objective function should also be examined.

The 2-machine problem for which $f_{AB} d_{12} = 1$ has

$$\underset{\sim}{y} = \begin{bmatrix} x_{A1} \\ x_{A2} \\ x_{B1} \\ x_{B2} \end{bmatrix} \quad \text{and} \quad Q = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

For any feasible solution $x_{\propto i} \geqslant 0$ and the coefficients of Q are all $\geqslant 0$ and so $\underset{\sim}{y}^T Q \underset{\sim}{y}$ is also $\geqslant 0$. But this is not sufficient to prove that Z is convex in the feasible region.

Consider $\underset{\sim}{y} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ -1 \end{bmatrix}$. Then $\underset{\sim}{y}^T Q \underset{\sim}{y} = -2$. This negative result means that the function in general is not convex, even in the positive region. But it does not take into account the effect of the constraint set. It is conceivable that inclusion of the constraints will make the total system convex. This is certainly true for the 2-machine case, as was shown in section 3.4 .

Definitions of Convexity

A precise definition of convexity is needed for a function and also for a system consisting of a function and a set of constraints; these definitions must relate directly to the existence of multiple local minima. Therefore define $Z(y)$ to be convex if, for all vectors $\underset{\sim}{y}_1$ and $\underset{\sim}{y}_2$, and for all real numbers a, $0 \leqslant a \leqslant 1$, $Z(a\underset{\sim}{y}_1 + (1-a)\underset{\sim}{y}_2) \leqslant a Z(\underset{\sim}{y}_1) + (1-a) Z(\underset{\sim}{y}_2)$. Similarly define a system to be convex if, for all feasible vectors $\underset{\sim}{y}_1$ and $\underset{\sim}{y}_2$, and all real numbers a, $0 \leqslant a \leqslant 1$,

$Z(a\underset{\sim}{y}_1 + (1-a)\underset{\sim}{y}_2) \leqslant a Z(\underset{\sim}{y}_1) + (1-a) Z(\underset{\sim}{y}_2)$.

This definition of a convex function can be shown to be equivalent, in the case of a quadratic function, to the earlier definition involving $\underset{\sim}{y}^T Q \underset{\sim}{y}$ (66). In fact, the vector used above to show that $\underset{\sim}{y}^T Q \underset{\sim}{y}$ can be negative can also be used to show that $Z(\underset{\sim}{y})$ is not convex in the

positive quadrant. Let $\underset{\sim}{y}_1 = \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$, $\underset{\sim}{y}_2 = \underset{\sim}{y}_1 + \begin{bmatrix}1\\0\\0\\-1\end{bmatrix} = \begin{bmatrix}1\\0\\0\\0\end{bmatrix}$ and take $a = \frac{1}{2}$.

For Q as defined before and assuming $\underset{\sim}{r} = 0$, $Z(\underset{\sim}{y}_1) = 0 = Z(\underset{\sim}{y}_2)$ but

$$Z(\tfrac{1}{2}\underset{\sim}{y}_1 + \tfrac{1}{2}\underset{\sim}{y}_2) = Z(\begin{bmatrix}\frac{1}{2}\\0\\0\\\frac{1}{2}\end{bmatrix}) = \tfrac{1}{2} \geqslant \tfrac{1}{2}Z(\underset{\sim}{y}_1) + \tfrac{1}{2}Z(\underset{\sim}{y}_2) .$$

But note that none of these variables are actually feasible. A vector $\underset{\sim}{y} = (x_{A1}\ x_{A2}\ x_{B1}\ x_{B2})^T$ is feasible if and only if $0 \leqslant x_{A1} \leqslant 1$, $x_{A2} = x_{B1} = 1 - x_{A1}$ and $x_{B2} = x_{A1}$. Even if $\underset{\sim}{r} = (c_{A1}\ c_{A2}\ c_{B1}\ c_{B2})^T$ is not zero, $Z(\underset{\sim}{y})$ can be written as a function of one variable :

$$Z(\underset{\sim}{y}) = Z(x_{A1}) = 2x_{A1}^2 + 2(1-x_{A1})^2 + (c_{A1}+c_{B2})x_{A1} + (c_{A2}+c_{B1})(1-x_{A1})$$

$$= (2x_{A1}-1)^2 + (c_{A1} - c_{A2} - c_{B1} + c_{B2})x_{A1} + c_{A1} + c_{B1} + 1 .$$

Now two arbitrary feasible vectors, $\underset{\sim}{y}_1$ and $\underset{\sim}{y}_2$, can be characterised by $x_{A1} = u$ and $x_{A1} = v$ where $0 \leqslant u, v \leqslant 1$.

Then $a Z(\underset{\sim}{y}_1) + (1-a) Z(\underset{\sim}{y}_2) - Z(a\underset{\sim}{y}_1 + (1-a)\underset{\sim}{y}_2)$

$= a(2u-1)^2 + (1-a)(2v-1)^2 - (2au + 2(1-a)v - 1)^2$

$= 4au^2 - 4au + a + 4(1-a)v^2 - 4(1-a)v + 1 - a - 4a^2u^2 - 4(1-a)^2v^2 +$

$\quad - 1 - 8a(1-a)uv + 4au + 4(1-a)v$

$= 4a(1-a)u^2 + 4(1-a)av^2 - 8a(1-a)uv$

$= 4a(1-a)(u-v)^2$

$\geqslant 0$.

Hence the system of objective function and constraints is convex, at least for the 2-machine problem (see also figure 3-4).

Unfortunately it is not at all easy to generalise the preceding analysis to problems with more than two machines, but it is important to know whether or not larger systems are convex.

Empirical Test for Convexity

A pragmatic computerised test for convexity of the quadratic programming system has been developed, based on the actual definition of convexity. Program CONVEX randomly generates two feasible vectors, $\underset{\sim}{y}_1$ and $\underset{\sim}{y}_2$, and evaluates $Z(\underset{\sim}{y}_1)$ and $Z(\underset{\sim}{y}_2)$. For a random number, a, between zero and unity it also evaluates $Z(a\underset{\sim}{y}_1 + (1-a)\underset{\sim}{y}_2)$. Then it calculates the "degree of convexity" (DOC) which is defined to be $aZ(\underset{\sim}{y}_1) + (1-a)Z(\underset{\sim}{y}_2) - Z(a\underset{\sim}{y}_1 + (1-a)\underset{\sim}{y}_2)$.

If the degree of convexity is negative then the system is non-convex. If it is positive for each of a large number of feasible vector pairs then there is a strong likelihood that the system is convex; note that convexity can never be definitely proved in this way.

A flow-chart of CONVEX is given in figure 3-6. The number of vector pairs to be considered must be specified by the user. Note that it is difficult to generate vectors with a uniform probability density over the entire feasible space; the program does not quite succeed, but at least every feasible vector can be generated.

The three problems considered in this chapter were analysed by CONVEX (table 3-1(a)). Many vector pairs were generated for each problem and all the problems were found to be non-convex.

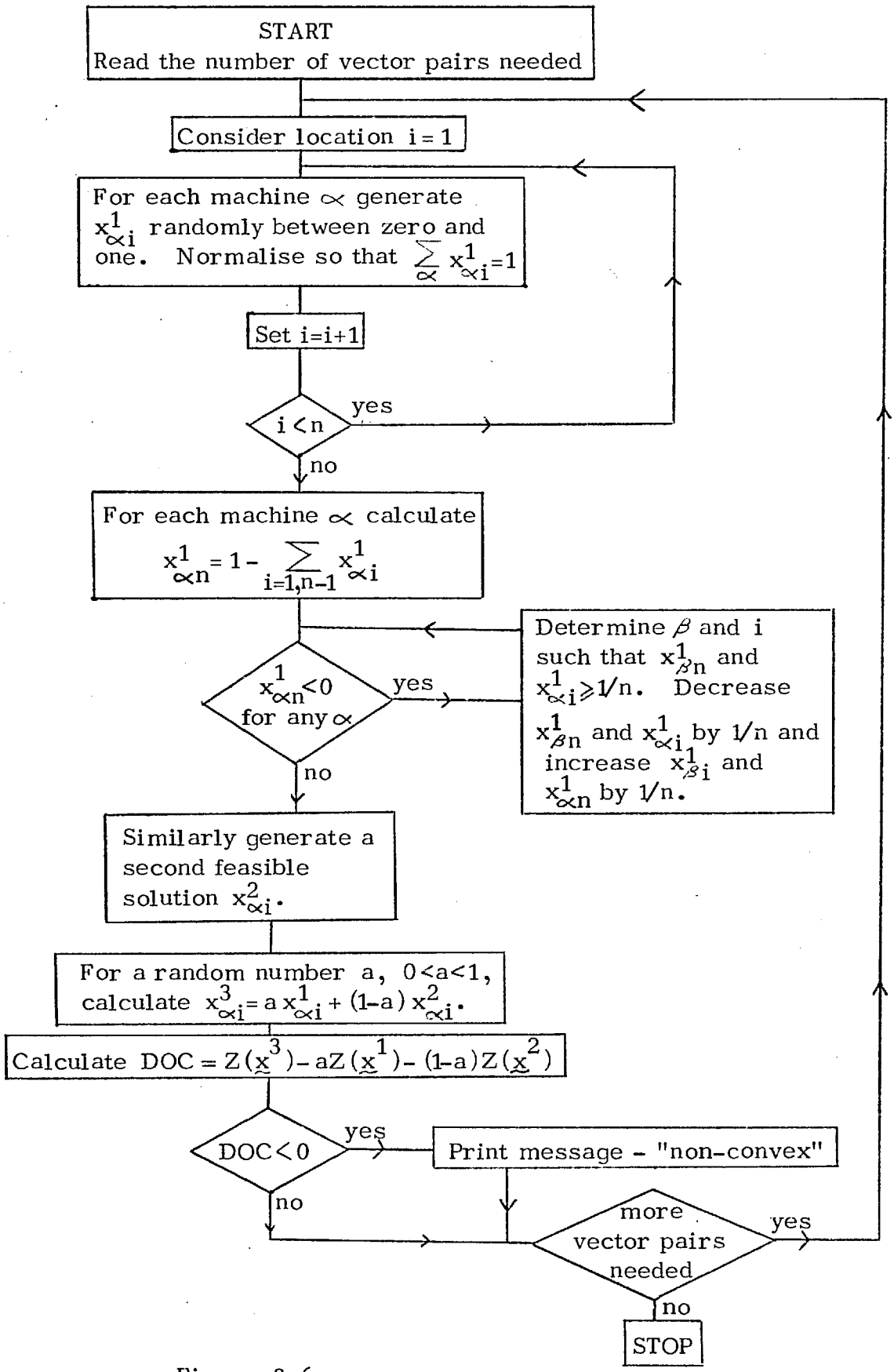


Figure 3-6.

Flow-chart for program CONVEX.

Problem	Number of vector pairs	DOC < 0		Smallest DOC	
		Number	%	Value	As % of optimum
GP4	500	7	1.4	-0.813	0.20
LA7	100	3	3.0	-1.904	0.32
NVR8	50	4	8.0	-0.633	0.59

(a) Test for convexity.

Problem	Number of vector pairs	DOWC < 0		Smallest DOWC	
		Number	%	Value	As % of optimum
GP4	8000	1	0.01	-0.553	0.14
LA7	400	2	0.5	-2.404	0.41
NVR8	300	5	1.6	-0.170	0.16

(b) Test for weak convexity.

Table 3-1.

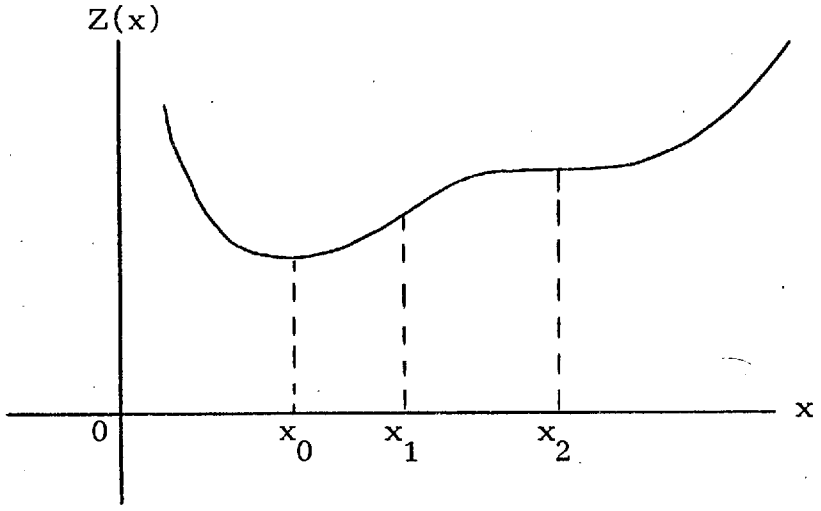
Results of convexity tests.

It appears that the systems are only "slightly" non-convex. Perhaps they do not have multiple minima despite being non-convex; remember that convexity is a sufficient but not a necessary condition for a unique minimum. Figure 3-7 demonstrates that non-convex functions may have only one minimum; graphs (a) and (b) are concave between x_1 and x_2, and x_0 is the global minimum in each case, but graph (b) alone has an additional minimum at x_3.

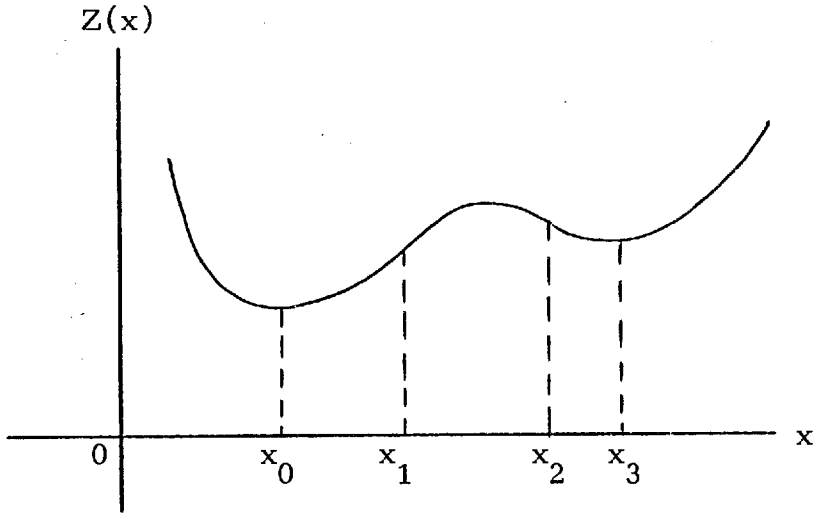
A system consisting of $Z(\underset{\sim}{y})$ and a set of constraints may be defined to be <u>weakly convex</u> if, for all feasible vectors $\underset{\sim}{y}_1$ and $\underset{\sim}{y}_2$, and for all real numbers a, $0 \leqslant a \leqslant 1$, $Z(a\underset{\sim}{y}_1 + (1-a)\underset{\sim}{y}_2) \leqslant \max(Z(\underset{\sim}{y}_1), Z(\underset{\sim}{y}_2))$. It is clear that (1) any convex system is also weakly convex, and (2) any weakly convex system has only one minimum (but the converse is not true for functions of more than one variable). In figure 3-7, function (a) is weakly convex but (b) is not.

Program CONVEX was modified to test for weak convexity. The only change needed is to replace DOC by DOWC (degree of weak convexity) which is defined as $\max(Z(\underset{\sim}{y}_1), Z(\underset{\sim}{y}_2)) - Z(a\underset{\sim}{y}_1 + (1-a)\underset{\sim}{y}_2)$. The results of the modified program are given in table 3-1(b). None of the three problems proved to be weakly convex, although 8000 vector pairs produced only one counter-example for GP4.

All these results suggest that the quadratic programming systems may have non-optimal local minima, but there has been no proof. The QPS package actually provided a direct proof in the case of



(a) A non-convex function with only one minimum.



(b) A non-convex function with two minima.

Figure 3-7.

Multiple minima of non-convex functions.

NVR8 when it found a non-optimal local minimum. This is recorded in figure 3-8.

Negative Biasing

It is easy to show theoretically that any quadratic programming system can be made convex by adding the bias terms of section 3.4 with sufficiently negative coefficients. Program CONVEX has shown that very moderate negative biases are sufficient for the three problems considered here; e.g. no non-convexity was found for NVR8 with all $b_{\propto i} = -1$.

Negative biases, of course, also tend to introduce more fractional variables into the QPS solution. The procedure given in section 3.5 (figure 3-5) appears to be marginally more successful if it is initialised with all biases equally negative; the value chosen should make almost all variables fractional. At least this prevents the possibility of starting from a non-optimal solution as given in figure 3-8.

3.7 DETERMINATION OF BIAS COEFFICIENTS

The purpose of introducing bias is to produce a quadratic program which is at least weakly convex and also has an integer vector as its minimum. These two conditions are to some extent contradictory and success seems to depend on a fine balance between them. It has become apparent that some bias coefficients may have to be negative

$X =$

	1	2	3	4	5	6	7	8
A					.31	.08	.61	
B							.31	.69
C					.69			.31
D	.10	.69				.13	.08	
E			.45	.55				
F			.55	.45				
G	.48	.31				.21		
H	.41					.59		

$Z = 106.90.$

Figure 3-8.

A non-optimal local minimum for NVR8. There is a better solution with $Z = 101.57$.

(See appendix B)

and some positive. The heuristic used earlier is inadequate - even if it does produce an assignment, it can't be proved optimal if the system is not weakly convex.

This section derives formulae for the bias coefficients by considering the partial derivatives of the objective function. Throughout the section the linear cost coefficients, $c_{\alpha i}$, are ignored for the sake of convenience; their inclusion would not present any analytical difficulty, but the algebra would become rather tedious.

First consider the partial derivatives of the unbiased objective function.

$$Z(X) = \sum_{\alpha}\sum_{\beta>\alpha}\sum_{i}\sum_{j\neq i} f_{\alpha\beta}\, d_{ij}\, x_{\alpha i}\, x_{\beta j}$$

$$\frac{\partial Z}{\partial x_{\gamma k}} = \sum_{\alpha}\sum_{\beta>\alpha}\sum_{i}\sum_{j\neq i} f_{\alpha\beta}\, d_{ij} \left\{ x_{\alpha i}\, \frac{\partial x_{\beta j}}{\partial x_{\gamma k}} + x_{\beta j}\, \frac{\partial x_{\alpha i}}{\partial x_{\gamma k}} \right\}$$

$$= \sum_{\alpha<\gamma}\sum_{i\neq k} f_{\alpha\gamma}\, d_{ik}\, x_{\alpha i} + \sum_{\beta>\gamma}\sum_{j\neq k} f_{\gamma\beta}\, d_{kj}\, x_{\beta j}$$

$$= \sum_{\beta\neq\gamma}\sum_{j\neq k} f_{\gamma\beta}\, d_{kj}\, x_{\beta j}$$

i.e. $$\frac{\partial Z}{\partial x_{\alpha i}} = \sum_{\beta\neq\alpha}\sum_{j\neq k} f_{\alpha\beta}\, d_{ij}\, x_{\beta j}$$

Thus the partial derivative of $Z(X)$ with respect to the variable $x_{\alpha i}$ is a positive linear combination of the other variables and is independent of $x_{\alpha i}$ itself.

If a particular $x_{\alpha i}$ is varied while the other variables are fixed, Z increases linearly with $x_{\alpha i}$. If the only constraints were that

the variables could not take negative values then clearly Z would achieve its minimum value when all the variables were zero. To find a minimal assignment the constraints must be incorporated somehow.

Substitution of Constraints

The equality constraints (3-3) of the quadratic program can be rewritten:

$$x_{\alpha 1} = 1 - \sum_{i \neq 1} x_{\alpha i} \quad \text{for all machines } \alpha$$
$$\text{and} \quad x_{Ai} = 1 - \sum_{\alpha \neq A} x_{\alpha i} \quad \text{for all locations } i . \qquad 3\text{-}9$$

Then $x_{A1} = 1 - \sum_{i \neq 1} x_{Ai} = 2 - n + \sum_{\alpha \neq A} \sum_{i \neq 1} x_{\alpha i}$. These 2n-1 marginal variables can now be eliminated from the objective function and hence the equality constraints become implicitly incorporated in the objective function.

After some simplification this substitution gives:

$$Z = \sum_{\substack{\alpha \neq A \\ (\alpha,i) \neq (\beta,j)}} \sum_{\beta \geq \alpha} \sum_{i \neq 1} \sum_{j \neq 1} (f_{\alpha A} + f_{A\beta} - f_{\alpha\beta})(d_{i1} + d_{1j} - d_{ij}) x_{\alpha i} x_{\beta j} +$$
$$+ 2 \sum_{\alpha \neq A} \sum_{i \neq 1} f_{\alpha A} d_{i1} x_{\alpha i}^2 + F_A D_1 +$$
$$+ \sum_{\alpha \neq A} \sum_{i \neq 1} \left\{ (F_\alpha - F_A) d_{i1} + f_{\alpha A}(D_i - D_1) - n f_{\alpha A} d_{i1} \right\} x_{\alpha i}$$

where $F_\alpha = \sum_{\beta \neq \alpha} f_{\alpha\beta}$ and $D_i = \sum_{j \neq i} d_{ij}$.

The result of differentiation is:

$$\frac{\partial Z}{\partial x_{\alpha i}} = \sum_{\substack{\beta \neq A \\ (\beta,j) \neq (\alpha,i)}} \sum_{j \neq 1} (f_{\alpha A} + f_{A\beta} - f_{\alpha\beta})(d_{i1} + d_{1j} - d_{ij}) x_{\beta j} + 4 f_{\alpha A} d_{i1} x_{\alpha i} +$$
$$+ (F_\alpha - F_A) d_{i1} + f_{\alpha A}(D_i - D_1) - n f_{\alpha A} d_{i1} . \qquad 3\text{-}10$$

Clearly the partial derivative of $Z(X)$ with respect to $x_{\alpha i}$ is a linearly increasing function of $x_{\alpha i}$ and so $Z(X)$ is a convex quadratic function of $x_{\alpha i}$. This convexity ensures that Z as a function of $x_{\alpha i}$ does not have multiple minima over the range 0 to 1, but it does not ensure that the minimum will be at an integer point.

The minimum will be at 0 or 1 if the derivative has the same sign over the whole range from $x_{\alpha i}= 0$ to $x_{\alpha i}= 1$. The best way of achieving this is to force the derivative to be independent of $x_{\alpha i}$ by suitable choice of bias coefficients.

The total bias is $B = \sum_{\alpha} \sum_{i} b_{\alpha i} x_{\alpha i}(1-x_{\alpha i})$. Substitution for the marginal variables according to formula 3-9 leads to:

$$B = -b_{A1} \sum_{\alpha\beta ij} x_{\alpha i} x_{\beta j} - \sum_{\alpha\beta i} b_{Ai} x_{\alpha i} x_{\beta i} - \sum_{\alpha ij} b_{\alpha 1} x_{\alpha i} x_{\alpha j} - \sum_{\alpha i} b_{\alpha i} x^2_{\alpha i} +$$

$$+ \sum_{\alpha i} (b_{\alpha i}+b_{Ai}+b_{\alpha 1}+(2n-3)b_{A1}) x_{\alpha i} - (n-1)(n-2)b_{A1}$$

where all summations exclude machine A and location 1.

The partial derivative of the bias function is:

$$\frac{\partial B}{\partial x_{\alpha i}} = -2b_{A1} \sum_{(\beta,j)\neq(\alpha,i)} x_{\beta j} - 2b_{\alpha 1} \sum_{j\neq i} x_{\alpha j} - 2b_{Ai} \sum_{\beta\neq\alpha} x_{\beta i} + \qquad 3\text{-}11$$

$$+ b_{\alpha i}+b_{Ai}+b_{\alpha 1}+(2n-3)b_{A1} - 2(b_{A1}+b_{Ai}+b_{\alpha 1}+b_{\alpha i}) x_{\alpha i}.$$

This is a linearly decreasing function of $x_{\alpha i}$ and so B is a concave quadratic function of $x_{\alpha i}$.

The sum $Z+B$ is to be made a linear function of $x_{\alpha i}$ and so $\frac{\partial(Z+B)}{\partial x_{\alpha i}}$ must be made independent of $x_{\alpha i}$. This can only be achieved if $4f_{\alpha A}d_{i1} - 2(b_{A1}+b_{Ai}+b_{\alpha 1}+b_{\alpha i}) = 0$.

i.e. $b_{\alpha i} = 2f_{\alpha A}d_{i1} - b_{A1} - b_{Ai} - b_{\alpha 1}$. 3-12

For any choice of the 2n-1 marginal bias coefficients this formula gives the unique values of the remaining coefficients which make Z+B linear in each variable when considered alone. Subject to the constraints $0 \leqslant x_{\alpha i} \leqslant 1$ for $\alpha \neq A$ and $i \neq 1$ any local minimum of Z+B will be integer. But note that Z+B still contains products of variables and so is not a true linear function and need not be convex.

Practical Techniques

Bias coefficients calculated according to formula 3-12 could simply be used as input for the QPS quadratic programming package, but the above analysis suggests a more direct method for finding a minimal solution.

Consider any 0-1 solution to the problem with the marginal variables removed. The partial derivatives of Z+B can be calculated; combining equations 3-10, 3-11 and 3-12 gives an explicit formula:

$$\frac{\partial(Z+B)}{\partial x_{\alpha i}} = \sum_{\substack{\beta \neq A \\ (\beta,j) \neq (\alpha,i)}} \sum_{j \neq 1} \left\{ (f_{\alpha A} + f_{A\beta} - f_{\alpha\beta})(d_{i1} + d_{1j} - d_{ij}) - 2b_{A1} \right\} - 2b_{Ai} \sum_{\substack{\beta \neq \alpha \\ \beta \neq A}} x_{\beta i}$$

$$- 2b_{\alpha 1} \sum_{\substack{j \neq 1 \\ j \neq i}} x_{\alpha j} + (F_{\alpha} - F_{A})d_{i1} + f_{\alpha A}(D_i - D_1) - (n-2)(f_{\alpha A}d_{i1} - 2b_{A1}) \qquad 3\text{-}13$$

Now if the present solution contains any variable $x_{\alpha i} = 1$ while $\frac{\partial(Z+B)}{\partial x_{\alpha i}} > 0$, the cost of the solution could be reduced by changing $x_{\alpha i}$ to 0. Similarly if $x_{\alpha i} = 0$ and the derivative is negative, the cost would be reduced by setting $x_{\alpha i} = 1$. Changing $x_{\alpha i}$ does not change the

derivative with respect to $x_{\alpha i}$ but it may change other partial derivatives.

This change in the value of one variable is the basic step of the iterative procedure defined by a flow-chart in figure 3-9. At each step the objective function is reduced until a minimum is found; then all variables will be compatible with their derivatives. Since Z+B may not be convex there is no guarantee that this local minimum will be the global minimum, but it will be integral and it will be a solution which QPS could have produced, given the same bias coefficients.

There are two ways in which the solution found by the iterative procedure might prove unsatisfactory. Firstly, it may not represent an assignment. The marginal variables have been removed from the problem and so the values imputed to them may be negative and the corner variable x_{A1} may be greater than one.

The second fear is that, although feasible, the assignment may not be optimal because the objective function may be non-convex. The method allows a completely free choice of the 2n-1 marginal bias coefficients and it is hoped that this flexibility will overcome both these difficulties.

Computational Experience

How does this procedure perform in practice ? The 2-machine problem is trivial since the (partial) derivative is always zero, indicating that the two possible assignments have the same cost; if $c_{\alpha i}$ terms are introduced the procedure successfully chooses the cheaper

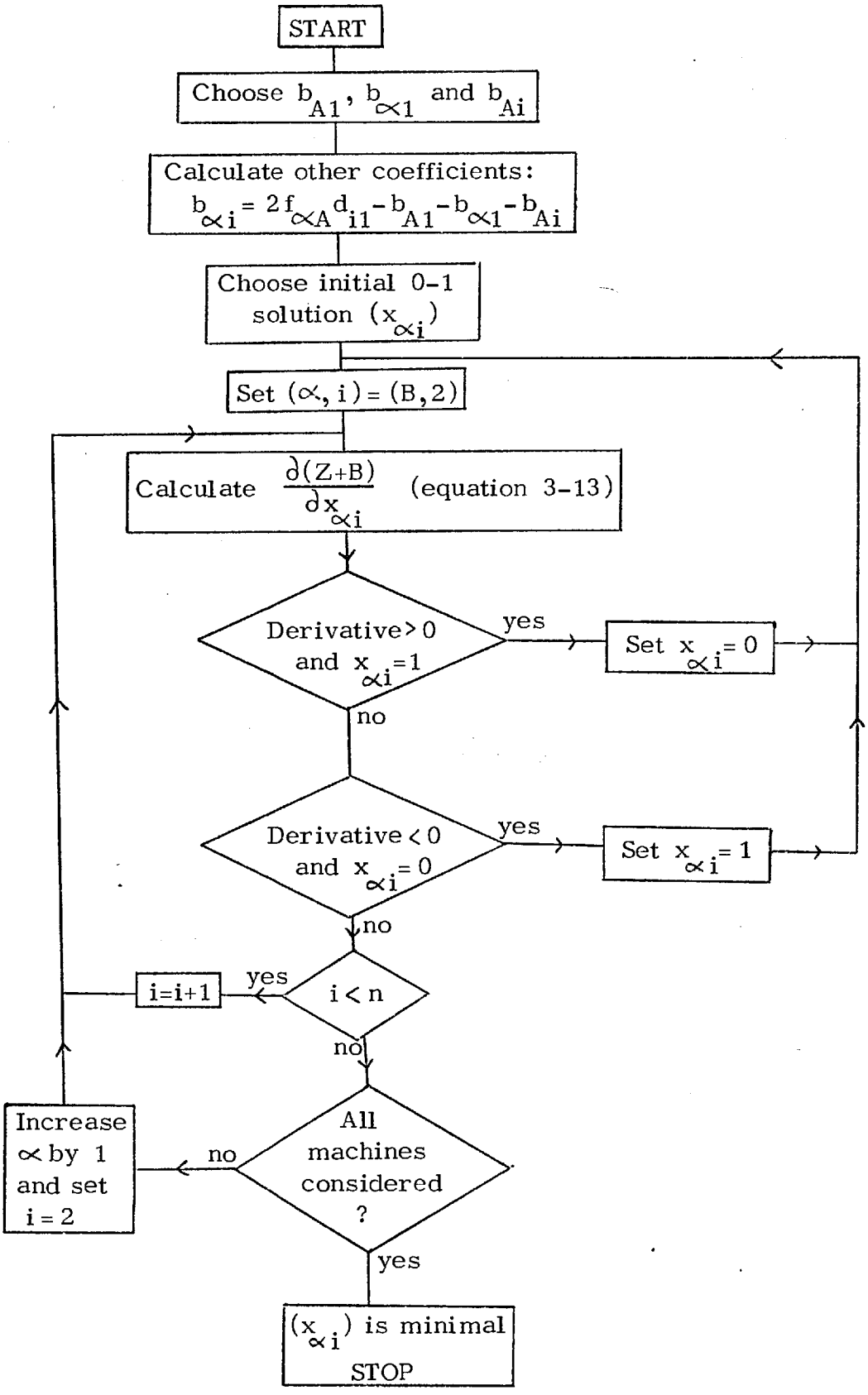


Figure 3-9.

Iterative procedure to minimise Z+B.

assignment.

A 3-machine problem was analysed in detail manually with the marginal bias coefficients all set to zero. From all feasible initial solutions the procedure converged to one of three local minima. Of these minima, one was the true optimal assignment (Z=19), a second was feasible but non-optimal (Z=21) and the third was not feasible (Z=20, $x_{B2}=1=x_{C2}$ which means that $x_{A2}=-1$). Adjusting the marginal bias coefficients successfully removed the infeasible solution, but an ad hoc analysis showed that no choice of marginal coefficients could remove the second solution without introducing an infeasible solution.

The difficulty of infeasible solutions does not arise if the bias coefficients are used with the QPS package and the marginal variables are left in. Figure 3-10 shows the outcome of doing this for NVR8 with all the marginal coefficients zero. The solution is not an assignment, but three points are worth noting.

Firstly, the magnitude of the biases is much greater than was used with some success in section 3.5. Here there are two coefficients of 80 whereas appendix B gives an optimal assignment without using values greater than 6.

Secondly, the cost of the solution is almost 50% greater than the optimum. This indicates that the biased objective function is highly non-convex - not surprising in view of the huge biases used.

B =

	1	2	3	4	5	6	7	8
A	48	36	24	12	36	24	12	0
B	0	0	0	0	0	0	0	0
C	40	30	20	10	30	20	10	0
D	80	60	40	20	60	40	20	0
E	0	0	0	0	0	0	0	0
F	8	6	4	2	6	4	2	0
G	80	60	40	20	60	40	20	0
H	0	0	0	0	0	0	0	0

X =

	1	2	3	4	5	6	7	8
A						1		
B					1			
C							1	
D				1				
E	.625		.375					
F	.375		.625					
G								1
H		1						

Z = 154.75 (Optimum is 107.)

Figure 3-10.

Result of using calculated bias with QPS for NVR8.

Note: the marginal machine used here was H rather than A and the marginal location was 8 rather than 1.

The third point concerns the variables themselves; most are integers but four are not. This means that the iterative procedure of figure 3-9 would have forced some marginal variables to be negative. The solution might have been as in figure 3-10 except for $x_{E1}=1=x_{F1}=x_{E3}=x_{F3}$, $x_{H1}=-1=x_{H3}=x_{E8}=x_{F8}$ and $x_{H8}=2$. Note that the fractional variables are exact fractions (3/8 and 5/8); this is probably a consequence of the partially linear nature of the biased objective function.

When the marginal bias coefficient b_{E8} was changed from 0 to -2, QPS found a proper assignment, but it was very far from optimal.

The results of the formula for bias coefficients have proved so unpromising that no further research has been done to determine values for the marginal coefficients. A different approach seemed to be required, one which did not arbitrarily select one machine and one location and consider them different to the others.

3.8 PENALTY FUNCTION METHOD

The previous section incorporated the equality constraints (3-3) into the objective function by eliminating some of the variables; unfortunately this also eliminated the inequality constraints associated with those variables. This section incorporates the equality constraints by adding a penalty function to the objective function in a similar way to

the bias function introduced in section 3.4 .

P(X) is a suitable penalty function if its value is zero for all vectors satisfying the constraints but very large for all infeasible solutions. Then the minimum value of Z+P must automatically satisfy the constraints. A penalty function as described above is impractical because it must be discontinuous at the boundary of the feasible space. It proves sufficient to simply require P to be positive for all infeasible solutions.

Since the objective function and the bias function are quadratic it is convenient to use the following quadratic penalty function:

$$P(X) = \sum_{\alpha} p_{\alpha} \left(\sum_{i} x_{\alpha i} - 1\right)^2 + \sum_{i} q_i \left(\sum_{\alpha} x_{\alpha i} - 1\right)^2$$ for positive parameters p_{α} and q_i. This is clearly zero if $\sum_{i} x_{\alpha i} = 1$ for all machines α and $\sum_{\alpha} x_{\alpha i} = 1$ for all locations i ; otherwise its value is positive.

The penalty function does not force the variables to be integers and so the bias function is still needed. The complete objective function to be minimised is now Z+B+P where

$$Z = \sum_{\alpha} \sum_{\beta > \alpha} \sum_{i} \sum_{j \neq i} f_{\alpha\beta} d_{ij} x_{\alpha i} x_{\beta j}$$

$$B = \sum_{\alpha} \sum_{i} b_{\alpha i} x_{\alpha i} (1 - x_{\alpha i})$$

and P is defined above. Note that the linear cost coefficients $c_{\alpha i}$ are again being omitted; they add neither difficulty nor interest to the following analysis.

The partial derivatives of each of these 3 components are:

$$\frac{\partial Z}{\partial x_{\alpha i}} = \sum_{\beta \neq \alpha} \sum_{j \neq i} f_{\alpha\beta} d_{ij} x_{\beta j}$$

$$\frac{\partial B}{\partial x_{\alpha i}} = b_{\alpha i}(1-2x_{\alpha i})$$

and $$\frac{\partial P}{\partial x_{\alpha i}} = 2p_{\alpha}(\sum_{j} x_{\alpha j} - 1) + 2q_{i}(\sum_{\beta} x_{\beta i} - 1) .$$

As in the previous section it is useful if the bias coefficients and penalty parameters are chosen to make the complete objective function linear in each variable when considered as a function of that variable alone; i.e. the partial derivative of Z+B+P with respect to $x_{\alpha i}$ should be independent of $x_{\alpha i}$. This requires $-2b_{\alpha i} + 2p_{\alpha} + 2q_{i} = 0$, or more simply, $b_{\alpha i} = p_{\alpha} + q_{i}$.

This linearising choice of bias coefficients ensures that any minimum of the complete objective function, subject only to $0 \leq x_{\alpha i} \leq 1$, will be a 0-1 solution. An iterative algorithm analagous to that proposed in the previous section can be used to find such a minimum. A flow-chart for the method is given in figure 3-11. The formula for the partial derivatives is

$$\frac{\partial(Z+B+P)}{\partial x_{\alpha i}} = \sum_{\beta \neq \alpha} \sum_{j \neq i} f_{\alpha\beta} d_{ij} x_{\beta j} + 2p_{\alpha} \sum_{j \neq i} x_{\alpha j} + 2q_{i} \sum_{\beta \neq \alpha} x_{\beta i} - p_{\alpha} - q_{i} . \quad 3\text{-}14$$

3.9 RESULTS OF PROGRAM "PENALTY"

The penalty procedure was programmed in FORTRAN to be run interactively on the College's CDC 6400 computer. The interactive

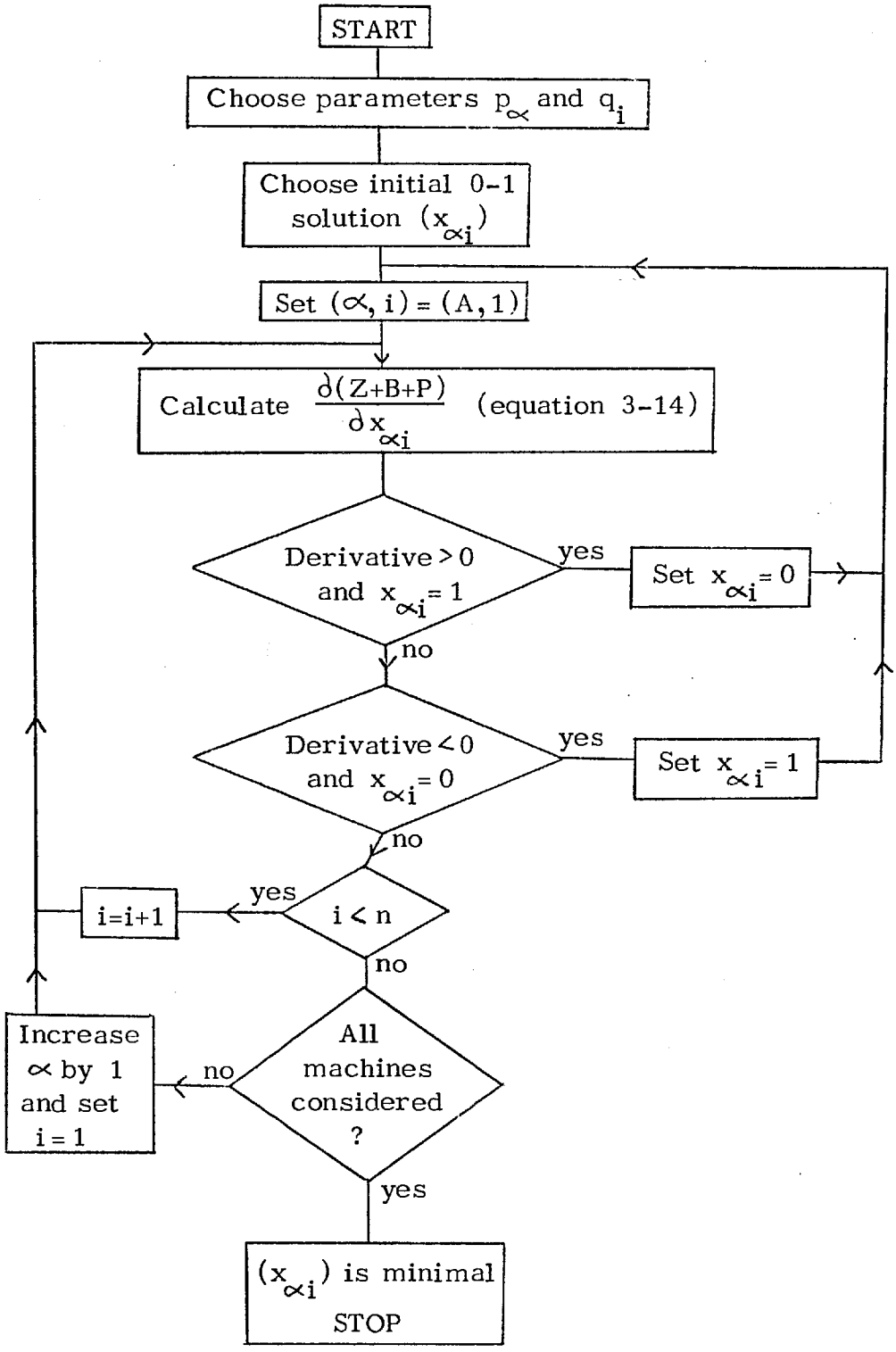


Figure 3-11.

Iterative procedure to minimise Z + B + P.

facility enabled the user to alter the penalty parameters from a terminal after a local minimum had been found, and also to specify different initial solutions.

The basic procedure defined by figure 3-11 would be inefficient because it calculates many derivatives unnecessarily and each derivative is the sum of approximately n^2 terms. Program PENALTY starts with the penalty parameters all zero and uses $x_{\alpha i} = 0$ as its initial (infeasible) solution; hence the derivatives at the beginning are simply equal to $c_{\alpha i}$ (PENALTY includes the linear cost coefficients). Derivatives are never completely recalculated, but merely adjusted whenever a variable changes or the penalties are altered.

Figure 3-12 gives an outline of the program's logic. The user controls execution by typing key-words of which only the three most basic are shown in the flow-chart. To increase $p_{B\Lambda}$ by 3 and decrease q_1 by 1.7, for example, the user would type "alter B 3 1-1.7". Key-words which have been omitted from the flow-chart control the display of information to the user. An example showing the use of the program is included in appendix C.

The success of the program obviously depends on the penalty parameters used. If the parameters are too small some of the constraints may be broken; if too large the objective function becomes very non-convex and many non-optimal local minima are created.

With a little experimentation it is easy to find assignments with costs within 10% of the optimum. Even for large problems with

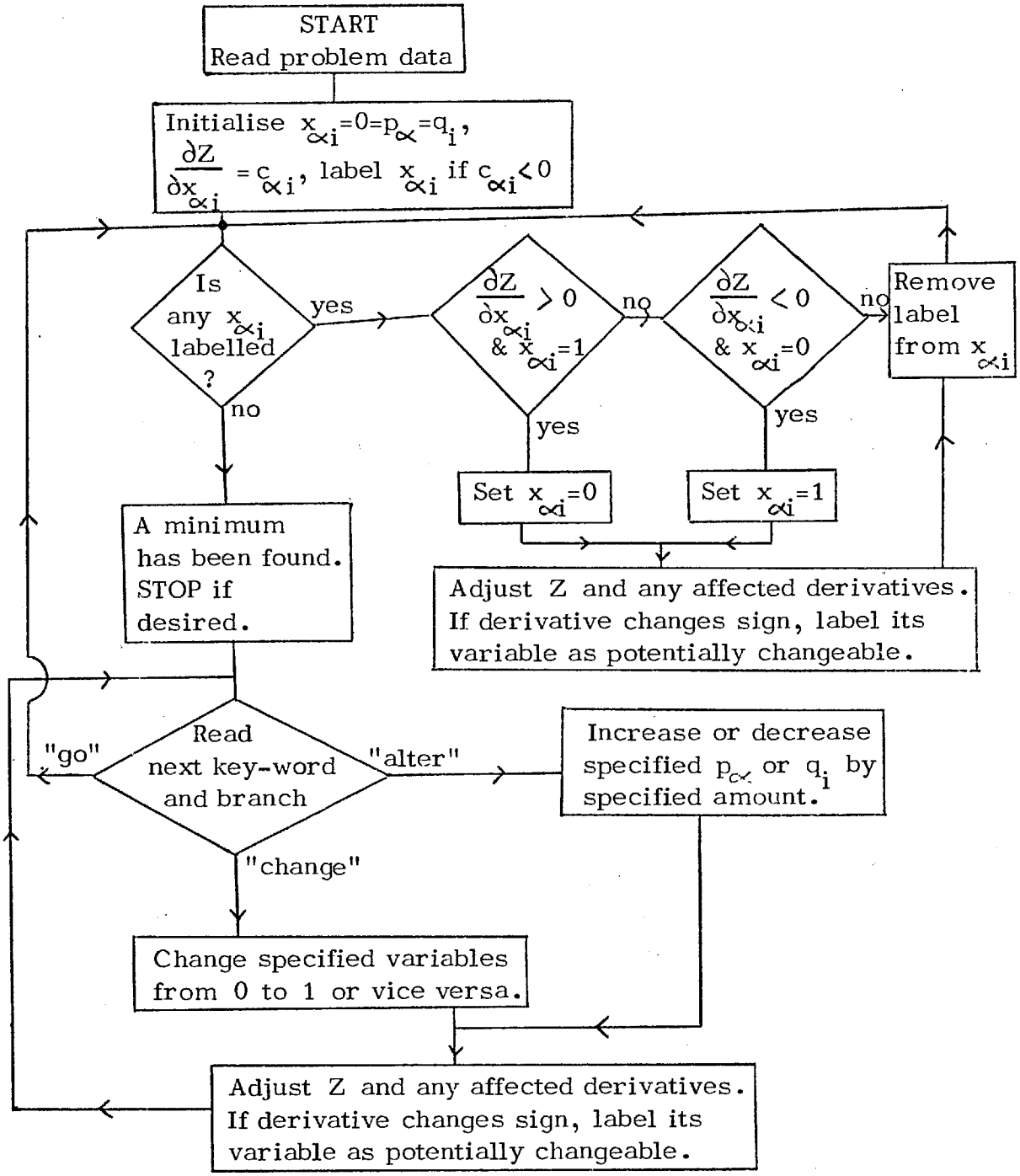


Figure 3-12.

Flow-chart of program PENALTY.

as many as twenty machines the computer time required is very small, usually less than ten seconds. Hence this program is at least a useful heuristic, even if it proves impossible to determine penalty parameters which produce the optimal solution.

Satisfactory penalties were found for a 2-machine problem. From any of the 16 feasible and infeasible initial solutions the program reached the optimal solution using the penalties $p_A = 10$, $p_B = 9.9$, $q_1 = 9.85$ and $q_2 = 10$. The details of this problem and its convergence to the optimal assignment are given in figure 3-13.

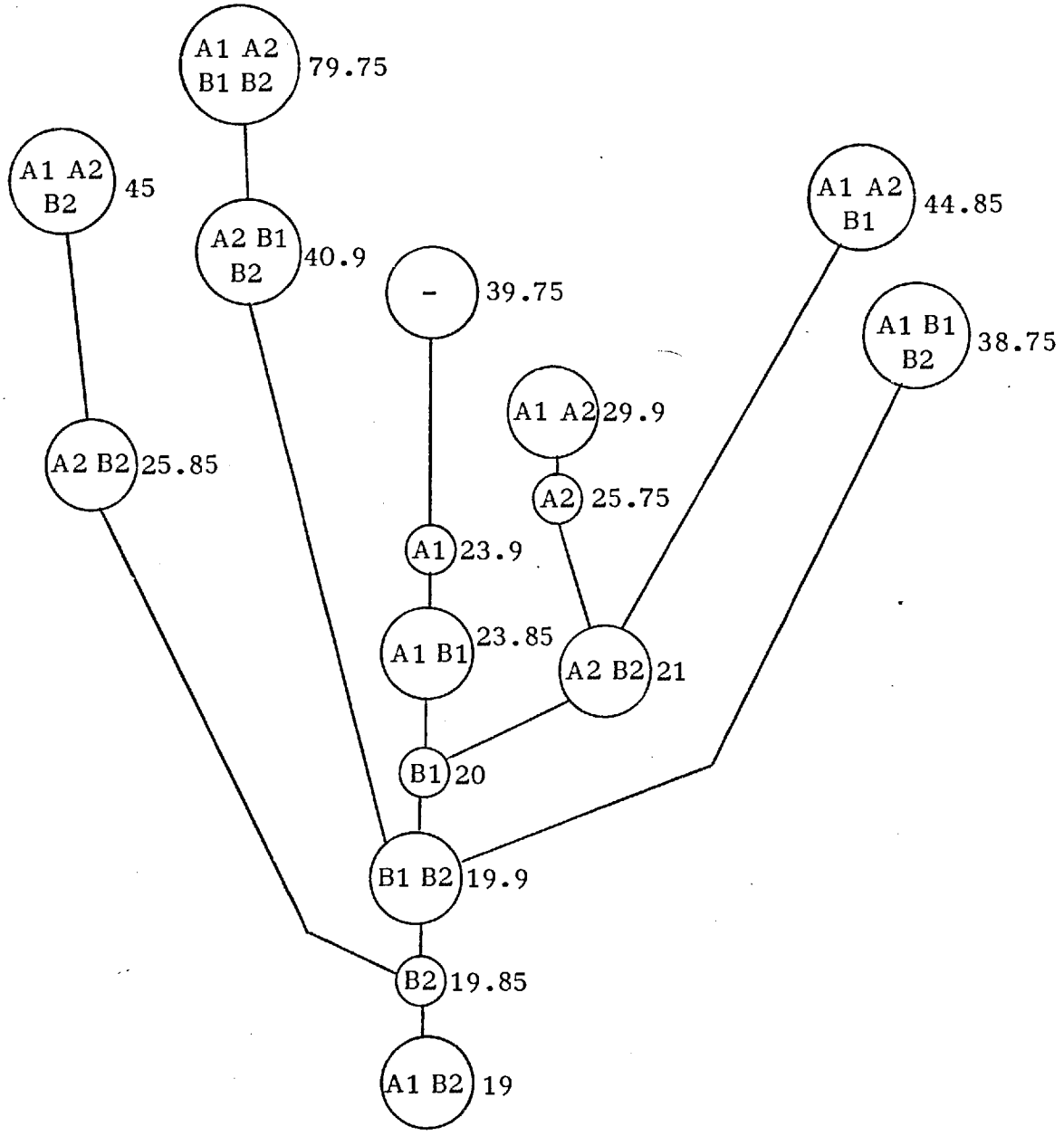
For larger problems such as NVR8 and NVR12 the program found optimal assignments after the penalty parameters were selectively altered several times. But if the parameters were kept fixed and a new initial solution used, the program sometimes converged to a different solution which could be non-optimal or infeasible. This indicates the non-convexity of the function.

3.10 CALCULATING PENALTIES WITH L.P.

The penalty parameters must be large enough to make minimal solutions feasible, but as small as possible to make the complete objective function convex. This suggests that mathematical programming might be used to choose the parameters. The objective is to minimise

$$V = \sum_{\alpha} p_{\alpha} + \sum_{i} q_i \qquad 3\text{-}15$$

and the constraints must eliminate infeasible solutions.



Problem data: $F_{AB}= 3$, $d_{12}= 5$, $c_{A1}= 4$, $c_{A2}= 6$, $c_{B1}= 0 = c_{B2}$.

Penalty parameters used: $p_A=10=q_2$, $p_B= 9.9$, $q_1= 9.85$.

Figure 3-13.

This 2-machine problem was solved by PENALTY. The "tree" shows the costs of all 16 solutions and how they converge on the optimum (A1 B2).

Suppose the cost of an optimal assignment is known, or at least suspected, to be Z^*. If a solution X is infeasible then it can be eliminated by the constraint

$$Z(X) + B(X) + P(X) > Z^* \qquad 3\text{-}16$$

This constraint can be written in the form

$$r + \sum_{\alpha} s_{\alpha} p_{\alpha} + \sum_{i} t_i q_i > Z^*$$

where the coefficients r, s_{α} and t_i depend only on the problem data and the solution X. Hence constraint 3-16 is linear and so function 3-15 and constraints of the form 3-16 together constitute a basis for a linear program; its solution gives the parameters to be used by program PENALTY.

But the number of constraints is quite immense - one for every infeasible solution. Many of these constraints will be dominated by others and so perhaps quite a small subset of the constraints will suffice. Figure 3-14 describes a method for selecting a satisfactory set of constraints.

This method successfully found the optimal assignment for GP4 and a good sub-optimal assignment for LA7. NVR8 could not be solved, however, because after six iterations the linear program had no feasible solution. A 3- machine problem, MG3, was subsequently constructed which also gave incompatible constraints. The data for this problem is included in appendix A.

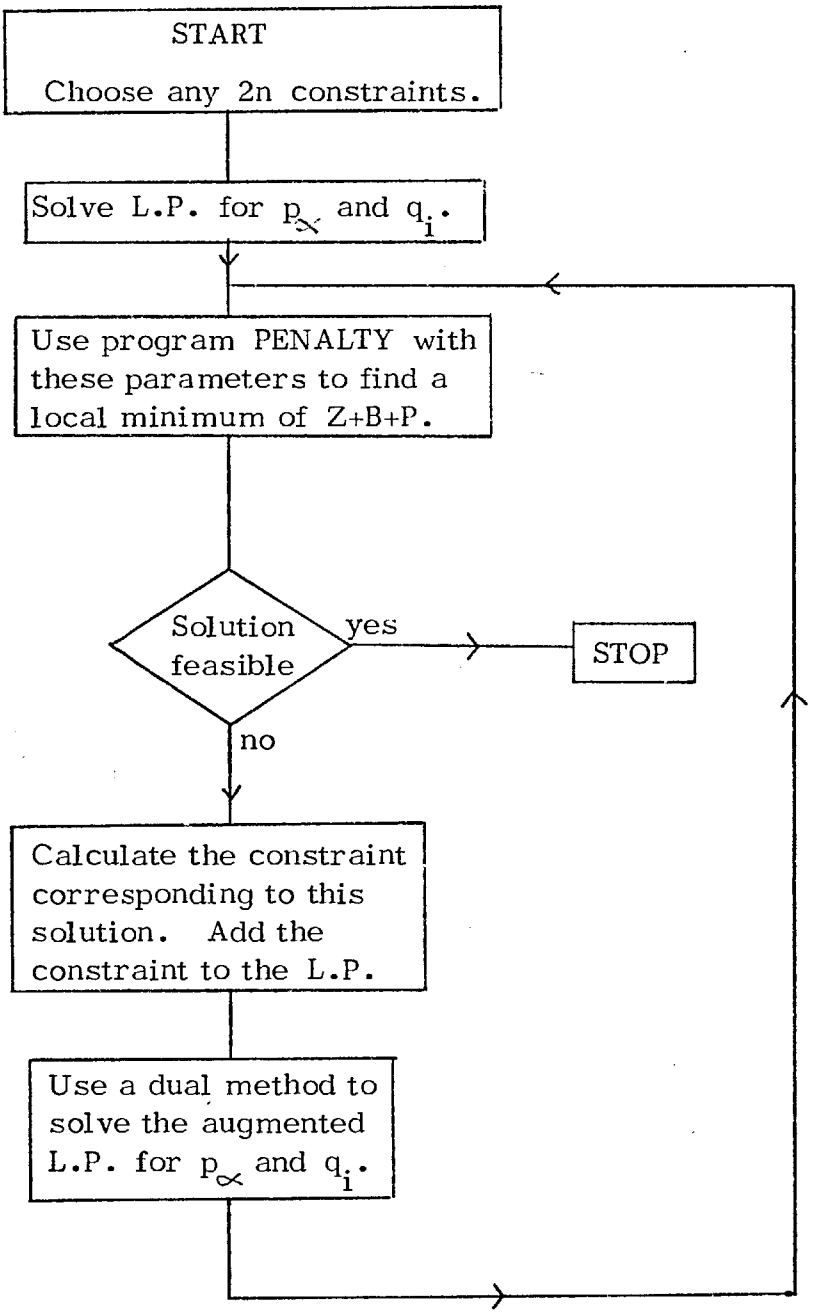


Figure 3-14.

Determining penalty parameters with linear programming.

3.11 CONCLUSIONS

This chapter has expressed the quadratic assignment problem as an integer quadratic program. Standard quadratic programming algorithms can only solve continuous convex programs, but it was hoped that the special nature of the formulation would suggest a suitable technique.

Several methods were proposed, of which the most successful modified the objective function to incorporate the constraints. Some techniques produced integer solutions which were not optimal because the function was not convex. Others were able to produce a unique minimum which was fractional. Still others found unique, integer solutions which were not feasible. No method was able to combine all the necessary attributes to find assignments which could be proved optimal.

These techniques can be useful, even though none of them can guarantee optimal assignments. The methods which give feasible, integer solutions usually find good assignments; program PENALTY is an example of such a heuristic which has the advantage of very fast computing time.

Even methods which yield fractional solutions are useful if the system is convex. The cost of such a solution is a lower bound on the cost of an optimal assignment and could be used in a branch-and-bound algorithm.

CHAPTER 4

TREE-SEARCH ALGORITHMS

4.1 WHAT IS TREE-SEARCH ?

Chapter 2 described several "tree-search" or "branch-and-bound" algorithms for the quadratic assignment problem and this chapter proposes a new one with a group of variations. It may be useful to begin with a brief discussion of tree-search techniques in general. A mathematical treatment of this subject has been given by Mitten (48) and Lawler and Wood (41) have compared some of its early applications.

Tree-search has been most frequently used for combinatorial optimising problems, like the quadratic assignment problem, for which there is a large but finite number of feasible solutions. The object of tree-search is to partition the set of feasible solutions into a collection of disjoint subsets; this partitioning is to be done in such a way that one of the subsets is singleton and the cost of its element can be shown to be better than all the solutions in each of the other subsets. The solution in this singleton subset must therefore be optimal.

Henceforth assume that optimising means minimising. Proving that the cost of the singleton, Z_0 say, is less than the cost of all solutions in a particular subset is equivalent to showing that Z_0 is a lower bound on the costs of solutions in that subset, or that Z_0 is less than such a lower bound. Calculation of lower bounds plays a major part in any tree-search algorithm and these bounds should be as large as possible.

The partitioning of feasible solutions into subsets usually proceeds in several stages or "levels". First the feasible solutions are partitioned

into a small number of subsets, often only two. Lower bounds are calculated for these subsets, which are considered as being "nodes" at level 1 of a "tree". Taking account of the bounds, one of the subsets is then partitioned into smaller subsets which become nodes at level 2. Lower bounds are calculated for each new node and the partitioning process continues until an optimal solution is found and proven optimal.

This repeated partitioning is conveniently represented as a directed graph as in figure 4-1. In graph theoretic terms this graph is a tree and it also resembles an inverted tree in the botanical sense; hence the name "tree-search". Partitioning a node is known as branching and this explains the alternative name "branch-and-bound".

At any point during the branching process those nodes which have not been partitioned are called "pendant" or "hanging" nodes. Any singleton subset is obviously pendant and the actual cost of its solution is usually used as the lower bound for the node. The solution in a singleton node is optimal if its cost is less than or equal to the bounds on all other pendant nodes.

A hypothetical problem which has only 20 feasible solutions is solved using tree-search in figure 4-1. The first partitioning is into three subsets of unequal size. Node 3 has the smallest bound and so appears most likely to contain a minimal solution; hence the next partitioning is of this subset. Node 5 is singleton and the cost of its solution is 77, which is less than the bounds on nodes 4 and 6; these two nodes may therefore be discarded. The next branching is from node 2

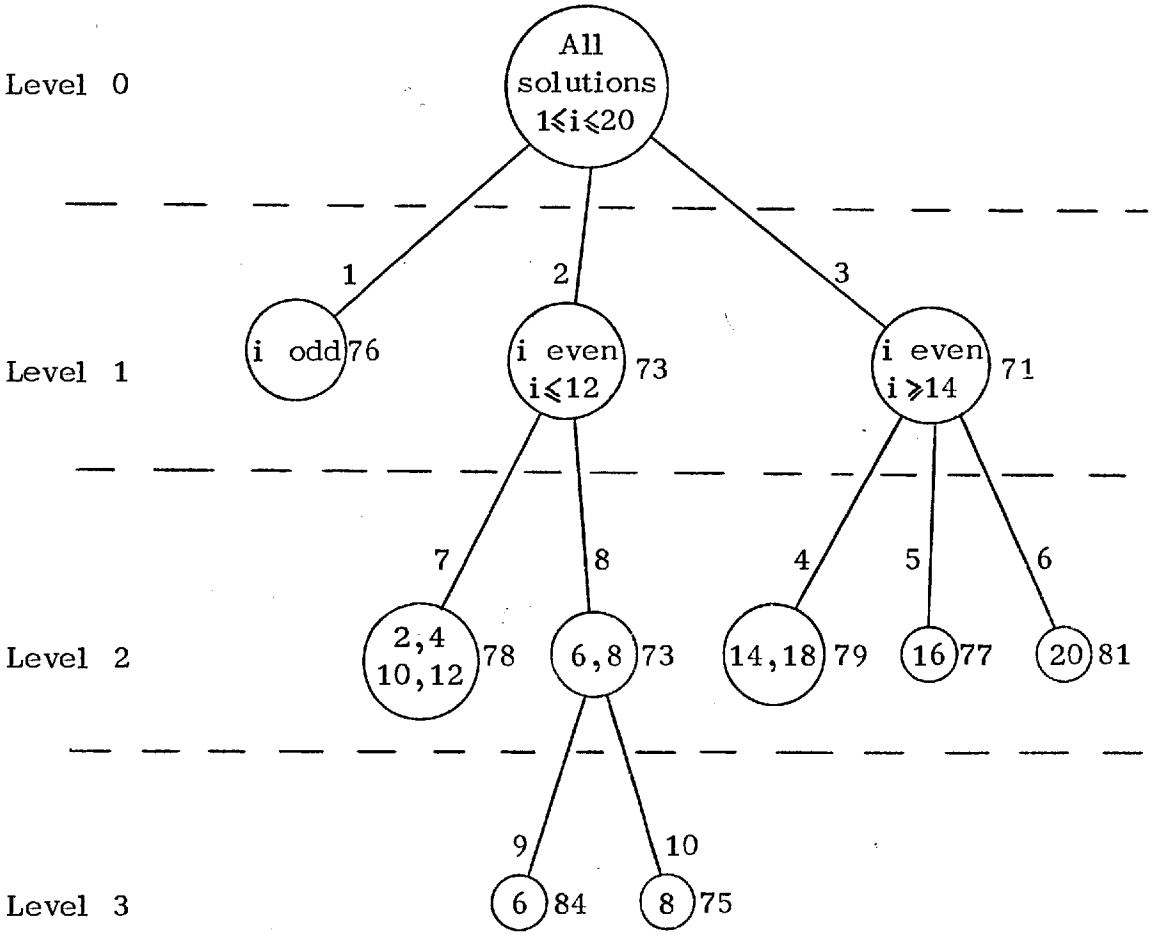


Figure 4-1.

Tree-search method for a problem with 20 feasible solutions. The number beside each node is its lower bound. The numbers on the branches show the order in which the nodes were generated.

and this eventually leads to the singleton node 10 whose cost is less than the bounds on all other pendant nodes. Hence the optimal solution is number 8 with cost 75.

There are two types of strategic decisions needed for any tree-search procedure. The first concerns branching: the rules for partitioning a subset and the order in which nodes are considered. The second decision is how the bounds are to be calculated for each node. Both decisions can have a profound effect on the computational efficiency of the method, but the quality of the bounds is usually the more important factor. In figure 4-1, for example, if the bound on node 1 had been 74 instead of 76, then many more nodes might have been generated. On the other hand, branching from node 1 before node 3 or 2 might also generate many more nodes.

4.2 BRANCHING STRATEGY

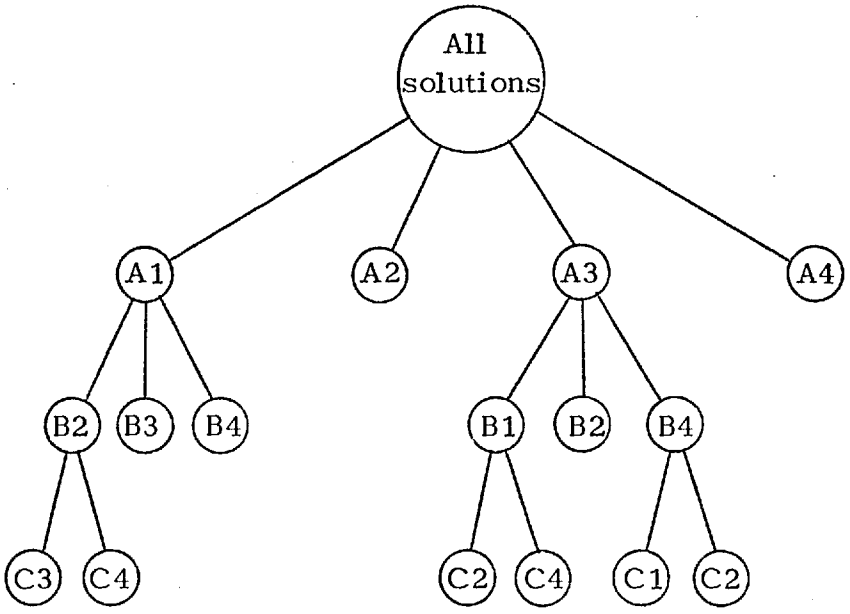
There are many decisions to be made concerning branching, which determines the nature of the tree. Into how many subsets should each node be partitioned? Should this be fixed or vary with the level of the node? Should there be any symmetry between subsets of the same node? In particular should all subsets at the same level of the tree contain the same number of solutions? Since it is not practical to enumerate the elements of a subset, how are they to be characterised? Node 1 of figure 4-1, for example, was characterised by its elements

being odd. Several authors (21,22,25,39) have partitioned each node at level k into n-k nodes at level k+1, with each branch being characterised by the assignment of a specific machine to a specific location. Thus a particular node at level 1 might consist of all solutions for which machine A is assigned to location 7, and a node at level 3 might consist of all solutions with A assigned to 7, B to 1 and C to 4. This branching policy is considered symmetric (see figure 4-2(a)).

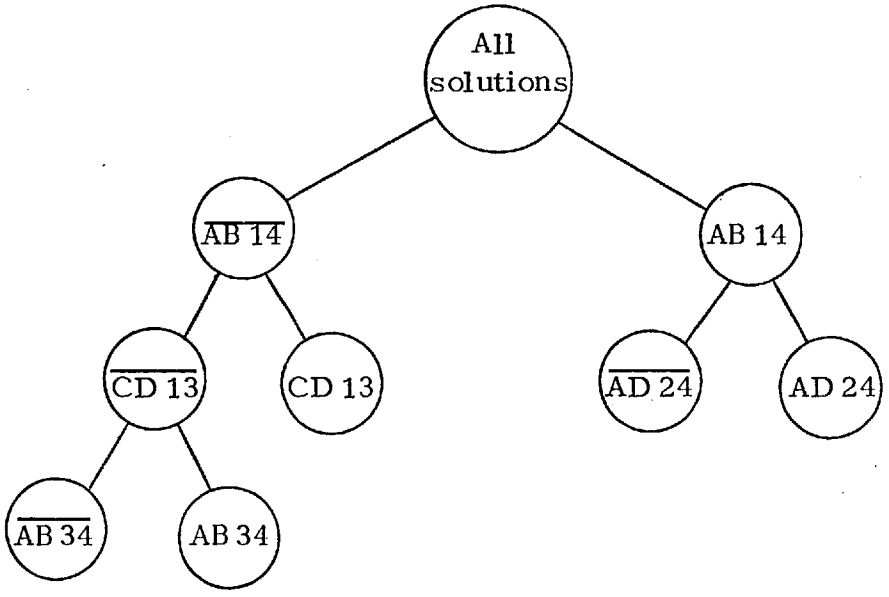
Other authors (20,38) adopted an asymmetric, binary branching policy for which each branch is characterised by the assignment or the non-assignment of a specific pair of machines to a specific pair of locations. Then a node at level 3 might consist of all solutions for which the machines A and B are assigned to locations 3 and 5 (in either way), E and G are not assigned to 1 and 4, and B and D are not assigned to 5 and 6. A possible tree for this strategy is shown in figure 4-2(b).

The branching used for this investigation is also binary and asymmetric, but is more closely related to the first strategy described above. A specific machine α and location i are chosen for each branching; the node is then partitioned so that all solutions which assign machine α to location i are in one subset (called an "inclusion" node), and all solutions which assign machine α to any other location are in the other subset (called an "exclusion" node).

The selection of the particular machine and location depends on the lower bounds used and will be explained in section 4.4 . In terms of quadratic programming, the solutions with $x_{\alpha i} = 1$ are in the inclusion



(a) A multi-level symmetric tree.



(b) A binary symmetric tree.

A horizontal bar indicates that the pair of machines is not assigned to the pair of locations.

Figure 4-2.

Typical trees resulting from different branching strategies.

node and those with $x_{\alpha i} = 0$ are in the exclusion node.

This partitioning rule determines the structure of the resulting tree, but does not state the order in which the nodes of the tree should be generated. The rule followed for figure 4-1 was to always branch from the pendant node with the smallest lower bound. This is sometimes known as a breadth-first search because all branches are explored simultaneously. It ensures that only the minimum number of nodes are generated to find and prove an optimal assignment. One of its disadvantages for computer applications is that a record must be kept of all pendant nodes and there may be many thousands of them.

The search rule used here is depth-first, for which one of the most recently generated nodes is always the next to be partitioned. Nodes are produced two at a time and so there is often a tie when deciding which node is most recent; in this case the inclusion node is given priority because it contains fewer solutions than the exclusion node and so is likely to produce fewer nodes below it before finding an optimal assignment or calculating bounds greater than a known assignment.

Figure 4-3 shows how the depth-first rule might explore a tree for a 4-machine problem. The first assignment found is (B3 A4 C1 D2). If the cost of this is less than the bounds on nodes 4 and 6 then all five nodes on the right of the tree can be discarded and only a note of the best assignment found so far need be kept. Next the branches below node 7 are explored. If the bounds on nodes 10, 11 and 12 are greater than

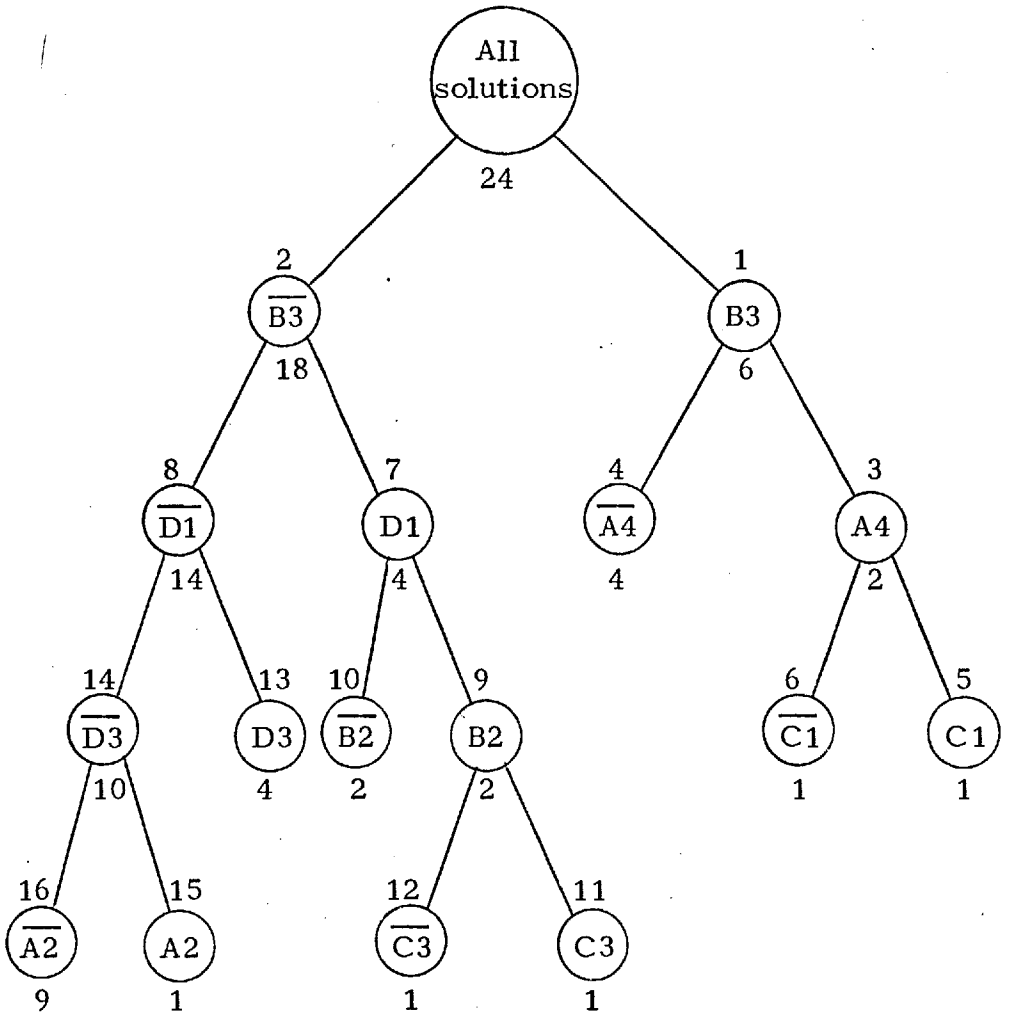


Figure 4-3.

A tree for a 4-machine problem.

The numbers above each node show the order of branching. The number of solutions in each node is given below each node.

the best known assignment, this new section is discarded; if not, the improved solution (node 11 or 12) replaces node 5 as the best assignment and the new section is still discarded. Thus only a small number of nodes need be recorded at any stage of the search.

4.3 THE BASIC BOUND

An excellent review by Pierce and Crowston (58) has shown that all the proposed tree-search algorithms for the quadratic assignment problem use only two fundamentally different bounds. This paucity is investigated in the following chapter.

The present algorithm uses a modification of the bound suggested independently by Lawler (39) and Gilmore (21). This bound is always greater than or equal to the other bound but requires considerably more computation. This extra calculation is considered justified because, as was stated in section 4.1, tree-search methods are critically dependent on the quality of the bounds.

First a lower bound on the overall problem (i.e. the node at level 0 containing all assignments) will be developed; this is exactly equivalent to Lawler's bound. Then it will be generalised to deal with all the nodes involving both the inclusion and exclusion of machine-location pairs. This is necessarily different from Lawler's bound because of the different branching strategy.

Let ρ, ρ' and ρ'' represent assignments. Then, by definition,

for any assignment ρ,

$$Z(\rho) = \sum_{\alpha} \left(c_{\alpha\rho(\alpha)} + \sum_{\beta>\alpha} f_{\alpha\beta}\, d_{\rho(\alpha)\rho(\beta)} \right)$$

$$= \sum_{\alpha} \left(c_{\alpha\rho(\alpha)} + \tfrac{1}{2} \sum_{\beta\neq\alpha} f_{\alpha\beta}\, d_{\rho(\alpha)\rho(\beta)} \right)$$

$$\geqslant \sum_{\alpha} \left(c_{\alpha\rho(\alpha)} + \tfrac{1}{2} \min_{\substack{\rho'' \\ \rho''(\alpha)=\rho(\alpha)}} \left(\sum_{\beta\neq\alpha} f_{\alpha\beta}\, d_{\rho(\alpha)\rho''(\beta)} \right) \right)$$

$$\geqslant \min_{\rho'} \left[\sum_{\alpha} \left(c_{\alpha\rho'(\alpha)} + \tfrac{1}{2} \min_{\substack{\rho'' \\ \rho''(\alpha)=\rho'(\alpha)}} \left(\sum_{\beta\neq\alpha} f_{\alpha\beta}\, d_{\rho'(\alpha)\rho''(\beta)} \right) \right) \right] \qquad 4\text{-}1$$

This last expression is independent of ρ and is clearly a lower bound on the cost of all assignments; it will be denoted by L and referred to as the basic bound. It is not at all obvious that L can be easily calculated and some notation must be introduced to explain how it can be done.

Let $$a'_{\alpha i} = \min_{\substack{\rho \\ \rho(\alpha)=i}} \left(\sum_{\beta\neq\alpha} f_{\alpha\beta}\, d_{i\rho(\beta)} \right) \qquad 4\text{-}2$$

and $$a_{\alpha i} = c_{\alpha i} + \tfrac{1}{2} a'_{\alpha i} \,. \qquad 4\text{-}3$$

Now $L = \min_{\rho} \left(\sum_{\alpha} a_{\alpha\rho(\alpha)} \right)$ and this minimisation is simply a linear assignment problem of dimension n x n. It can be solved efficiently using the Hungarian algorithm for example (36,70).

The only remaining computational difficulty is calculating $a'_{\alpha i}$ for n^2 values of (α, i) according to formula 4-2. This minimisation is also a linear assignment problem, this time of dimension (n-1)x(n-1)

since $\rho(\alpha)=i$ is fixed.

Physically, $\sum_{\beta \neq \alpha} f_{\alpha\beta} d_{i\rho(\beta)}$ is the cost of flows to and from machine α in location i; $a'_{\alpha i}$ is the minimum value of this cost when all possible assignments of the other n-1 machines are considered. Several authors (e.g. Gilmore (21)) have shown that this minimum is achieved when the machine with the smallest of the n-1 flows from machine α is assigned to the location furthest from location i, the second-to-smallest flow to the second-to-largest distance and so on.

Thus $a'_{\alpha i}$ can be calculated very easily. Simply rank the n-1 flows from machine α in increasing order, the n-1 distances from location i in decreasing order, and then sum the products of corresponding numbers. This ranking procedure is much faster than solving a linear assignment problem, even the definition of which requires $(n-1)^2$ multiplications.

The flow-chart in figure 4-4 summarises the procedure for calculating the basic bound on all the solutions of the quadratic assignment problem. Note that only 2n sets of n-1 numbers need be ranked to find the n^2 values $a'_{\alpha i}$.

The Bound for General Nodes

A tree-search algorithm does not, strictly speaking, require a bound for the level 0 nodes, but it was convenient to explain it in detail for that simplest case before generalising.

A node is characterised by the assignments it includes and those it

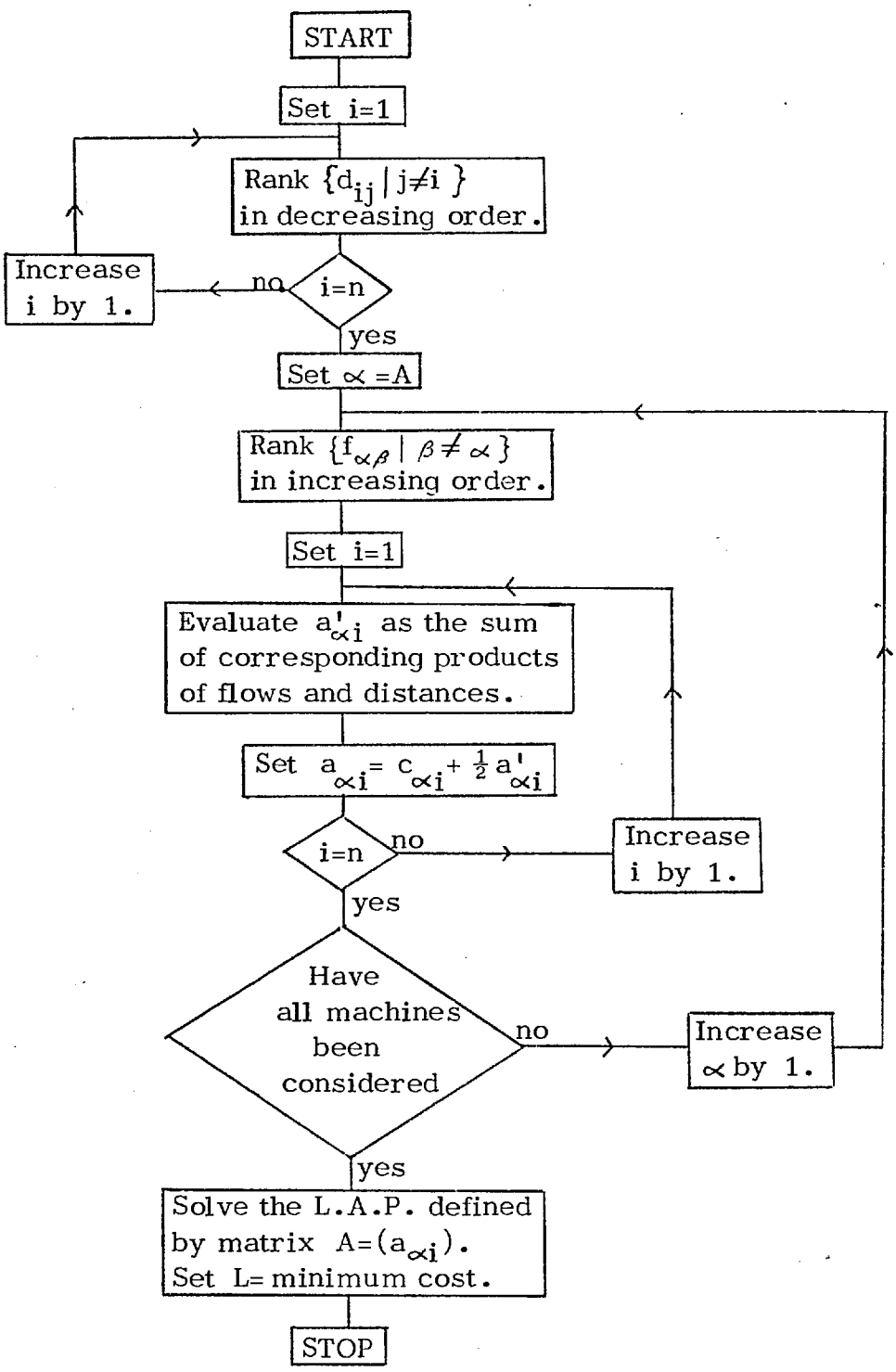


Figure 4-4.

Flow-chart to calculate L, the basic bound on all solutions.

excludes. For any node let

$I = \{(\alpha, i) \mid$ machine α must be assigned to location $i\}$ and

$E = \{(\alpha, i) \mid$ machine α must not be assigned to location $i\}$.

I and E are called the inclusion and exclusion sets respectively. The number of machine-location pairs in the inclusion set is $|I|$. Standard set notation will be extended so that $\alpha \in I$ means that $(\alpha, i) \in I$ for some location i, etc.

For any assignment, ρ, which satisfies the inclusion and exclusion sets I and E:

$$Z(\rho) = \sum_{\alpha} \left(c_{\alpha\rho(\alpha)} + \tfrac{1}{2} \sum_{\beta \neq \alpha} f_{\alpha\beta} d_{\rho(\alpha)\rho(\beta)} \right)$$

$$= \sum_{\alpha \in I} \left(c_{\alpha\rho(\alpha)} + \tfrac{1}{2} \sum_{\substack{\beta \neq \alpha \\ \beta \in I}} f_{\alpha\beta} d_{\rho(\alpha)\rho(\beta)} \right) +$$

$$+ \tfrac{1}{2} \sum_{\alpha \in I} \sum_{\beta \notin I} f_{\alpha\beta} d_{\rho(\alpha)\rho(\beta)} + \tfrac{1}{2} \sum_{\alpha \notin I} \sum_{\beta \in I} f_{\alpha\beta} d_{\rho(\alpha)\rho(\beta)} + \sum_{\alpha \in I} c_{\alpha\rho(\alpha)}$$

$$+ \tfrac{1}{2} \sum_{\alpha \notin I} \sum_{\substack{\beta \notin I \\ \beta \neq \alpha}} f_{\alpha\beta} d_{\rho(\alpha)\rho(\beta)}$$

$$= g(I) + \sum_{\alpha \notin I} C(I)_{\alpha\rho(\alpha)} + \tfrac{1}{2} \sum_{\alpha \notin I} \sum_{\substack{\beta \notin I \\ \beta \neq \alpha}} f_{\alpha\beta} d_{\rho(\alpha)\rho(\beta)} \qquad 4\text{-}4$$

where $g(I) = \sum_{\alpha \in I} \left(c_{\alpha\rho(\alpha)} + \tfrac{1}{2} \sum_{\substack{\beta \neq \alpha \\ \beta \in I}} f_{\alpha\beta} d_{\rho(\alpha)\rho(\beta)} \right)$ 4-5

and $C(I)_{\alpha i} = c_{\alpha i} + \sum_{\beta \in I} f_{\alpha\beta} d_{i\rho(\beta)}$. 4-6

The expression 4-4 is exactly in the form of a constant, $g(I)$, plus a quadratic assignment problem involving $(n - |I|)$ machines and

locations. Note that formulae 4-5 and 4-6 are well-defined because $\rho(\alpha)$ is known for $\alpha \in I$.

The restrictions of the inclusion set are automatically enforced by formulae 4-5 and 4-6, but the exclusion set has been ignored, This can be rectified by defining :

$$C(I,E)_{\alpha i} = \begin{cases} \infty \text{ if } (\alpha, i) \in E \\ C(I)_{\alpha i} \text{ if } (\alpha, i) \notin E \end{cases} \qquad 4\text{-}7$$

To summarise, a node with inclusion and exclusion sets I and E can be used to define a new quadratic assignment problem which will be called P(I,E). Its machines and locations are those not present in I, and its objective function is

$$Z(\rho,I,E) = \sum_{\alpha} C(I,E)_{\alpha\rho(\alpha)} + \sum_{\alpha} \sum_{\beta>\alpha} f_{\alpha\beta} d_{\rho(\alpha)\rho(\beta)}$$

where all summations exclude machines in I. If the basic bound on P(I,E) is L(P(I,E)) then the basic bound on the node (I,E) is $L(I,E) = g(I) + L(P(I,E))$ where g(I) is defined by 4-5.

4.4 PROGRAM "LOCATE"

A FORTRAN program called LOCATE was written to implement the tree-search algorithm. Figure 4-5 gives a general flow-chart of the program's logic. Before discussing some features of the program which are omitted from this flow-chart, there are two steps included in figure 4-5 which need explanation.

One point is the deletion of a node "and all its dependents".

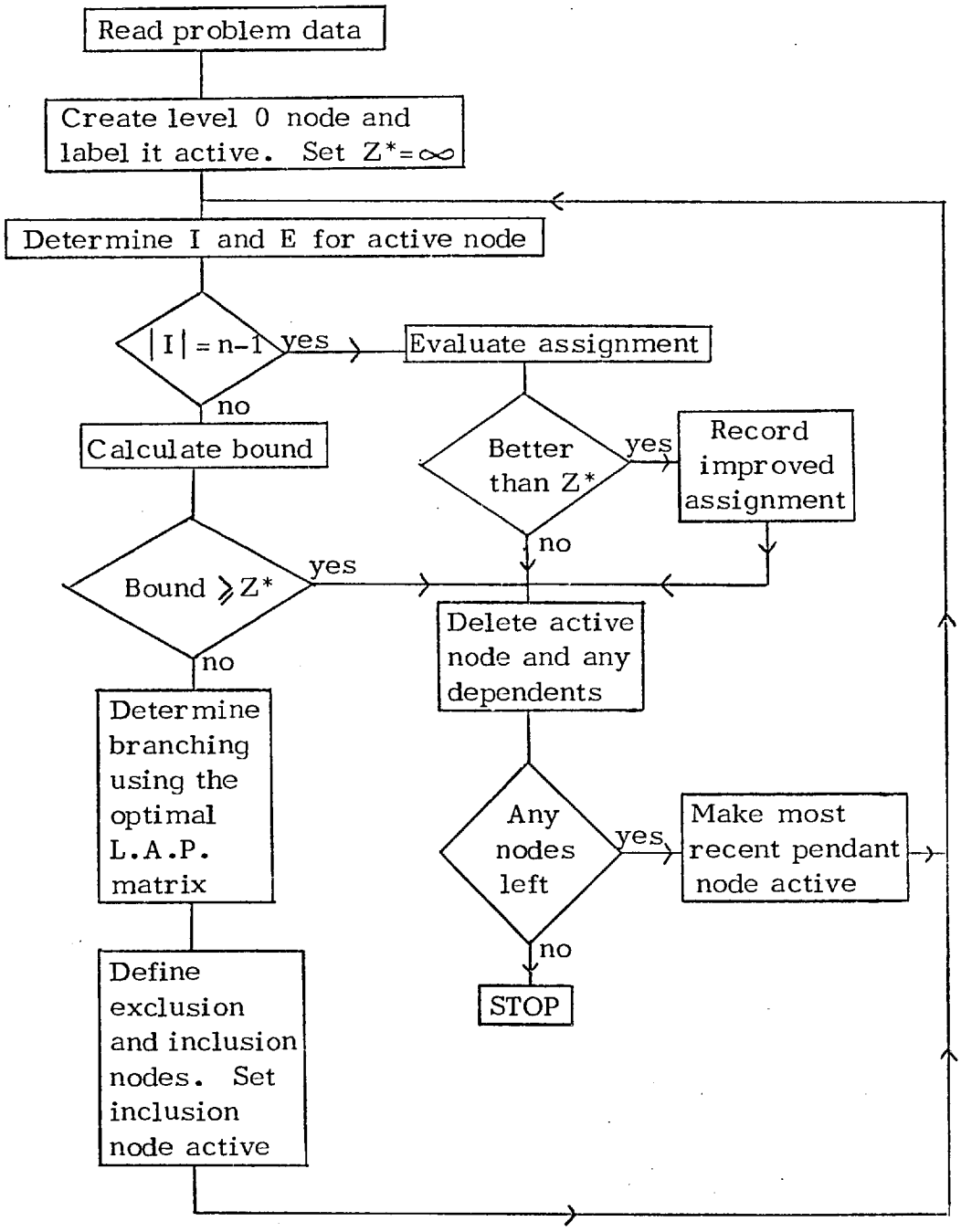


Figure 4-5

Overall flow-chart of program LOCATE.

Basically, a dependent node is one which relies on the active node to justify its existence. The tree in figure 4-6, for example, has two dependent nodes, 6 and 7. If active node 8 is to be deleted then 6 and 7 are also deleted and node 5 becomes active; unless an improved assignment has been found recently such that the bound on node 4 exceeds Z^*, and then nodes 5, 4, 3 and 2 would also be dependents and would be deleted, making node 1 active.

A more fundamental vagueness in figure 4-5 concerns the rule for branching. The objective is to branch so that the cheapest assignment from the active node is included in the inclusion node. This is equivalent to saying that the bound on the new exclusion node should be as large as possible. As implied on the flow-chart, it is possible to estimate this bound without extensive calculation by inspecting the matrix left after solution of the linear assignment problem for the bound on the active node.

Figure 4-7 demonstrates how branching is done for a node of a 6-machine problem. The Hungarian algorithm is applied to the matrix A and it subtracts constants from the rows and columns of the matrix until a "zero-cost" assignment is found: for matrix A' the assignment (A3 C4 D2 F6) is zero-cost. Consider the result of excluding any machine-location pair from this assignment. If D2 were excluded, for example, the best location for D would be 4 and the best machine for 2 would be C: The cost of the assignment would therefore be at least 1+3=4 greater than the original cost. The value 4 is called the "penalty" for excluding D2.

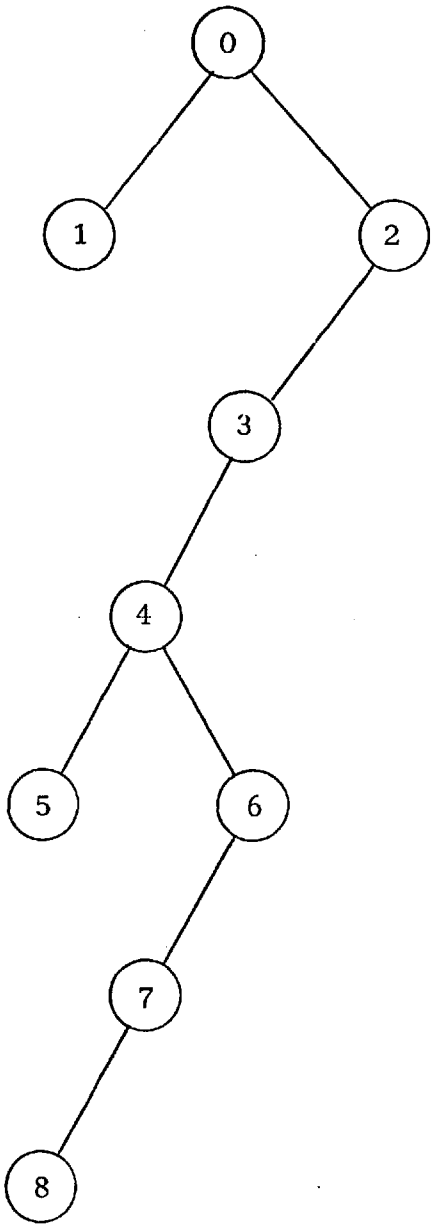


Figure 4-6.

A partially enumerated tree showing dependent nodes.

$I = \{(B,5),(E,1)\}$, $E = \{(C,3),(C,6),(D,5),(E,3),(F,2)\}$,
$g(I) = 28$.

Original matrix, $A =$

	2	3	4	6
A	16	10	25	21
C	14	∞	16	∞
D	9	19	15	13
F	∞	12	14	11

After solving L.A.P. $A' =$

	2	3	4	6
A	7	0	11	10
C	3	∞	0	∞
D	0	9	1	2
F	∞	2	0	0

$L(I,E) = 28 + 46 = 74$

The critical assignment is $(A,3)$ with penalty $7+2=9$.

If $(A,3)$ is excluded and the L.A.P. re-minimised,

$A'' =$

	2	3	4	6
A	0	∞	4	1
C	3	∞	0	∞
D	0	5	1	0
F	∞	0	2	0

The bound on the new exclusion node is

$$L(I,E\cup\{(A,3)\}) = L(I,E) + 11 = 85.$$

The next critical assignment is $(F,3)$ with penalty 5+0=5 etc.

Figure 4-7.

Example of penalty calculations for branching.

Penalties can be calculated for each of the four machine-location pairs and the largest penalty determines the next branching. In this case the maximum penalty is 9 for A3. The actual bound for the new exclusion node can be calculated by setting the A3 matrix element to infinity and re-minimising with the Hungarian algorithm.

Note how easy it is to calculate the bound for an exclusion node when the bound for the active node above it has just been determined. This process can be extended to calculate a long string of exclusion node bounds by making minor changes to the matrix. Program LOCATE does this automatically.

Interactive Use of Program

No special effort was made to code LOCATE very efficiently because flexibility was considered more important, flexibility to allow different search strategies and different bounds to be incorporated without extensive reprogramming. In keeping with this experimental philosophy the program can be run interactively from a computer terminal so that the user can both watch and direct the search.

The user can specify how often he wants to interrupt the program: e.g. after a specified number of nodes have been generated or when an improved solution is found. Having interrupted, he may change the active node or delete some nodes before continuing. Or he may be content to simply print out the current best assignment and then terminate the search. A complete guide to interactive use of LOCATE can be found

in appendix D with an example.

4.5 RESULTS FOR THE BASIC BOUND

The three small problems considered in chapter 3 were solved easily using LOCATE. Statistics for these and seven other problems are given in tables 4-1 and 4-2. Data for all problems are given in appendix A. The tree created for LA7 is given as an example in figure 4-8.

Two trends are very noticeable in table 4-1. Firstly, the computation time increases very rapidly as the number of machines increases. Very roughly it appears that each additional machine doubles the number of nodes generated and almost trebles the time required.

The second obvious trend is that computation time is highly dependent on the data used as well as the number of machines. In particular it seems that problems with all $c_{\propto i}$ equal to zero are considerably more difficult than problems with random positive $c_{\propto i}$. The NVR problems have zero $c_{\propto i}$ while for the MG problems the $c_{\propto i}$ are random, positive, with mean and variance chosen so that the linear and quadratic components of the objective function have approximately equal weight.

To investigate this phenomenon directly, two new problems were formed from MG10 and NVR12; problem MG10Z is MG10 with $c_{\propto i} = 0$ and NVR12P is NVR12 with $c_{\propto i}$ equal to random positive numbers.

Problem	Number of Nodes	Time (secs)
GP4	9	0.069
LA7	19	0.356
NVR8	316	5.421
TSP10	354	9.370
MG10	214	9.479
MG12	500	32.752
MG10Z	1398	52.109
NVR12P	2344	113.186
NVR12	11148	481.678
MG14	12586	1115.978

Table 4-1.

Basic statistics for LOCATE with ten problems.

Problem	First Solution	Optimum	Number of Nodes before optimum	% of Nodes before optimum
GP4	403	403	6	66.7
LA7	559	559	12	32.4
NVR8	107	107	14	4.4
TSP10	23	22	53	15.0
MG10	1092	1092	18	8.4
MG12	1241	1241	22	4.4
MG10Z	965	915	846	60.5
NVR12P	489	470	89	3.8
NVR12	293	289	397	3.5
MG14	1886	1873	147	1.2

Table 4-2.

Analysis of when the optimal assignment is found.

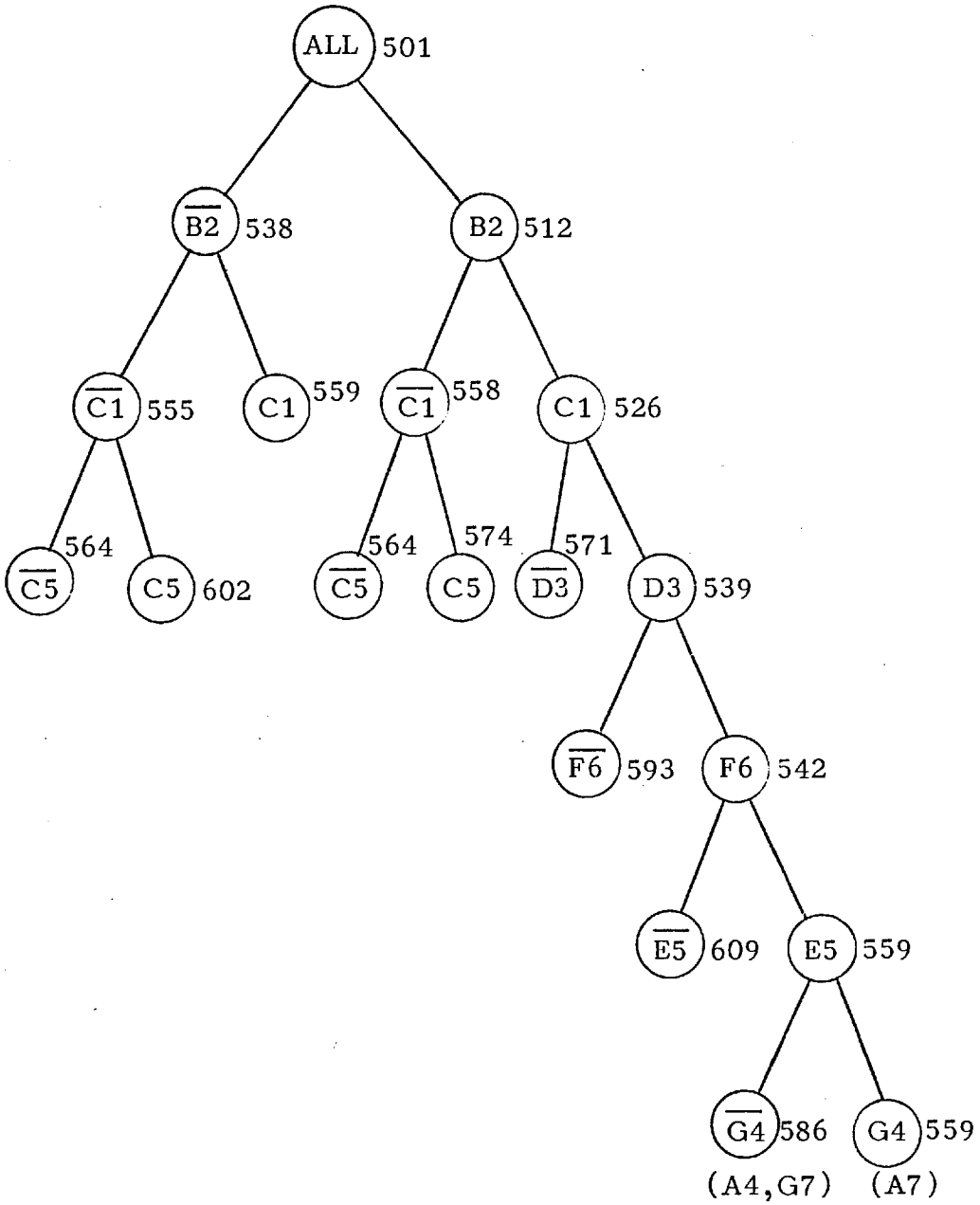


Figure 4-8.

Tree for a 7-machine problem, LA7.

(It is largely coincidence that C1 appears twice on level 2; similarly C5 on level 3.)

In both cases the problem with positive $c_{\propto i}$ was solved approximately five times faster than the same problem with zero $c_{\propto i}$.

But the data-dependence of the algorithm's efficiency cannot be properly described in terms of $c_{\propto i}$ alone. The travelling salesman problem, TSP10, has zero $c_{\propto i}$ but was actually solved slightly faster than MG10.

The transformation due to Burkard described in section 2.3 can introduce positive $c_{\propto i}$, but experimentation has shown that it does not improve the efficiency of the tree-search. Applying the tree-search algorithm to the transformed problem, which has smaller flows and distances but larger linear costs, proved just as likely to increase computation time as decrease it.

Table 4-2 shows that the first assignment found by LOCATE is often optimal. If it is not, then the first assignment is usually very good and the optimal assignment is found near the beginning of the search. This situation is typical of depth-first searches (41).

Table 4-3 and figure 4-9 give an analysis of the computing time used for each phase of the program. The smallest problems are untypical, but the larger ones are quite consistent. Over 80% of the computing time is spent calculating bounds for nodes while they are active and this time is divided roughly equally between calculating the elements of matrix A (using formulae 4-2 and 4-3), and solving the linear assignment problem for this matrix.

Problem	Calculating matrix A	L. A. P. for : Active node	Exclusion node	Overheads & branching	Total
GP4	0.013	0.013	0.006	0.037	0.069
LA7	0.096	0.083	0.057	0.120	0.356
NVR8	2.777	1.335	0.505	0.804	5.421
TSP10	5.848	1.472	0.642	1.408	9.370
MG10	3.798	4.188	0.697	0.796	9.479
MG12	14.710	13.429	2.158	2.455	32.752
MG10Z	21.751	22.586	3.205	4.567	52.109
NVR12P	62.723	30.779	8.960	10.721	113.183

Table 4-3.

Time (seconds) used by each section of program LOCATE.

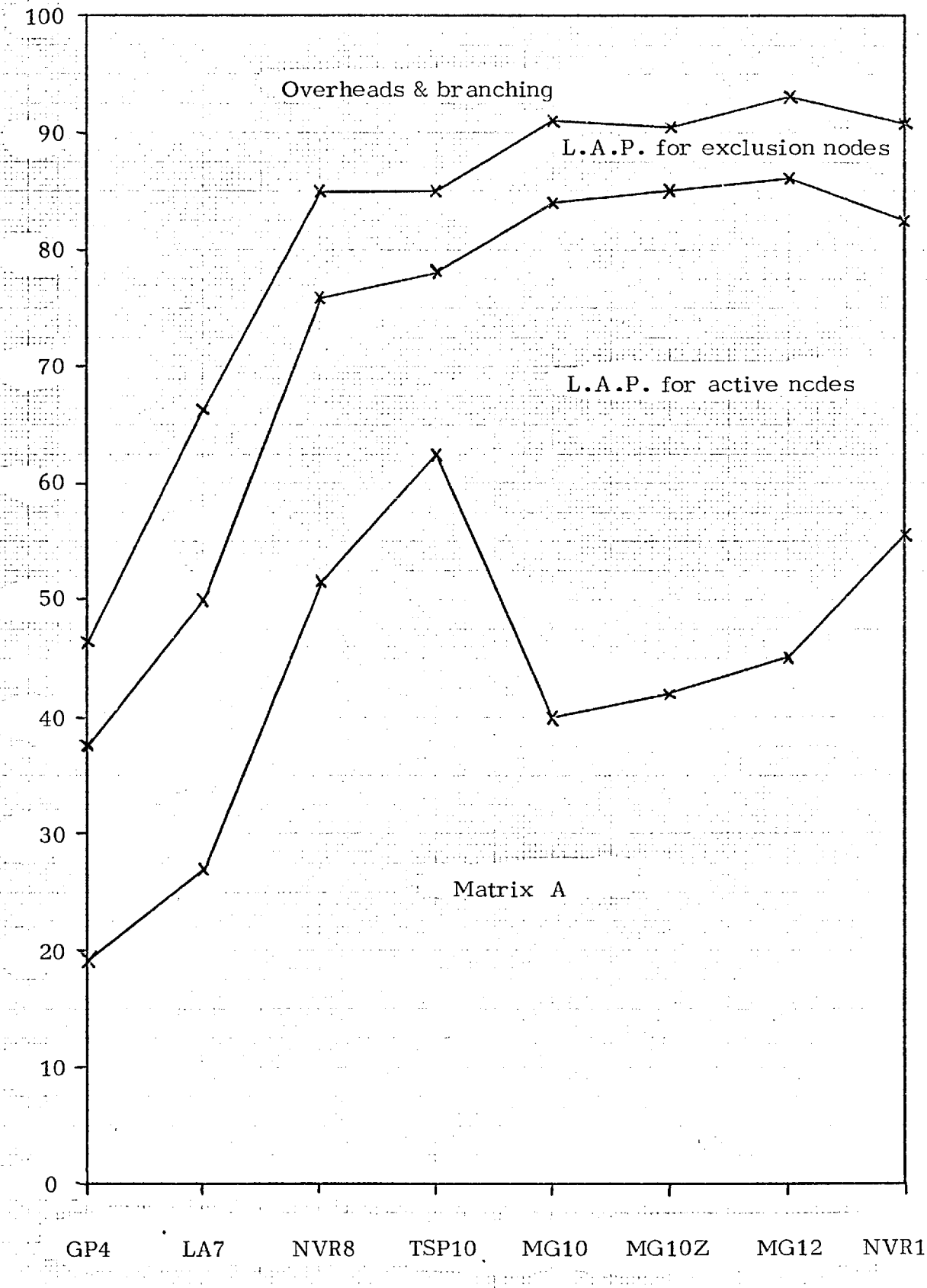


Figure 4-9.

Percentage of time used in each section of program LOCATE.

These two phases must be the target for any improvement in the coding of the tree-search algorithm, as opposed to improvement of the algorithm itself. A small improvement in calculating the matrix elements was achieved by ranking the flows and distances from each machine and location only once, before any nodes are created. Note the efficiency with which bounds for exclusion nodes are calculated by modifying the optimal matrix for the active node and then re-minimising.

4.6 MODIFIED BOUNDS

Figure 4-9 shows that approximately 40% of computation time is used to solve linear assignment problems, the solutions to which give lower bounds for nodes of the tree. But it is well-known that a good lower bound on the solution to a linear assignment problem can be found by subtracting constants from each row and column of the matrix until each row and column contains a zero but no negative elements; the bound is simply the sum of the constants used. This bound is also a bound on the quadratic assignment problem.

The tree-search program was modified to use this simpler lower bound, which is clearly weaker than the basic bound and so the number of nodes generated by the modified algorithm is greater than before. But on the other hand, the time needed to analyse each node is reduced by nearly 40%. Statistics for this weaker bound are given in table 4-4; they show a definite trend in favour of using the basic bound.

Problem	Number of Nodes	Time (secs)
GP4	12	0.059
LA7	34	0.488
NVR8	523	6.388
TSP10	494	9.790
MG10	558	12.729
MG12	1375	48.866
MG10Z	2651	55.753
NVR12P	4906	151.282

Table 4-4.

Statistics for LOCATE with weakened bounds. (c.f. table 4-1)

A Bound Directly Incorporating Linear Costs

Consider the physical interpretation of the formulae 4-2 and 4-3 in section 4.3.

$$\text{i.e.} \quad a'_{\alpha i} = \min_{\substack{\rho \\ \rho(\alpha)=i}} \left(\sum_{\beta \neq \alpha} f_{\alpha\beta} d_{i\rho(\beta)} \right) \tag{4-2}$$

$$\text{and} \quad a_{\alpha i} = c_{\alpha i} + \tfrac{1}{2} a'_{\alpha i} . \tag{4-3}$$

The first expression is a lower bound on the cost of flows to and from machine α if it is placed at location i. Since $c_{\alpha i}$ is the linear cost incurred by such placement, $a_{\alpha i}$ is a lower bound on the total cost of the placement.

But 4-2 completely ignores some possible effects of the linear costs. Suppose, for example, that $(\alpha, i)=(A,1)$, f_{AB} is the smallest flow from A and d_{12} is the greatest distance from 1. Then formula 4-2 automatically places machine B at location 2, even though c_{B2} may have a very large value. The subsequent inclusion of c_{B2} in the formula for a_{B2} does not remedy the low value calculated for a_{A1}.

Remembering that there are n-1 machines interacting with machine A, it can be said that the basic bound associates the quadratic cost $f_{AB}d_{12}$ with the linear cost $c_{A1}/(n-1)$ when a'_{A1} is being calculated, but with $c_{B2}/(n-1)$ when a'_{B2} is being calculated. A more rational scheme would use $(c_{A1}+c_{B2})/2(n-1)$ in both cases. The derivation of the lower bound in section 4.3 can now be modified with this in mind.

For any assignment ρ,

$$Z(\rho) = \sum_{\alpha} \left(c_{\alpha\rho(\alpha)} + \tfrac{1}{2} \sum_{\beta \neq \alpha} f_{\alpha\beta} d_{\rho(\alpha)\rho(\beta)} \right)$$

$$= \sum_{\alpha} \left(\tfrac{1}{2} c_{\alpha\rho(\alpha)} + \tfrac{1}{2} \sum_{\beta \neq \alpha} \left(f_{\alpha\beta} d_{\rho(\alpha)\rho(\beta)} + \frac{1}{n-1} c_{\beta\rho(\beta)} \right) \right)$$

$$\geqslant \sum_{\alpha} \left(\tfrac{1}{2} c_{\alpha\rho(\alpha)} + \tfrac{1}{2} \min_{\substack{\rho'' \\ \rho''(\alpha)=\rho(\alpha)}} \left[\sum_{\beta \neq \alpha} \left(f_{\alpha\beta} d_{\rho(\alpha)\rho''(\beta)} + \frac{1}{n-1} c_{\beta\rho''(\beta)} \right) \right] \right)$$

$$\geqslant \min_{\rho'} \left\{ \sum_{\alpha} \left(\tfrac{1}{2} c_{\alpha\rho'(\alpha)} + \tfrac{1}{2} \min_{\substack{\rho'' \\ \rho''(\alpha)=\rho'(\alpha)}} \left[\sum_{\beta \neq \alpha} \left(f_{\alpha\beta} d_{\rho'(\alpha)\rho''(\beta)} + \frac{1}{n-1} c_{\beta\rho''(\beta)} \right) \right] \right) \right\}$$

Define $$a'_{\alpha i} = \min_{\substack{\rho \\ \rho(\alpha)=i}} \left[\sum_{\beta \neq \alpha} \left(f_{\alpha\beta} d_{i\rho(\beta)} + \frac{1}{n-1} c_{\beta\rho(\beta)} \right) \right] \qquad 4\text{-}8$$

and $$a_{\alpha i} = \tfrac{1}{2} c_{\alpha i} + \tfrac{1}{2} a'_{\alpha i} . \qquad 4\text{-}9$$

Now, as before, $L = \min_{\rho} \left(\sum_{\alpha} a_{\alpha\rho(\alpha)} \right)$ is a lower bound on the quadratic assignment problem.

This bound may be expected to prove better than the basic bound. It is especially important for nodes far down the tree because each level of the tree increases the linear costs - inclusion nodes by adding flow-distance products and exclusion nodes by introducing infinite costs.

The minimisation in formula 4-8 cannot, unfortunately, be solved with the simple ranking procedure used for formula 4-2. It is a linear assignment problem of dimension n-1 and could be solved by a general method such as the Hungarian algorithm. This would have to be solved for each of the n^2 elements of matrix A and the computation

required was considered too great to incorporate into the tree-search program.

A Bound using Restricted Ranking

The argument used to derive formulae 4-8 and 4-9 can be directly generalised so that the coefficient of $c_{\alpha i}$ is t rather than $\frac{1}{2}$, where the parameter t may take any value between 0 and 1 inclusive. The formulae then become:

$$a'_{\alpha i} = \min_{\substack{\rho \\ \rho(\alpha)=i}} \left[\sum_{\beta \neq \alpha} \left(f_{\alpha\beta} d_{i\rho(\beta)} + \frac{2(1-t)}{n-1} c_{\beta\rho(\beta)} \right) \right] \qquad 4\text{-}10$$

and $$a_{\alpha i} = t c_{\alpha i} + \tfrac{1}{2} a'_{\alpha i} . \qquad 4\text{-}11$$

Suppose these formulae are being applied to a node with an exclusion set $E \neq 0$; i.e. there is at least one machine-location pair (β, j) for which $c_{\beta j} = \infty$ and so $\rho(\beta)$ cannot be j. The notation $\rho \sim E$ will be taken to mean that assignment ρ satisfies all the exclusions of the set E. Then, for any t between zero and one, it is clear that 4-10 can be re-written as:

$$a'_{\alpha i} = \min_{\substack{\rho \\ \rho(\alpha)=i \\ \rho \sim E}} \left[\sum_{\beta \neq \alpha} \left(f_{\alpha\beta} d_{i\rho(\beta)} + \frac{2(1-t)}{n-1} c_{\beta\rho(\beta)} \right) \right] .$$

Taking the limit $t \rightarrow 1$, the equations revert to almost their original form:

$$a'_{\alpha i} = \min_{\substack{\rho \\ \rho(\alpha)=i \\ \rho \sim E}} \left(\sum_{\beta \neq \alpha} f_{\alpha\beta} d_{i\rho(\beta)} \right) \qquad 4\text{-}12$$

and $$a_{\alpha i} = c_{\alpha i} + \tfrac{1}{2} a'_{\alpha i} . \qquad 4\text{-}13$$

This bound is similar to the previous one in that it tries to take proper account of linear costs. It is weaker than the previous one because it only considers the infinite linear costs, but stronger than the basic bound.

There is some difficulty in evaluating 4-12. Simple ranking is insufficient and the standard algorithms for the linear assignment problem are too slow, as for the previous bound. The following section develops an efficient algorithm to evaluate 4-12.

Before leaving this section it is worthwhile comparing the bounds produced by all the methods which have been discussed, including the weaker of Gilmore's bounds described in section 2.7 . This is done in table 4-5 which gives the bounds for two problems, at both level 0 and a node further down the tree. The cost of the true minimal assignment is also given.

4.7 RESTRICTED RANKING ALGORITHM

This section modifies the Hungarian algorithm to solve the linear assignment problem defined by formula 4-12 in a way which takes advantage of the ranking procedure and does not actually generate the complete matrix.

The subscripts α and i are irrelevent and would be a nuisance and so the entire notation will be changed for this section only. The problem to be solved can be written:

Bound	GP4		NVR12	
	All Solutions	I={A4} E={D2,C2}	All Solutions	I={C1,I2,K6} E={A3,B3,B4,E12,L3}
True minimum	403	479	289	310
With linear costs	399	470	258	289
Restricted ranking	396	470	247	274
Basic bound	396	462	247	273
Gilmore's weak bound	389	462	243	258
Weaker bound	369	462	243	258

Table 4-5.

Direct comparison of six different lower bounds.

$$\min_{\rho \sim E} V(\rho) = \sum_{i=1,m} p_i q_{\rho(i)} \qquad 4\text{-}14$$

where $p_1 \leqslant p_2 \leqslant \cdots \leqslant p_m$, $q_1 \geqslant q_2 \geqslant \cdots \geqslant q_m$, ρ is a permutation of $1,2, \ldots ,m$ satisfying the exclusion set E, and $E = \{(i,j) \mid \rho(i) \text{ cannot equal } j\}$.

This is a linear assignment problem and therefore, according to section 3.1, can be put in linear programming form:

$$\min \; V(X) = \sum_{ij} p_i q_j x_{ij}$$

$$\sum_j x_{ij} = 1 \qquad \text{for } i=1,m$$

$$\sum_i x_{ij} = 1 \qquad \text{for } j=1,m$$

$$x_{ij} \geqslant 0 \qquad \text{for } i,j = 1,m$$

$$x_{ij} = 0 \qquad \text{for } (i,j) \in E .$$

As for all linear programs there is an equivalent dual formulation (see (19) for example) and for this program it has the following form:

$$\max \; W(\underset{\sim}{\mu},\underset{\sim}{\nu}) = \sum_i \mu_i + \sum_j \nu_j$$

$$\mu_i + \nu_j \leqslant p_i q_j \qquad \text{for } (i,j) \notin E$$

$$\mu_i, \; \nu_j \text{ unrestricted in sign} \qquad \text{for } i,j = 1,m .$$

Optimal solutions to the two problems are closely related and in particular $\min V(X) = \max W(\underset{\sim}{\mu},\underset{\sim}{\nu})$. Also, if $(\underset{\sim}{\mu},\underset{\sim}{\nu})$ is an optimal solution to the dual then there is an optimal solution to the primal such that $x_{ij} = 0$ for all pairs $(i,j) \notin E$ for which $p_i q_j - \mu_i - \nu_j > 0$.

It is now apparent that the Hungarian algorithm is simply a

way of solving the dual problem directly. The amount subtracted from row i of matrix $(p_i q_j)$ is μ_i and the amount subtracted from column j is ν_j. The maximum total amount subtracted is equal to the minimum assignment cost. The (i,j) element of the reduced matrix is $p_i q_j - \mu_i - \nu_j$ and if this is positive then $x_{ij} = 0$; i.e. $\rho(i) \neq j$.

If no assignments are excluded ($E = \emptyset$), then the ranking procedure gives the optimal assignment as being $\rho(i) = i$, or $x_{ii} = 1$ for $i = 1, m$. Duality theory states that $p_i q_j - \mu_i - \nu_i$, the diagonal elements of the reduced matrix, must all be zero; this is also obvious when interpreted in terms of the Hungarian algorithm. The following formulae for $\underset{\sim}{\mu}$ and $\underset{\sim}{\nu}$ satisfy these duality relations:

$$\begin{aligned} \nu_1 &= 0 \\ \mu_i &= p_i q_i - \nu_i \qquad \text{for } i = 1, m \\ \nu_{i+1} &= p_i q_{i+1} - \mu_i \qquad \text{for } i = 1, m-1. \end{aligned} \tag{4-15}$$

It can be easily shown that $p_i q_j - \mu_i - \nu_j \geqslant 0$ for this choice of $(\underset{\sim}{\mu}, \underset{\sim}{\nu})$ and that the expression is zero when i=j.

The above results give an explicit formula for the optimal reduced matrix of the linear assignment problem, 4-14, when $E = \emptyset$ - namely that the element (i,j) is $(p_i q_j - \mu_i - \nu_j)$ where $\underset{\sim}{\mu}$ and $\underset{\sim}{\nu}$ are defined by 4-15. Starting from this reduced matrix and the assignment $\rho(i) = i$, it is possible to introduce the exclusion constraints of E one at a time, re-optimising the reduced matrix according to the Hungarian algorithm after each new constraint is added.

The constraints of E may interact with the reduced matrix in any of three ways. First, if a constraint is not on the diagonal it will not affect the assignment at all and can be ignored, at least initially.

The second effect arises when one of the constraints is on the diagonal - i.e. $(i,i) \in E$ for some value of i. Then the value of the relevent matrix element is effectively infinite and so the Hungarian algorithm must be invoked to create another zero in the reduced matrix to find an optimal assignment taking account of this new constraint. It is easy to show that, provided no other constraints of E affect the immediate neighbourhood of (i,i), a new optimal assignment can always be obtained by either pairing p_i with q_{i-1} and p_{i-1} with q_i, or pairing p_i with q_{i+1} and p_{i+1} with q_i. When the cheaper of these alternatives has been determined, the assignment and dual variables can be adjusted accordingly.

The final possible effect arises when two or more constraints involve neighbouring matrix elements - e.g. $E=\{(i,i),(i,i+1)\}$ or even $E=\{(i,i),(i+2,i+2)\}$. Then re-optimisation involves a full iteration of the Hungarian algorithm, but even this does not require calculation of the reduced matrix. At all times the matrix elements are defined by just four vectors ($\underset{\sim}{p}$, $\underset{\sim}{q}$, $\underset{\sim}{\mu}$ and $\underset{\sim}{\nu}$) plus the exclusion set E. Any element can be evaluated as needed, and changing a whole row or column is achieved by changing just one dual variable.

The simple example in figure 4-10 should clarify the algorithm.

Statement of problem:

$\underset{\sim}{p} = (0,2,3,5)$
$\underset{\sim}{q} = (5,4,1,0)$
$\tilde{E} = \{(3,1),(2,2),(4,3),(3,3)\}$

$$(p_i q_j) = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 10 & 8 & 2 & 0 \\ 15 & 12 & 3 & 0 \\ 30 & 24 & 6 & 0 \end{bmatrix}$$

Ignoring E, formula 4-15 gives:

$\underset{\sim}{\mu} = (0,8,9,9)$
$\underset{\sim}{\nu} = (0,0,-6,-9)$
$\rho(1)=1,\ \rho(2)=2,\ \rho(3)=3,\ \rho(4)=4$
$W=11=V$

$$(p_i q_j - \mu_i - \nu_j) = \begin{bmatrix} 0^* & 0 & 6 & 9 \\ 2 & 0^* & 0 & 1 \\ 6 & 3 & 0^* & 0 \\ 21 & 15 & 3 & 0^* \end{bmatrix}$$

Allowing for $(3,1) \notin E$ does not invalidate this assignment.

$$\begin{bmatrix} 0^* & 0 & 6 & 9 \\ 2 & \infty & 0 & 1 \\ \infty & 3 & 0^* & 0 \\ 21 & 15 & 3 & 0^* \end{bmatrix}$$

But $(2,2) \notin E$ causes a conflict. It is resolved by $\rho(1)=2$, $\rho(2)=1$, $W=13=V$, decreasing μ_1 by 2 and increasing ν_1 and ν_2 by 2.

$$\begin{bmatrix} 0 & 0^* & 8 & 11 \\ 0^* & \infty & 0 & 1 \\ \infty & 1 & 0^* & 0 \\ 19 & 13 & 3 & 0^* \end{bmatrix}$$

$(4,3)$ does not cause any conflict, but $(3,3)$ does and it interacts with both $(2,2)$ and $(4,3)$. An iteration of the Hungarian algorithm is needed and this results in μ_1 and μ_2 being decreased by 1 and ν_1, ν_2 and ν_3 being increased by 1.

$$\begin{bmatrix} 0 & 0^* & 8 & 11 \\ 0^* & \infty & 0 & 1 \\ \infty & 1 & \infty & 0 \\ 19 & 13 & \infty & 0^* \end{bmatrix}$$

Now $\underset{\sim}{\mu}=(-3,7,9,9)$, $\underset{\sim}{\nu}=(3,3,-5,-9)$,
$\rho(1)=1,\ \rho(2)=3,\ \rho(3)=2,\ \rho(4)=4,$
$W=14=V.$

$$\begin{bmatrix} 0^* & 0 & 8 & 12 \\ 0 & \infty & 0^* & 2 \\ \infty & 0^* & \infty & 0 \\ 18 & 12 & \infty & 0^* \end{bmatrix}$$

Figure 4-10.

Example of the restricted ranking algorithm.

The matrices are only explanatory - none are actually calculated.

The unconstrained solution is found by ranking, V=11. The first and third constraints cause no trouble, and the second only needs a simple exchange which increases V to 13, but the fourth constraint is diagonal and interferes with two other constraints; hence an iteration of the Hungarian algorithm is needed and, in fact, it would still be required even if the third constraint, (4,3), were not present. The minimum cost for this restricted problem is V=14.

4.8 COMPUTATIONAL ANALYSIS OF RESTRICTED RANKING

The restricted ranking algorithm was programmed in FORTRAN for the CDC 6400 computer. The program can be divided into two phases and flow-charts for both are given in figure 4-11. Phase A begins with the unrestricted ranking assignment and then adjusts this by simple pairwise interchanges to allow for as many constraints as possible. This will yield the optimal solution for many problems.

Phase B deals with any constraints which could not be accommodated in phase A. Each constraint is considered in turn and the assignment and dual variables are continually updated. The method used is adapted from Yaspan's labelling procedure for the Hungarian algorithm (70). The matrix mentioned in the flow-chart is abstract in the sense that its elements, $(p_i q_j - u_i - v_j)$, are not stored at all, but merely calculated when a particular element is required.

The labelling procedure only needs to know the zero elements

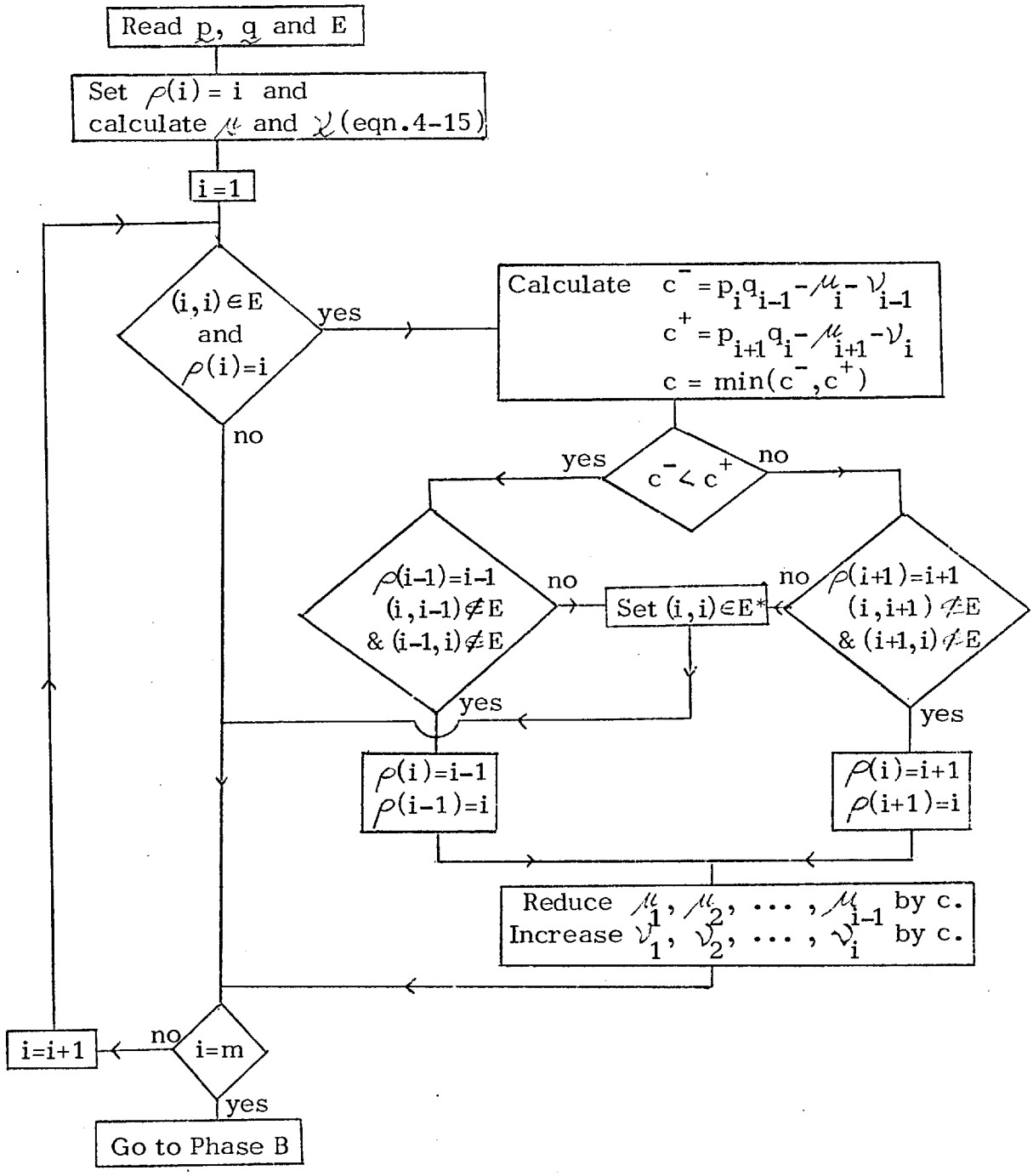


Figure 4-11(A).

Phase A of the restricted ranking program.

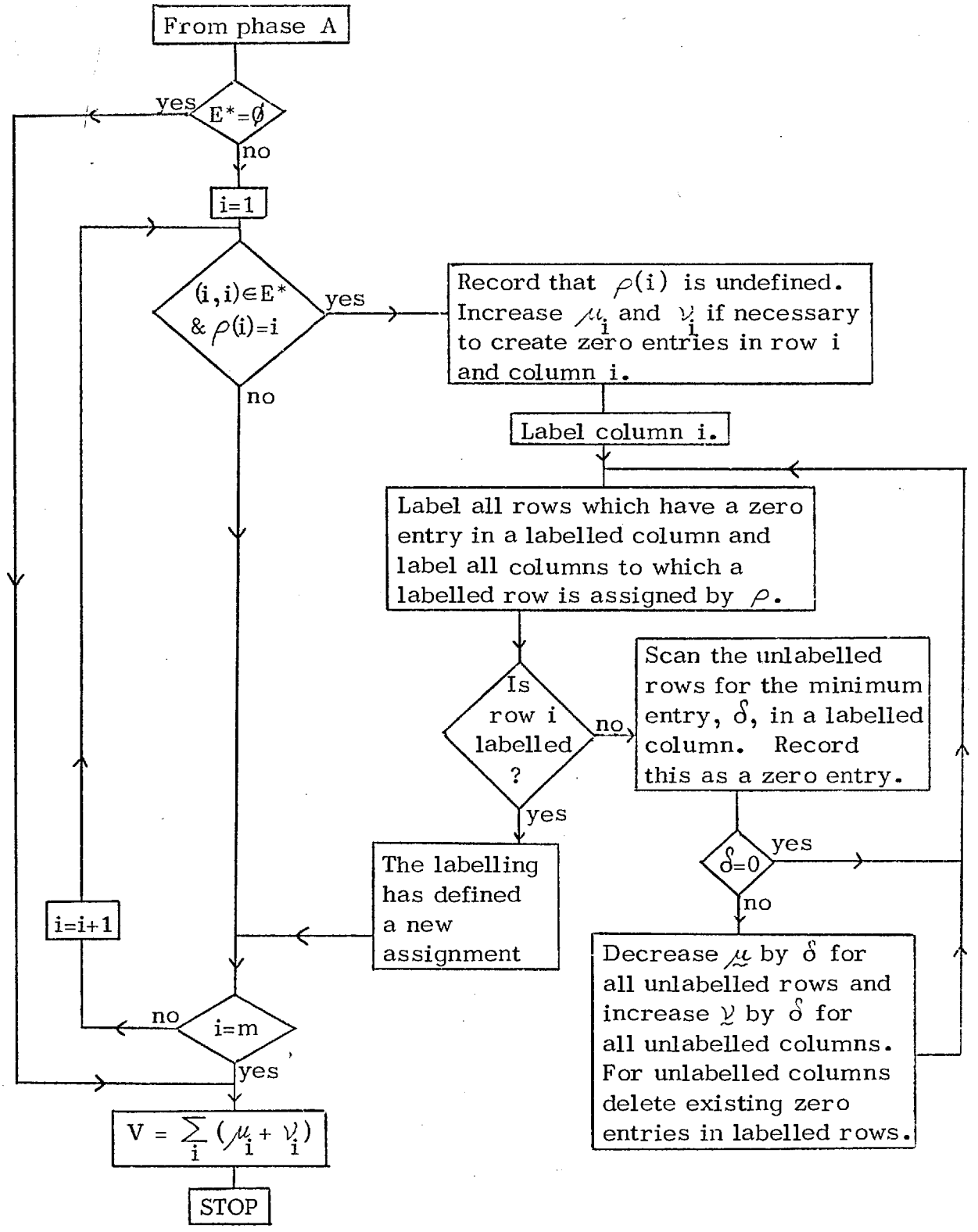


Figure 4-11(B).

Phase B of the restricted ranking program.

of the matrix and so a record of zero entries is kept for each column. The original record is made at the beginning of phase B and thereafter updated as described in the flow-chart. The only occasion on which other entries must be calculated is when the labelling procedure fails to label row i; then only approximately a sixth of the matrix is evaluated.

Efficiency

This program should be more efficient than a general program for the assignment problem and so an experimental comparison was made between it and the assignment routine from the CERN program library (8); that routine uses the Hungarian algorithm and is based on Munkres's method (54).

Table 4-6 shows the time used by each routine for several problems. The specialised algorithm is very much faster than the standard routine, 800 times faster for one problem.

The time needed by the CERN routine depends mainly on the size of the problem, m, but for the restricted ranking program the number of constraints is as important, and also the form of interaction of the constraints. For all the problems except number 4 the interaction was strong enough to require phase B. The constraints for problem 5 were deliberately chosen to make solution as difficult as possible: for the optimal assignment $|\rho(i)-i|$ was greater than or equal to two for 15 out of the 40 numbers.

The great efficiency of the algorithm is due to two factors:

Problem Number	m	|E|	Computation Time (secs.) Restricted Ranking	CERN	CERN*
1	6	8	0.005	0.012	0.005
2	15	15	0.014	0.521	0.063
3	40	20	0.052	41.722	0.573
4	40	40**	0.042	31.770	0.446
5	40	50***	0.302	27.976	0.798

Note: ** - this problem had "easy" constraints

*** - this problem had "hard" constraints.

Table 4-6.

Comparison of computation times for five restricted ranking problems.

(1) it starts with the unrestricted ranking assignment which should be near the final assignment and (2) it does not have to manipulate a large matrix. The relative importance of these factors was examined using the CERN routine. To do this the cost matrix was defined to be $(p_i q_j - \mu_i - \nu_j)$ rather than simply $(p_i q_j)$; $\underset{\sim}{\mu}$ and $\underset{\sim}{\nu}$ were calculated according to equation 4-15. Thus all the finite diagonal elements are zero and this is equivalent to starting the routine with the assignment found by unconstrained ranking.

The computing times for this modified problem are given in table 4-6 under the heading CERN*. These results are very much better than for the original CERN routine, but still significantly slower than for the specialised routine.

Applications

Apart from its use in calculating lower bounds for the quadratic assignment problem, the restricted ranking formulation can be applied to other situations.

Still in a factory environment, the machines may not interact with each other, but all interact with some fixed facility such as a warehouse or store-room. Then p_i measures the flow of goods from machine i and q_j is the distance from location j to the fixed facility. If a particular machine cannot be assigned to a particular location because of bad ventilation, a low ceiling or any other reason, that machine-location pair would be included in E.

Another example. An exporter might manufacture goods at many different plants but export everything through just one port. Here p_i measures the volume of goods produced at plant i and q_j is the cost of transport to the port from location j. The set E contains the plants which cannot be placed at particular locations; for example some plants may need to be near a river for cooling purposes, or far from urban areas because of hazardous or polluting processes.

The restricted ranking algorithm that has been described here can solve such problems extremely efficiently. More than a thousand machines could probably be handled without great difficulty whereas the CERN routine would be hopelessly inadequate, needing a matrix consisting of more than a million elements.

4.9 A BOUND USING RESTRICTED RANKING

Returning to the ideas in section 4.6, the restricted ranking algorithm can be used to calculate a lower bound. This will be at least as good as the basic bound and may be considerably better for nodes with many exclusions.

Program LOCATE was modified to calculate this bound according to the equations 4-12 and 4-13. Some results for the modified program are given in table 4-7. Comparison with table 4-1 shows that, although the number of nodes is reduced by approximately 10% for most problems, the time needed is increased by roughly 30%.

Problem	Number of Nodes	Time (secs.)
GP4	9	0.121
LA7	16	0.398
NVR8	283	7.885
TSP10	347	14.063
MG10	140	11.239
MG12	405	39.308
MG10Z	1248	85.214
NVR12P	2033	134.520
NVR12	10126	707.804
MG14	11576	1590.237

Table 4-7.

Statistics for LOCATE including restricted ranking.

A similar but more exaggerated result would be expected if the best bound proposed in section 3.6 were programmed - i.e. a smaller number of nodes but a much greater computing time.

4.10 LARGE PROBLEMS

For more than 13 or 14 machines the computing time of LOCATE becomes excessive, but the program can still be used in a heuristic fashion to determine good assignments. This is because LOCATE produces a sequence of assignments, each being cheaper than the previous ones.

The fastest heuristic simply takes the first assignment to be found. Not only is this method fast, but the time is predictable and is only dependent on the number of machines. Table 4-8 gives the computing times for the larger problems proposed by Nugent et al. The quality of the assignments is considered in chapter 7.

Any level of compromise is possible between this heuristic and the complete LOCATE algorithm. The time needed to find an improved assignment if one exists is, of course, quite unpredictable.

Problem	Time (secs.)
NVR12	0.98
NVR15	1.98
NVR20	6.96
NVR30	30.15
ST36	62.37

Table 4-8.

Computing time for LOCATE to find its initial assignment.

CHAPTER 5

GRAPHICAL ANALYSIS OF BOUNDS

5.1 THE SEARCH FOR LOWER BOUNDS

A tree-search algorithm can only be effective if its lower bounds are large enough to eliminate many nodes high in the tree so that most branches are cut off quickly. But despite the obvious importance of using good bounds, all the algorithms in the literature (20,21,38,39,58) use minor variations of just two known bounds.

One purpose of this chapter is to discover why no other bounds have been discovered. The two known bounds are shown to belong to a large class of bounds which is exhaustively examined.

The difficulty of the quadratic assignment problem lies in the quadratic rather than the linear part of the cost function; hence it is assumed that $c_{\alpha i}=0$ throughout the initial examination. A bound which cannot be applied to the quadratic cost alone will not be helpful for the complete problem and so no useful bounds will be overlooked because of this restriction.

5.2 GRAPHICAL REPRESENTATION

The quadratic assignment problem can be represented as the mapping of a flow graph to a distance graph. The flow graph has n nodes representing the machines and the flows between machines are represented by arcs which are weighted accordingly. Similarly the n nodes of the distance graph represent the locations and they are

connected by arcs which are weighted with the distances between the relevant locations.

An assignment is a mapping of the nodes of the flow graph on to the nodes of the distance graph. Each graph is complete and so an assignment will pair each flow arc with a distance arc. The cost of an assignment is simply the sum of the products of these flow-distance pairs. Figure 5-1 gives the graphs for a 5-machine problem.

The assumption that there are no linear costs is convenient but not essential for this representation. The linear costs would be associated with the nodes whereas the quadratic costs are associated with the arcs. It is the analysis in the next two sections which requires the zero linear cost assumption.

5.3 A BOUND USING STAR GRAPHS

A complete n-node graph can be decomposed into n star graphs, as shown in figure 5-2. Each star has n-1 arcs and each arc occurs in two stars. The stars of a flow graph can be labelled $\phi_A, \phi_B, \ldots, \phi_\alpha \ldots$ where the subscript denotes the central node; similarly the stars of a distance graph are $\delta_1, \delta_2, \ldots, \delta_i, \ldots$.

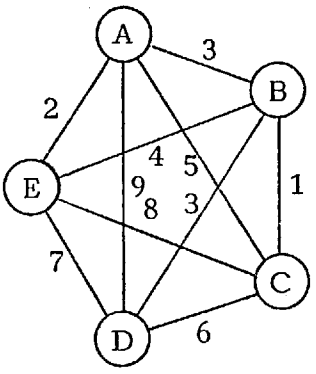
Consider any assignment, ρ, for the quadratic assignment problem and any partial graph of the flow graph which is a star, say ϕ_α. The arcs of this star will all be mapped to distance arcs which are incident on location $\rho(\alpha)$; hence ϕ_α will be mapped to the distance

	A			
B	3	B		
C	5	1	C	
D	9	3	6	D
E	2	4	8	7

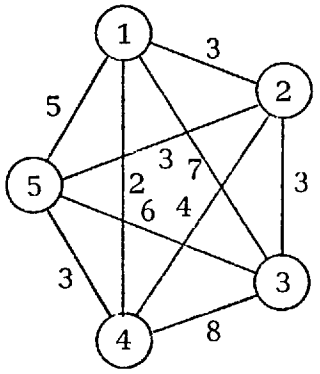
Flow Matrix

	1			
2	3	2		
3	7	3	3	
4	2	4	8	4
5	5	3	6	3

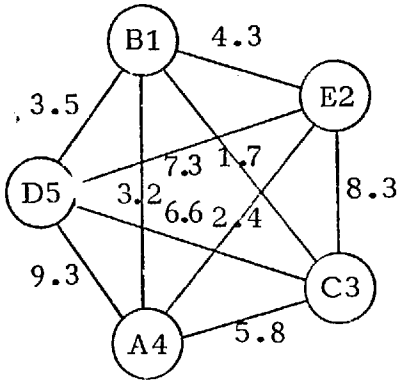
Distance Matrix



Flow Graph



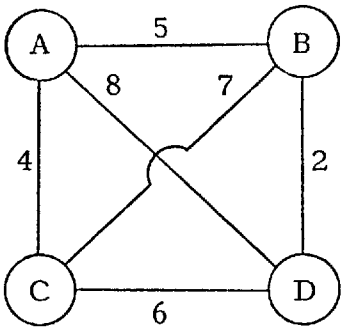
Distance Graph

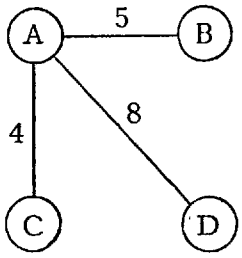


An assignment with cost = 196.

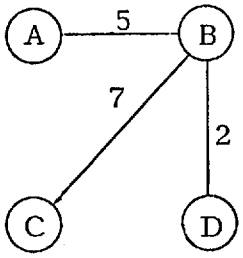
Figure 5-1.

Graphical representation of a problem with 5 machines.

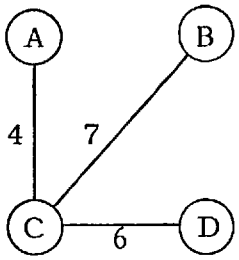




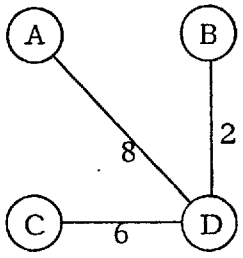
ϕ_A



ϕ_B



ϕ_C



ϕ_D

Figure 5-2.

Decomposition of a 4-node flow graph into 4 star graphs.

star $\delta_{\rho(\alpha)}$.

Define $C(\alpha,\rho)$ to be the cost of mapping the flow star ϕ_α to the distance star $\delta_{\rho(\alpha)}$; this is the sum of the products of the flows with the distances to which they are mapped. Algebraically this means $C(\alpha,\rho) = \sum_{\beta\neq\alpha} f_{\alpha\beta}\, d_{\rho(\alpha)\rho(\beta)}$. Then the total cost of the n star mappings defined by ρ is exactly twice $Z(\rho)$ because the products in the expressions for cost are identical, but each product appears in exactly two star mappings. This allows a lower bound for the quadratic assignment problem to be found in terms of lower bounds on star mappings.

As noted earlier, any assignment will map each flow star ϕ_α to a distance star $\delta_{\rho(\alpha)}$. If the complete assignment is unknown, but $\rho(\alpha) = i$, then a lower bound on the cost of mapping ϕ_α to δ_i is $b_{\alpha i} = \min\left\{ C(\alpha,\rho) \;\middle|\; \rho(\alpha)=i \right\}$. This expression can be easily calculated by ranking the n-1 distances in decreasing order, the n-1 flows in increasing order and summing the products of corresponding numbers; this technique has already been used in section 4.3 and is considered specifically in section 6.4 .

For any assignment $2Z(\rho) \geqslant \sum_\alpha b_{\alpha\rho(\alpha)} \geqslant B$, where $B = \min_{\text{all}\,\rho}\left\{\sum_\alpha b_{\alpha\rho(\alpha)}\right\}$. Finding B, having calculated $(b_{\alpha i})$, is simply a linear assignment problem which can be solved very easily (36,54). Clearly $\frac{1}{2}B$ is a lower bound for the quadratic assignment problem. It is mathematically equivalent to the purely algebraic bounds proposed

by Lawler (39) and Gilmore (21) and formally validated in section 4.3.

5.4 BOUNDS USING OTHER PARTIAL GRAPHS

Since star graphs form the basis of an effective lower bound for the quadratic assignment problem, perhaps other partial graphs can also prove useful in this way.

How can the star graph technique be generalised? It begins with a set of flow partial graphs, $\Phi = \{\phi_1, \phi_2, \ldots, \phi_p, \ldots, \phi_P\}$. For stars there are n partial graphs, each characterised by a particular node, but this need not be so. It is only necessary that each arc of the complete graph should appear in the same number, k say, of partial graphs. The partial graphs need not all be of the same form, but it will become apparent later that the bound is more efficient if the partial graphs are all isomorphic.

Once Φ has been chosen, the set of distance partial graphs, $\Delta = \{\delta_1, \delta_2, \ldots, \delta_q, \ldots, \delta_Q\}$, can be determined accordingly: $\Delta = \{\rho(\phi_p) \mid p = 1,2, \ldots, P;$ all assignments $\rho\}$, where $\rho(\phi_p)$ is the distance partial graph to which ρ maps ϕ_p. Note that $Q \geqslant P$.

It should be clear why Δ must include every image graph of every flow partial graph. Consider the 5-node quadratic assignment problem in figure 5-3 for which Φ consists of two Hamiltonian circuits, ACDEBA and AECBDA. Suppose Δ does not include the circuit 123451. Then any attempt to find a bound would map the flow circuit ACDEBA to

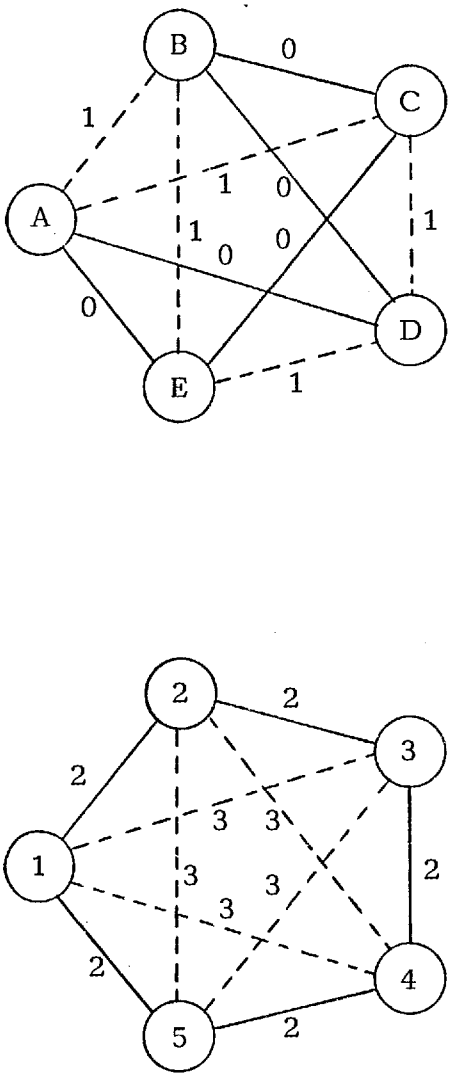


Figure 5-3.

The arcs of a 5-node graph can be covered by a pair of Hamiltonian circuits. One is shown by a continuous line, the other is broken.

a distance circuit which included a value of 3 and so the "bound" would be at least 11, whereas the actual optimal value is 10. Such difficulties can always arise if any image graphs are omitted.

The bounding procedure is the same as for star graphs once Φ and Δ have been chosen. Any assignment ρ determines a mapping from Φ on to Δ and also a mapping from each ϕ_p to $\delta_{\rho(p)}$. The cost of a partial graph mapping can be defined as for star graphs: $C(\phi_p, \rho)$ is the sum of the products of the flows in ϕ_p with the distances in $\delta_{\rho(p)}$ to which they are mapped.

Then $$k Z(\rho) = \sum_{p=1,P} C(\phi_p, \rho) \geqslant \sum_{p=1,P} b_{p\rho(p)} \geqslant B ,$$

where b_{pq} is the minimum cost of mapping ϕ_p to δ_q and B is the solution to the linear assignment problem defined by the matrix (b_{pq}); if necessary the matrix can be made square by adding rows of zeros. B/k is a lower bound on the quadratic assignment problem.

5.5 COMPUTATIONAL CONSIDERATIONS

The above analysis can theoretically take any set Φ of flow partial graphs which cover every arc the same number of times and produce a lower bound. But for many sets the amount of computation would be wildly exorbitant.

The last stage of the calculation is a PxQ linear assignment problem to find B. This could be difficult if P or Q is greater than about a hundred. Evaluating the coefficients b_{pq} can pose two difficulties.

Firstly there are P.Q evaluations to be made and this number may be very large. Secondly it may not be trivial to find each minimal partial graph mapping.

When trying to synthesise useful flow partial graphs the primary concern is to keep Q small. One obvious aid is to choose all the flow partial graphs to be isomorphic so that they each generate identical distance partial graphs. In practice this usually means that Φ has exactly the same structure as Δ and P equals Q. The following analysis is not restricted to this symmetric choice of Φ, but that form appears to be most efficient.

For any partial flow graph define node α to be equivalent to node β if α is directly connected to every node to which β is directly connected and vice versa, ignoring any possible arc between α and β themselves. This is an equivalence relation which will divide the nodes into one or more disjoint equivalence classes.

If one class contains r nodes there are $\binom{n}{r}$ possible images for that equivalence class and so $Q \geqslant \binom{n}{r}$. Thus computation is probably feasible if $r = 1$ or n-1, difficult if $r = 2$ or n-2, and impractical if $3 \leqslant r \leqslant n-3$, even for problems with small values of P.

5.6 DETAILED ANALYSIS OF PARTIAL GRAPHS

This section considers all the potentially useful flow partial graphs in turn. First note that a partial graph with only one equivalence class either has no arcs at all or else is the complete graph; these

extreme cases are of no use.

Partial Graphs with two equivalence classes

Partial flow graphs with only two equivalence classes are the most likely to be computationally feasible. If one class has 1 node ($r = 1$), then the other has n-1 nodes. The central node must either be connected to all the other nodes or not connected to any of them; the outer nodes must similarly be either all connected to each other or all disconnected. The only two non-trivial such partial graphs are shown in figure 5-4. Case (a) is a star which is known to yield a useful bound.

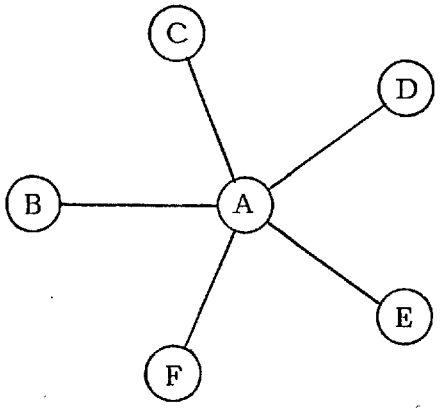
The converse graph, (b), is not computationally feasible because calculation of b_{pq} involves solving a quadratic assignment problem of size n-1. It is clearly impractical to solve n^2 problems of size n-1 merely to find a bound on a problem of size n.

Next consider partial graphs for which one equivalence class contains 2 nodes and the other n-2. Figure 5-5 shows the six possible partial graphs. Cases (a), (b) and (c) are unacceptable because calculation of each b'_{pq} is nearly as difficult as the original problem.

Case (d) yields a bound which is, in fact, the same as those proposed by Gavett and Plyter (20), Gilmore (21) and Land (38). It is particularly useful because the special structure of its final linear assignment problem can be exploited.

Cases (e) and (f) are very similar to each other. A difficulty arises in finding b_{pq} for these "double-stars" because the

(a)



(b)

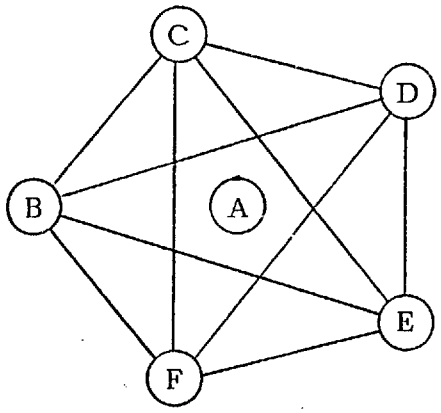


Figure 5-4.

The partial graphs with r = 1 & n-1.

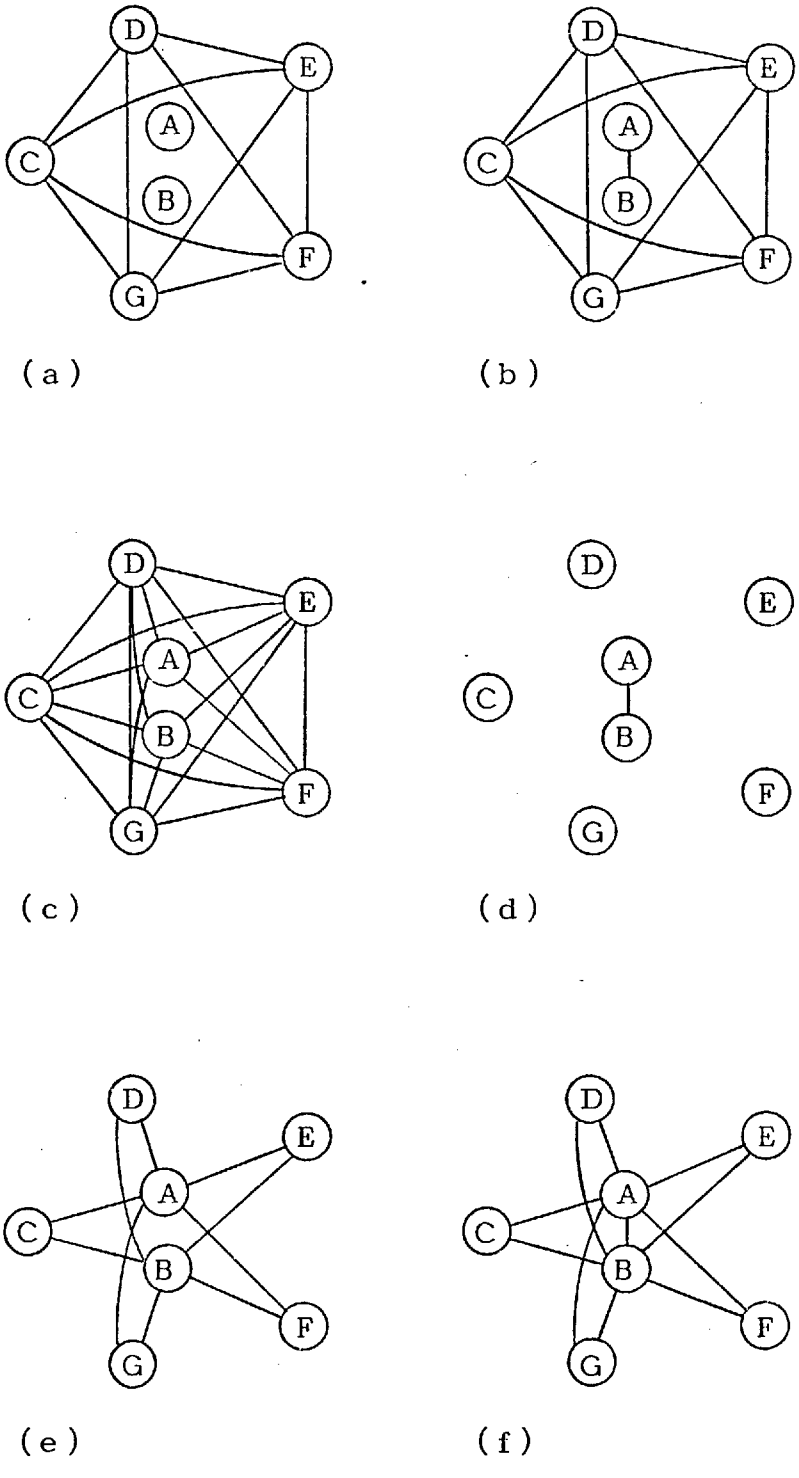


Figure 5-5.

The partial graphs with r = 2 & n-2.

simple ranking procedure used for stars cannot be extended. The coefficients can be calculated by solving an (n-2)x(n-2) linear assignment problem for each of them, but the large number of coefficients renders this approach cumbersome.

Partial Graphs with Three Equivalence Classes

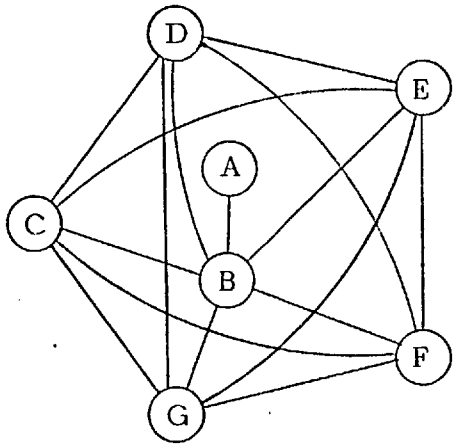
By now most of the potentially useful partial graphs have been examined, but those with 3 equivalence classes should also be considered. The only partial graphs which may be useful are those having two classes of one node each and the remaining class containing n-2 nodes. For such a graph Q can be as small as n(n-1).

There are only two such partial graphs and these are given in figure 5-6. For case (a) it is too difficult to calculate each coefficient. For case (b) the coefficients can be readily determined but there are a great many of them. The massive computation can only be justified if the bounds produced are exceptionally good - there is little reason to expect they are.

5.7 CONCLUSIONS

This chapter has presented an integrated analysis of lower bounds for the quadratic assignment problem. It has studied a large class of bounds which includes both those considered in the literature. Only three other bounds are conceivably useful, and even these require much more computation than the earlier bounds.

(a)



(b)

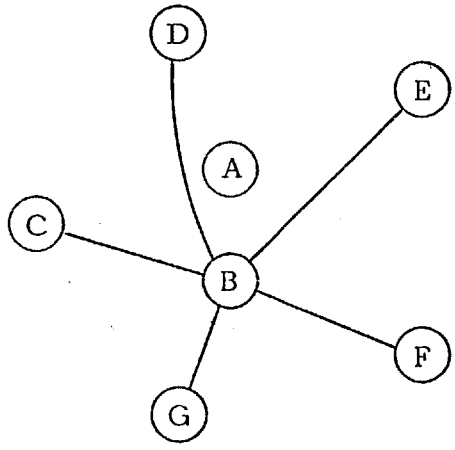


Figure 5-6.

The partial graphs for which r = 1, 1 & n-2.

A tree-search algorithm needs bounds on subsets of the feasible solution space (the nodes of the tree) as well as on the entire space. This study has been directed at the complete problem, but any of the bounds can be readily adapted as required without difficulty.

A second restriction has been to assume that the linear costs are zero. This assumption simplified the analysis and has not caused any bounds to be omitted. Gilmore (21) has suggested one general way to incorporate linear costs. Simply solve the linear assignment problem defined by the linear cost coefficients and add its solution to the bound on the quadratic cost calculated in any of the ways considered here (or otherwise). Bounds produced in this way are unlikely to be very good, but the linear costs can usually be incorporated more directly as was done in chapter 4 for the star graph bound.

The quality of the lower bounds has not been considered explicitly, but it seems unlikely that the three new bounds would be significantly better than the star graph bound - not sufficiently good to justify the much greater computation that would be required. This is disappointing since the tree-search algorithm with the star graph bound cannot solve problems with more than about 14 machines.

CHAPTER 6

SOME SPECIAL CASES

6.1 ZERO FLOWS AND INFINITE DISTANCES

The previous chapters have shown that the quadratic assignment problem can generally only be satisfactorily solved for quite small problems. There has been greater success with algorithms for specific problems, particularly the 1-dimensional case (40,64). Some related problems which can be artificially forced into the quadratic assignment format, such as the travelling salesman problem (10,27,28) and Bowman's input-output problem (3), can also be solved more easily. This chapter studies another class of special cases.

In a plant layout context there are usually many pairs of machines which do not interact with each other. In other words the flow matrix has many zero entries. The main assumption used throughout this chapter is that the flow matrix is sparse.

A second assumption which is sometimes made concerns the distance matrix. Movement directly between two locations may be prohibited for some reason and so such distances can be considered "infinite". At an international level the reason for a prohibition may often be political, but economic reasons are more likely for plant layout.

Any assignment which matches a positive flow to an infinite distance is, by definition, infeasible. For each infinite distance a feasible assignment must occupy the two locations involved with two machines which have zero flow between them.

6.2 GRAPHICAL INTERPRETATION

The graphical representation introduced in the previous chapter is again very convenient. As before, the flow graph has n nodes representing the machines and the distance graph has n nodes representing the locations.

Unlike chapter 5, however, these graphs are not necessarily complete. Only each positive flow is represented by an arc, zero flows being ignored. Likewise only the finite distances are represented by arcs. The flow and distance arcs carry weights according to the values of the corresponding flows and distances.

An assignment is a mapping of the flow graph to the distance graph and the quadratic cost of an assignment is the sum of the pairwise products of the weights on the arcs which are mapped to each other. An assignment is clearly feasible only if every flow arc is mapped to a distance arc; it is not necessary, however, for every distance arc to be the image of a flow arc.

If the distance graph is sparse - i.e. there are many infinite distances - then the number of feasible assignments may be small and perhaps they can be enumerated to find the optimum. The next section considers this situation and complete distance graphs are studied later.

6.3 ELEMENTARY PROBLEMS

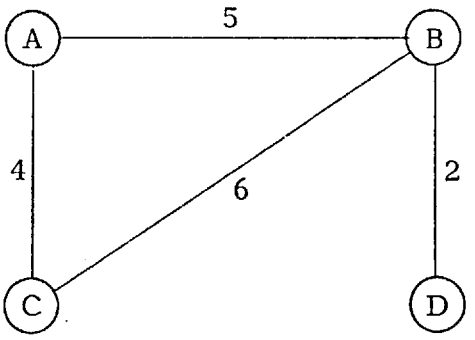
This section deals with the limiting special case in which the number of infinite distances exactly equals the number of zero flows. The number of feasible assignments may therefore be very small.

First note that the flow and distance graphs must be isomorphic for any feasible assignment to exist. For example, the 4-node flow and distance graphs in figure 6-1 have equal numbers of arcs but are not isomorphic and so any assignment must map at least one positive flow to an infinite distance. It can be difficult to determine whether graphs are isomorphic or not.

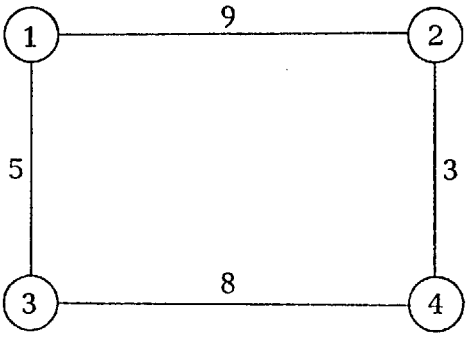
Isomorphic Number of a Graph

If the graphs are isomorphic then the number of feasible assignments depends on the actual shape of the graphs. Figure 6-2 shows a pair of isomorphic graphs (arc weights can be ignored in this section). For any feasible assignment $\rho(D)=1$ and $\rho(B)=2$, but machines A and C can be assigned to locations 3 and 4 in two possible ways.

The isomorphic number of a graph may be defined as the number of ways in which it can be mapped to itself without changing the relative positions of the arcs. Hence the graphs in figure 6-2 have isomorphic number 2. The number of feasible assignments is equal to the isomorphic number and computer enumeration of these assignments is possible if the number is not too large. Note that the isomorphic number of a complete graph is $n!$ and this becomes too large when n

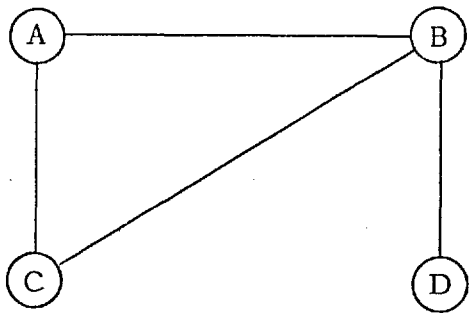


Flow Graph

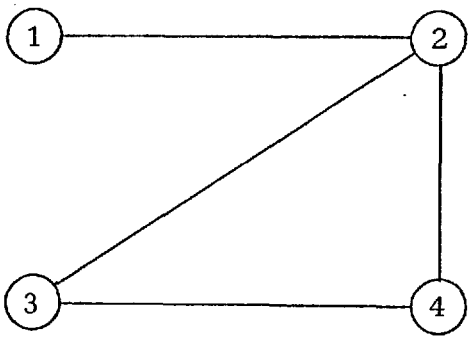


Distance Graph

Figure 6-1.



Flow Graph



Distance Graph

Figure 6-2.

exceeds about 8 or 9.

Chains and Cycles

Consider a chain of n nodes (figure 6-3(a)). Clearly machine A must be assigned to either location 1 or n. Once A is located, the assignment of all other machines is fixed and so the isomorphic number of a chain is 2.

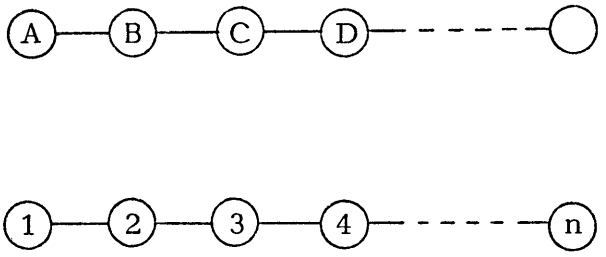
For a cycle (figure 6-3(b)) machine A can be assigned to any location and then the other machines must be assigned in one of only two ways, clockwise or anti-clockwise. Hence the isomorphic number is 2n.

Star-Cycles

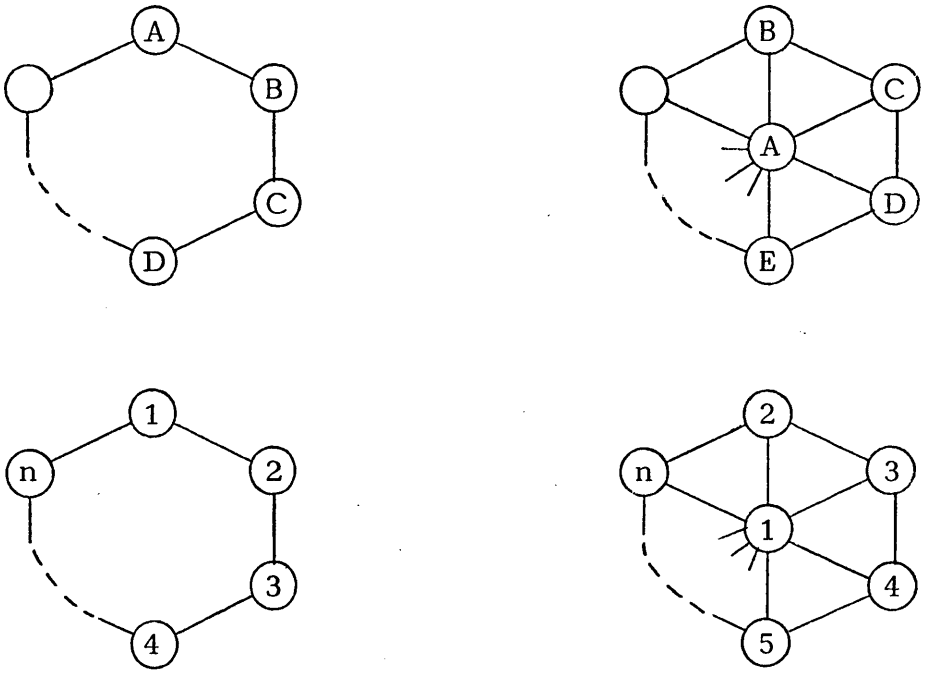
A star-cycle (figure 6-3(c)) is an easy extension of a cycle. Obviously a machine-node must be assigned to a location-node of the same degree and so $\rho(A)=1$ is essential. The symmetry of the remaining n-1 nodes is as for the cycle - therefore the isomorphic number of the star-cycle is 2(n-1).

Larger Graphs

The need for assigning machine-nodes to location-nodes of the same degree ensures that many graphs have small, even unitary, isomorphic numbers. The graphs in figure 6-4, for example, are isomorphic and have isomorphic number 1. A proof of this begins by assigning H to 3 as these are the only nodes of degree 4; then, working



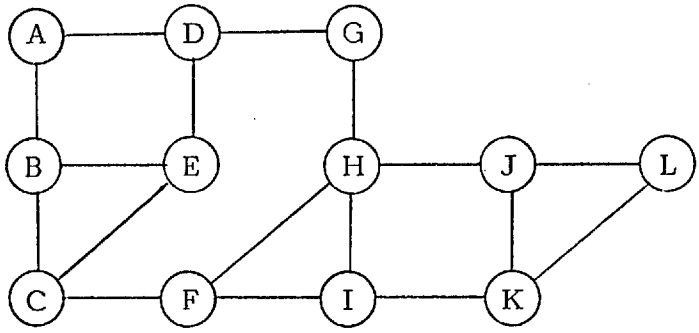
(a)



(b) (c)

Figure 6-3. Some simple graphs.

(a) Chain. (b) Cycle. (c) Star-cycle.



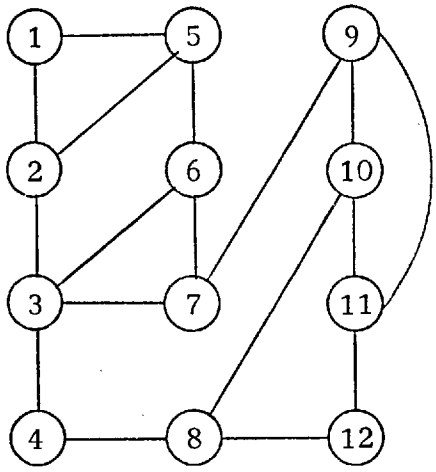


Figure 6-4.

Isomorhic graphs with IN = 1.

round the graphs, the assignments G-4, D-8, A-12, B-11, E-10, C-9, F-7, I-6, K-5, L-1 and J-2 are all necessary.

Regular Graphs

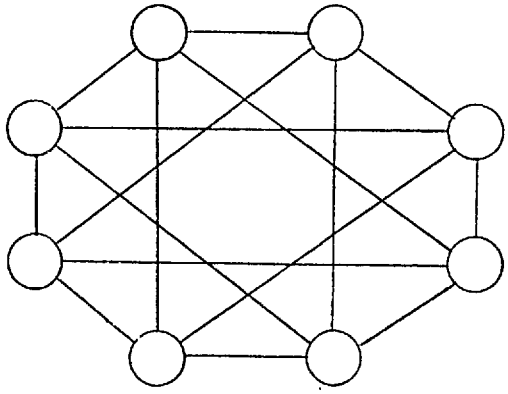
Regular graphs (all nodes having the same degree) are likely to have large isomorphic numbers because the nodes are less easily distinguishable. But even within this difficult class there is great variation; figure 6-5 shows two 8-node regular graphs of degree 4 which have very different isomorphic numbers. Cycles and complete graphs are both regular and indicate that the isomorhic number tends to increase with the degree of the nodes.

6.4 ELEMENTARY TREES

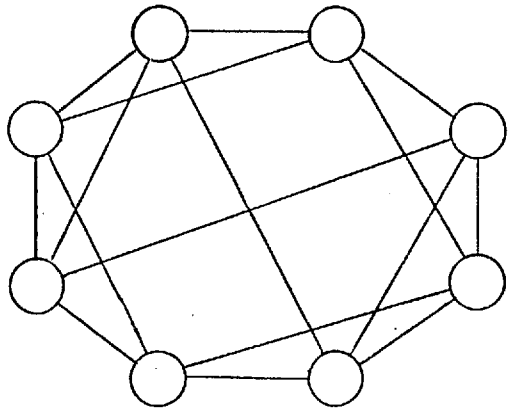
Many systems, especially hierarchical ones, have flow graphs in the form of a tree. If the distance graph is also of this form, the optimal assignment can often be found even if the isomorphic number is large. And, in fact, the isomorphic number may be quite small.

Multi-level Stars

A star is a special type of tree with complete symmetry about a central node. Figure 6-6 shows a third order star in which node B is in level 1, C is in level 2 and E is in the outermost level 3; node A constitutes level 0 and is called the centre. Nodes C and D are called the successors of node B. It is convenient to label each



(a) IN = 1152



(b) IN = 4

Figure 6-5.

Two regular graphs of degree 5.

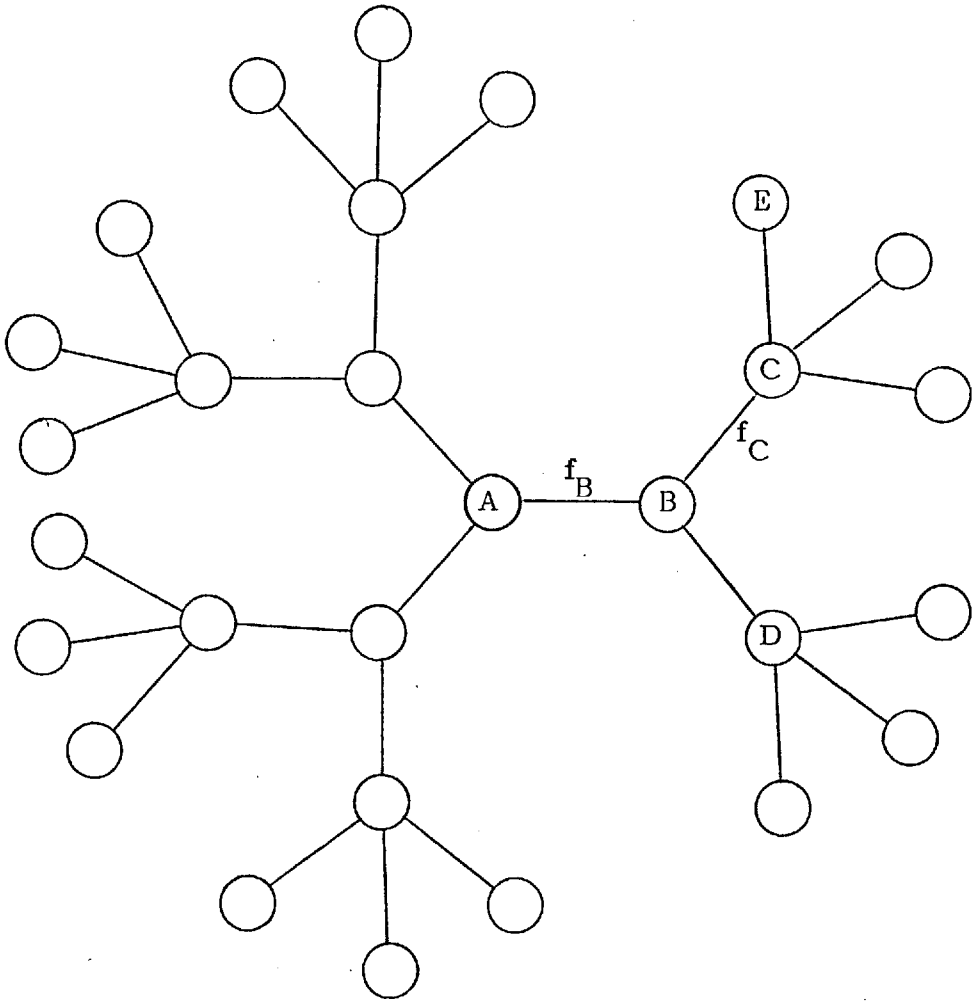


Figure 6-6.

A 3rd order star.

arc in terms of its outer node and so f_{BC} is simply called f_C for example.

Clearly each machine-node must be assigned to a location-node at the same level, but stars have such a high degree of symmetry that their isomorphic numbers are too large for practical enumeration. For the 28-node example the isomorphic number is 2,239,488.

Dynamic Programming Solution of Stars

For any machine-node $\propto$ and location node i define $C(\propto,i)$ to be the cost of assigning $\propto$ and all its successors to i and all its successors when this is done optimally; $\propto$ and i must be on the same levels of their respective graphs. The following algorithm calculates all such costs and hence solves the quadratic assignment problem for a K^{th} order star.

(1) For each machine $\propto$ and each location i at level K, the outermost level, set $C(\propto,i) = c_{\propto i} + f_{\propto} d_i$.

(2) Set level-pointer k = K-1.

(3) For each machine $\propto$ and each location i at level k calculate $C(\propto,i)$ as follows:-

(a) Identify the direct successors of $\propto = \beta_1, \beta_2, \ldots$ and the direct successors of $i = j_1, j_2, \ldots$

(b) Set up a matrix for a linear assignment problem as shown at the top of the next page.

$$\begin{bmatrix} C(\beta_1,j_1) & C(\beta_1,j_2) & \cdots \\ C(\beta_2,j_1) & C(\beta_2,j_2) & \cdots \\ \vdots & \vdots & \\ \vdots & \vdots & \\ \vdots & \vdots & \end{bmatrix}$$

(c) Set $C(\alpha,i) = c_{\alpha i} + f_{\alpha} d_i$ + solution of the linear assignment problem defined in (b).

(4) Set $k = k-1$. If $k \neq 0$ go to (3).

(5) Now $k=0$ and the centre is the only node at this level. Set up the linear assignment problem as in step (3). Its solution gives the optimal cost of a complete assignment and the assignments of individual machines can be determined from the solutions to the linear problems solved in step (3).

Ranking Procedure

For the outermost level of a star the assignment problems in step (3) of the dynamic program can be solved trivially. This is because the elements of the matrix are simple products of flows and distances. As for the bound given in section 4.3, it can be shown (21) that the total cost is minimised by matching the largest flow with the shortest distance and so on. Thus it is only necessary to rank the flows in decreasing order and the distances in increasing order to solve these assignment problems.

How efficient is this algorithm? Solving the star in figure 6-6 involves ranking 12 sets of 3 numbers, evaluating 36 costs at level 2,

solving nine 2x2 assignment problems at level 1 and an additional 3x3 assignment problem for the centre. This could be done by hand in about half an hour. Section 7.5 compares the dynamic program to the general tree-search method.

For a K^{th} order star in which each node has m successors, there are $\frac{m^{K+1}-1}{m-1}$ nodes. Solution requires $2m^{K-1}$ rankings, m^{2K-2} evaluations and $\frac{m^{2K-2}-1}{m^2-1}$ assignment problems, each mxm. This means that stars with several hundred nodes can be solved comfortably using a computer.

Dynamic Programming Solution of Trees

The symmetry of stars makes them the most difficult of trees to solve. Other trees may have small isomorphic numbers and therefore be amenable to enumeration of all feasible assignments.

A large isomorphic number can only be caused by different branches of the tree having identical structure in the same way as a star. Such branches can be solved using the dynamic program independently of the rest of the tree.

First Order Stars

Stars with only one level can be solved very efficiently since only one linear assignment problem is involved. If the linear costs, $c_{\propto i}$, are all zero then even this problem can be solved using the ranking procedure. This method has been assumed in sections 4.3 and 5.2 .

6.5 PRACTICAL PROBLEMS

Prohibited or infinite distances do not occur for some applications of the quadratic assignment problem and so this section assumes that all distances are finite. All assignments are now feasible and therefore isomorphic analysis is irrelevant; any useful approach must involve the arc-weights as well as the structure of the flow graph.

Stars

A first order star flow graph can be assigned to a complete distance graph quite easily. If the location of the central machine was given then only the n-1 distances from that location would be relevant; the remaining assignments could be optimised using the ranking procedure described in section 6.4 . The optimal assignment can be found by costing all the n possible assignments of the central machine in this way and selecting the cheapest.

This treatment cannot be usefully extended to multi-level stars, except in so far as a star is a tree.

General Trees

The following technique is applicable to trees and any other flow graphs for which most nodes have degree 1. Suppose there are n_1 nodes of degree 1 and n_2 nodes of greater degree. If the locations of the n_2 multiple-degree machines were known, then the optimal locations for the n_1 unit-degree machines could be found by

solving an $n_1 \times n_1$ linear assignment problem. The complete optimal assignment can be found by costing in this way all the possible assignments of the multiple-degree machines. The number of these assignments is $(n_1+n_2)(n_1+n_2-1) \ldots (n_1+1)$ and so this method is only practical if n_2 is very small.

An Assembly Line

A chain arises as a flow graph in assembly line operations and is therefore of considerable interest. A chain is, of course, a particular type of tree and so the above discussion applies, but it is not at all helpful.

In a sense this assembly line problem is a generalisation of the open-ended travelling salesman problem, since that problem results when all the flows along the chain are equal. It is not surprising, therefore, that the salesman's dynamic programming algorithm (26) can be generalised to solve this problem.

Dynamic Program for an Assembly Line

It is convenient to rename the machines and flows for this section. The machines become $\propto_1, \propto_2, \ldots, \propto_n$ ordered along the chain. The flow between $\propto_i$ and $\propto_{i+1}$ is called f_i. This is shown in figure 6-7.

For any set of k locations, $S_k = \{s_1, s_2, \ldots, s_k\}$, and any index i, $1 \leqslant i \leqslant k$, define the recursive function $g(S_k, i)$ to be the cost of an optimal assignment of the machines $\propto_1, \propto_2, \ldots, \propto_k$ to the

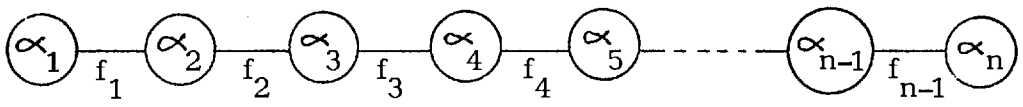


Figure 6-7.

Notation used in dynamic program for a chain flow.

locations S_k when α_k is assigned to location s_i.

For $k=1$, $g(S_1,1) = c_{\alpha_1 s_1}$.

For $2 \leqslant k \leqslant n$, $g(S_k,i) = c_{\alpha_k s_i} + \min_{\substack{j=1,k \\ j \neq i}} \left\{ f_{k-1} d_{s_j s_i} + g(S_k - \{s_i\}, j) \right\}$

The cost of an optimal assignment is $Z = \min_{i=1,n} \left\{ g(S_n,i) \right\}$.

These equations allow Z to be calculated and an optimal assignment to be determined, but a large number of functional values must be evaluated. The computational complexity is the same as for the travelling salesman problem and so the algorithm can solve assembly lines with up to about 18 machines.

6.6 MINIMAL WEIGHTED SPANNING TREES

The dynamic program described in the previous section is a generalisation of an algorithm for the travelling salesman problem, but that algorithm is not the most efficient for the simpler problem. This section tries to extend a spanning tree algorithm from the salesman to the assembly line problem.

The spanning tree algorithm for the travelling salesman problem is based on noticing that a chain is a particular type of tree. A spanning tree of a complete distance graph is simply a partial graph which is a tree; the cost of a spanning tree is the sum of the distances in the tree. Kruskal (35) has given a very efficient algorithm for

finding a minimal spanning tree for a graph.

If a minimal spanning tree is actually a chain then it is also a minimal tour for the travelling salesman problem. The algorithm adds penalties to nodes in such a way that the relative costs of chains remain unchanged but the costs of spanning trees vary until a minimal spanning tree is found which is a chain. Because Kruskal's spanning tree algorithm is so fast, optimal travelling salesman chains can be found for large problems - as many as a hundred nodes.

Kruskal's algorithm is very simple. It ranks the arcs in order of increasing distance and then arcs are taken from the top of this list until a spanning tree is completed. At each stage the shortest arc which does not complete a circuit is chosen. The minimal spanning tree is complete when n-1 arcs have been chosen.

The cost of an assembly line is the sum of the distances in the chain, each weighted with the appropriate flow - for the travelling salesman problem the weights are all one. The cost of a weighted spanning tree may be defined as the sum of the distances in the tree, each weighted with one of the n-1 flows with the flows and distances matched to give the minimal cost; i.e. the largest flow would be matched with the smallest distance in the tree and so on.

If the minimal weighted spanning tree is a chain, its cost may be less than the assembly line cost of that chain because the weights are used differently. But at least it will give a lower bound on the

assembly line cost and this could be used in a branch-and-bound algorithm. An efficient algorithm is needed to find minimal weighted spanning trees.

A Proof for a Minimal Weighted Spanning Tree Algorithm

It happens that any minimal spanning tree is also a minimal weighted spanning tree. Hence Kruskal's algorithm may be used for the weighted problem and a proof of this follows.

Let F be the set of n-1 flows in the assembly line.

If $T = \{t_1, t_2, \ldots, t_{n-1}\}$ is a spanning tree, $d(t_i)$ is the length of arc t_i and $d(T) = \{d(t_1), d(t_2), \ldots . d(t_{n-1})\}$, define the cost of the weighted tree to be $Z(T) = F.d(T)$ where "." is a scalar product which implicitly minimises - i.e. it multiplies the largest flow by the smallest distance and so on.

Now suppose that T is a tree produced by Kruskal's method with the arcs labelled in the order in which they were selected so that $d(t_1) \leqslant d(t_2)$ etc. Let T_0 be any other spanning tree.

Construct a sequence of spanning trees for $k = 1, 2, \ldots, n-1$ such that $\{t_1, t_2, \ldots, t_k\} \subseteq T_k$.

If $t_k \in T_{k-1}$ then $T_k = T_{k-1}$ and of course $Z(T_{k-1}) = Z(T_k)$.

If $t_k \notin T_{k-1}$ then form T_k from T_{k-1} by adding arc t_k and removing the largest arc (not t_k) in the circuit thus formed.

Since T_{k-1} contains $\{t_1, t_2, \ldots, t_{k-1}\}$ Kruskal's construction ensures that all other arcs in T_{k-1} must have

distances greater than or equal to $d(t_k)$. But the circuit must contain arcs other than $t_1, t_2, \ldots, t_{k-1}$ and so the arc removed, t' say, has distance $d(t') \geqslant d(t_k)$.

Now $Z(T_{k-1}) = F.d(T_{k-1})$

= optimal sum of products of F and $d(T_{k-1})$

$\geqslant$ that sum of products with $d(t')$ replaced by $d(t_k)$

= a sum of products of F and $d(T_k)$

$\geqslant F.d(T_k)$

$= Z(T_k)$

Now clearly $T_{n-1} = T$ and $Z(T_0) \geqslant Z(T_1) \geqslant \ldots \geqslant Z(T_{n-1}) = Z(T)$ and so the weighted cost of T is less than or equal to the weighted cost of any other spanning tree. Therefore Kruskal's algorithm produces a minimal weighted spanning tree.

A Spanning Tree Algorithm for Assembly Lines

The above proof shows that Kruskal's algorithm might be used as the basis of a branch-and-bound method for the assembly line problem. The cost of a minimal weighted spanning tree, whether or not the tree is a chain, is a lower bound for the assembly line problem. The general technique outlined in section 4.1 could be applied with various branching strategies to solve the problem.

The bound is likely to be better if the minimal tree is actually a chain and this could be achieved by adding penalties as for the travelling

salesman algorithm. Burkard (5,6) has shown how penalties can be applied to the general quadratic assignment problem (see section 2.3) and his formula simplifies significantly for this special situation.

Unfortunately preliminary calculations by hand have shown that the bounds produced by this method are not good enough to be useful, even when the minimal tree is a chain. In fact the general bound given in chapter 4 is often greater than this specific bound; the calculation of the general bound can be greatly simplified for the assembly line problem by first applying Burkard's transformation.

CHAPTER 7

COMPARISON OF METHODS

7.1 THE NEW METHODS

Earlier chapters have developed algorithms for the quadratic assignment problem and some heuristics have also evolved as a by-product. This chapter empirically evaluates all these methods with regard to reports in the literature of other research.

The non-specialised algorithms considered in this thesis are all of the tree-search type and differ from each other only in the lower bounds they use. The computational results of chapter 4 indicate that the basic bound described in section 4.3 gives the best compromise between the quality of the bounds and the time needed to calculate each bound. Hence this bound has been used when comparing tree-search with other methods.

The tree-search algorithm becomes a heuristic if some branches are ignored without calculating a lower bound which is greater than the cost of a known assignment. The simplest heuristic only considers one branch, the one without any exclusion nodes. This constructive heuristic is very similar to Gilmore's n^5 method described in section 2.4, differing only in its use of the solution to the linear assignment problem. The penalty rule described in section 4.4 is intuitively more reasonable than Gilmore's suggestion.

The quadratic programming methods do not guarantee an optimal assignment, but program PENALTY (section 3.8) finds sub-optimal assignments very quickly. The quality of these assignments

is examined in the next two sections.

Specialised algorithms have been proposed for the multi-level star problem and the assembly line problem in chapter 6. Section 7.5 compares the efficiency of these methods with that of the more general tree-search.

7.2 STEINBERG'S TEST PROBLEM

Two case studies have provided standard problems on which many researchers have tested their procedures. This chapter continues that sensible tradition and the data for all problems is reproduced in appendix A.

Steinberg (65) published a computer backboard design problem involving 34 modules which were to be located on a rectangular grid 9 units long and 4 units wide. This gives 36 locations and so 2 dummy modules (machines) must be added. Three distance functions have been proposed: (1) rectangular distance, (2) direct or Euclidean distance, and (3) direct distance squared.

Most of the constructive heuristics have been tested on this problem using at least one of the distance functions. The direct distance function has received most attention and so this has been adopted for the comparison.

The problem proved too large to solve optimally using the tree-search and so program LOCATE was used as a heuristic.

Program PENALTY, the quadratic programming heuristic, was also tested and the results are given in table 7-1.

The qualification "manual" in the table means that human intelligence directed the computer to some extent. Graves and Whinston noticed that the assignment first produced by their program contained two obviously poor placements and so they re-ran the program with these modules restricted. LOCATE and PENALTY are interactive programs and "manual" implies direction via a remote terminal for about half an hour. The non-manual assignments are those produced initially.

Table 7-1 does not mention the computing times for the various heuristics for two reasons. The first is that they were run on many different computers (sometimes unspecified) and so comparison would be difficult. The second reason is that all these heuristics are quite fast and so time is not critical. If run on a CDC 6400 computer they would all require less than one minute, with the possible exceptions of the Graves and Whinston program and the manual LOCATE.

Conclusions to be drawn from this comparison are that PENALTY is a rather poor method, undirected LOCATE is comparable with the best alternatives and, given time and human direction, LOCATE can produce better assignments.

Heuristic	Cost
Steinberg	4894.54
Gilmore n^4	4547.54
Gilmore n^5	4680.36
Hillier-Connors (Constructive)	4821.78
Graves-Whinston	4426.27
Graves-Whinston (Manual)	4344.97
Heider	4419.49
PENALTY	4905.21
PENALTY (Manual)	4450.79
LOCATE	4411.36
LOCATE (Manual)	4327.84

Table 7-1.

Cost of assignments produced by the heuristics for Steinberg's problem.

7.3 NUGENT'S TEST PROBLEMS

Another case study has been presented by Hillier (29). Nugent, Vollmann and Ruml(57) used the data for this 12-machine problem to develop eight problems with the same distribution of flow-values. Their problems range in size from 5 to 30 machines and all assume rectilinear distances between locations which form a rectangular grid (see appendix A).

Nugent et al. compared the assignments produced by four improvement heuristics on these problems. Since then Edwards et al. (13), Vollmann et al. (67), Khalil (33) and Hitchings (32) have also tested their improvement heuristics on these problems. Many of these heuristics are very similar and so tables 7-2 and 7-3 only reproduce the statistics for four of the more successful programs.

Table 7-2 gives the costs of assignments produced for the five larger problems by the four programs mentioned above as well as LOCATE and PENALTY. Since the last section showed that the first assignment found by PENALTY can be poor, the value given here is that of the best assignment found after a directed search has used 8 seconds of CDC 6400 computing time for the three smaller problems and 40 seconds for the larger problems.

Three values are given for LOCATE. The initial cost is that of the first assignment to be generated. The automatic cost is either that of the optimal assignment (for $n = 8$ and 12) or that of the best

Method	Number of Machines				
	8	12	15	20	30
Hillier (1963)	109	301	617	1384	3244
CRAFT	107	289	583	1324	3148
Nugent et al.	107	289	575	1304	3093
Hitchings	107	289	575	1296	3086
PENALTY	111	289	589	1347	3355
LOCATE (Initial)	107	293	575	1323	3219
LOCATE (Automatic)	107	289	575	1306	3103
LOCATE (Manual)	-	-	575	1282	3094

Table 7-2.

Costs of assignments for Nugent's problems.

Method	Number of Machines				
	8	12	15	20	30
Hillier (1963)*	14	55	78	168	398
CRAFT *	10	70	160	528	3150
Nugent et al. *	109	658	2192	6915	42724
PENALTY **	8	8	8	40	40
LOCATE (Initial) **	0.37	0.98	1.98	6.96	30.15
LOCATE (Automatic) **	5.42	481.67	1800	200	200
LOCATE (Manual) **	-	-	200	200	200

* GE 265 computer.

** CDC 6400 computer.

Table 7-3.

Computing times (seconds) for Nugent's problems.

assignment found within the given time limit (for n = 15, 20 and 30). When the time limit was exceeded a manually directed search continued for a further 200 seconds of CDC 6400 time.

Table 7-3 gives the computing times for the various programs. The three improvement heuristics were run on a GE 265 computer which is considerably slower than the CDC 6400 used for PENALTY and LOCATE. Hitchings' times are not reproduced here, partly because he used yet another computer and partly because the meaning of his time is not clear; it appears that his method is a little faster than Nugent et al.'s biased sampling but considerably slower than CRAFT.

These tables show that PENALTY is a very fast program which gives poor results. The initial assignment found by LOCATE is much better, although not as good as the best improvement heuristics; it is also very fast.

When allowed to run to completion LOCATE finds an optimal assignment, but the time needed for this increases very rapidly as the number of machines increases. It was noted in section 4.5 that the time also depends on the nature of the data and that these test problems have proved particularly difficult. Three of the four heuristics also found optimal assignments for the two smaller problems - although it was not then known that 289 is the optimum for NVR12.

For the larger problems all the methods found different assignments except that LOCATE, Nugent et al. and Hitchings all found an assignment costing 575 for NVR15. Therefore it appears likely that

575 is the optimum for NVR15, but there may well be cheaper assignments for the larger problems.

The initial assignment found by LOCATE is quite good and is produced extremely quickly by comparison with the improvement procedures. Continued searching for the larger problems is a heuristic which is comparable with the best alternatives in terms of quality and computing time.

7.4 A BOUNDED HEURISTIC

An annoying feature of the improvement heuristics is that they do not indicate how much their assignments are worse than optimal. Their are two ways in which a tree-search program such as LOCATE can rectify this situation.

One approach assumes that a good assignment has been found and a user only wishes to be assured that its cost is within, say, 5% of the optimum. Then LOCATE can be run with a special cut-off rule which ignores any node whose bound is greater than 95% of the cost of the known assignment. It is hoped that only a small part of the tree will be enumerated under these conditions to prove that the initial assignment is within the given percentage of the optimum.

The computational results for this method are rather disappointing. Almost 60 seconds were required to show that the optimum for NVR15 is greater than 500; i.e. to show that the suspected

optimum 575 is within 13% of the actual optimum. This indicates that either the bounds are not as good as hoped or that there is a much better assignment not yet discovered.

Gilmore (21) suggested an alternative means of searching a tree to find a solution within a given percentage of the unknown optimum. This does not assume that that a good assignment is known beforehand.

Suppose an assignment within 5% of the optimum is desired. He proposes that the tree-search algorithm should be used normally until the initial assignment is found and than the search continues but ignoring all nodes with bounds within 5% of the initial assignment. The result of this shortened search must either be to eliminate all nodes and thus prove that the initial assignment is satisfactory, or else to find a new assignment which is at least 5% cheaper than the first - this new assignment would then be treated as the initial assignment and the search continued. The procedure yields a sequence of assignments, each at least 5% cheaper than its predecessor and the last within 5% of the optimum.

This technique proves extremely unpredictable in practice and is very dependent on the cost of the initial assignment. Consider the problem of finding an assignment within 2% of the optimum for NVR12. Table 7-2 shows that the initial assignment has cost 293 and so the tree would be enumerated until all pendant nodes exceeded 287.1. Since the actual optimum is 289, this means searching almost the entire tree. On the other hand, if the initial assignment had cost 295 the optimum

might be found, perhaps quite quickly, and then the tree would only be developed up to 283.2; this would save extensive computation. Thus a long computing time may paradoxically correspond to a poor assignment for this technique. For this particular problem, in fact, it would be faster to calculate an assignment within 1% of the optimum than one within 2%!

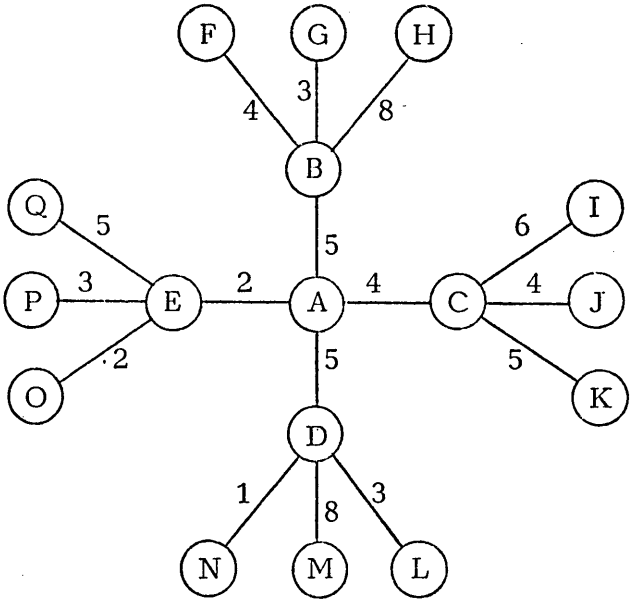
7.5 SPECIAL CASES USING TREE-SEARCH

Chapter 6 presented dynamic programming formulations for the multi-level star mapping and the assembly line problems. This section discusses the efficiency of these algorithms relative to the general tree-search algorithm.

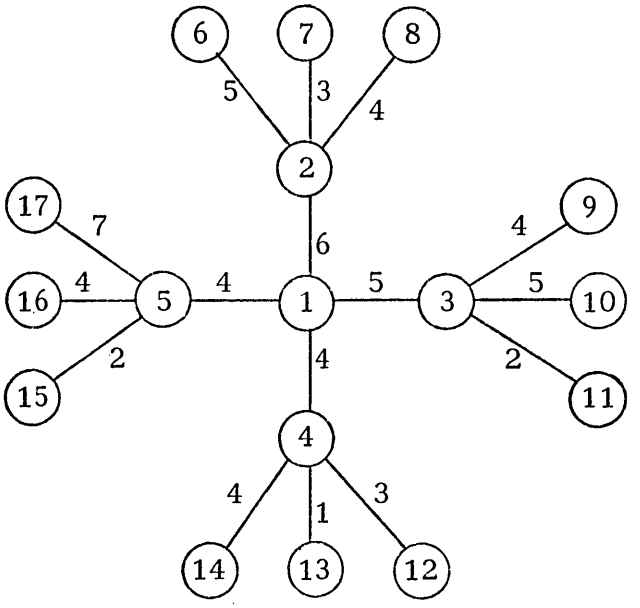
Figure 7-1 shows a 2^{nd} order star problem with 17 machines. This was solved by hand using the dynamic programming formulation in 14 minutes. Presumably a CDC 6400 computer could be programmed to solve it in very much less than one second.

As reported above, LOCATE was unable to prove optimality for a 15-machine problem, even with a generous allocation of computing time. The special structure of the star problem resulted in very tight bounds, however, and the tree-search program used only 5.274 seconds and generated only 121 nodes to solve this 17-machine problem. This still represents far more computation, of course, than the dynamic programming algorithm requires.

Flow Graph



Distance Graph



Optimal Assignment: A1 B4 C3 D5 E2 F12 G14 H13 I11
J10 K9 L16 M15 N17 O6 P8 Q7.

Cost: 228.

Figure 7-1.

A 2nd Order Star Problem.

The algorithm for the assembly line problem was not programmed, but its computing time can be accurately estimated since it is of the same complexity as the dynamic programming formulation for the travelling salesman problem (26). That algorithm has been programmed for a CDC6400 computer and required 18 seconds to solve any 13-city (or 13-machine) problem.

Several assembly line problems have been solved using LOCATE and the time needed to solve the 13-machine problems varied from 39 to 436 seconds. This indicates again that specialised structures can assist a general tree-search algorithm (c.f. NVR12) but that the specialised algorithm is significantly faster.

CHAPTER 8

CONCLUSIONS

8.1 THE LITERATURE

It was not until 1957 that Koopmans and Beckmann (34) first formulated the plant layout problem as a mathematical program and recognised it as being just one example of what they called the quadratic assignment problem. But their quadratic cost function had been accepted in practice long before this formal definition and various manual heuristics had been used to design efficient layouts.

The precise mathematical definition aroused interest in the concept of an optimal layout and in 1962 Lawler (39) and Gilmore (21) independently proposed almost identical tree-search algorithms. But apparently neither author implemented his method and the literature provides no report of computational experience for these algorithms. Other published algorithms appear to be only marginally more efficient than total enumeration of all n! possible assignments.

Most of the vast literature which has appeared in the last decade is concerned with heuristics which only yield sub-optimal assignments. Almost all these heuristics may be classified as either incomplete versions of the tree-search algorithms or as minor variations of the CRAFT 2-opt improvement procedure (4).

Several comparative tests of these heuristics have shown that the improvement methods are superior to the constructive methods in terms of the quality of assignment for a given amount of computer time. A major difficulty in evaluating these heuristics was that there

had been no indication of how sub-optimal their assignments were, because true optima were only known for very small problems - 8 machines or less.

This question was partially resolved recently by Scriabin and Vergin (63). They found that students using the "old-fashioned" technique of travel-charting with the computer only assisting as a calculator were able to design better layouts than the computer using the CRAFT heuristic. For larger problems the difference was as much as 6%. This not only shows that humans can be useful designers, but also that the available heuristics are quite significantly sub-optimal.

Hence further development of algorithms would seem useful because the efficiency of an optimal layout is likely to be much greater than for the best heuristic layouts. Also, the development of new heuristics would benefit from an algorithm which could calculate the optimum and thereby provide an absolute standard for comparison.

8.2 THIS RESEARCH

In the spirit of the previous section, this research tried to develop algorithms for the quadratic assignment problem which would solve moderately large problems. The only report of previous computational experience came from Gavett and Plyter (20) who were unable to solve problems with more than 8 machines.

Chapter 3 tried to solve the problem directly as a 0-1 quadratic program. A standard computer package was able to solve

the program without the 0-1 condition but gave fractional values. The concept of biasing was introduced to eliminate the fractions but the objective function was then non-convex and the resulting assignment was not necessarily optimal; this method proved ineffective, even as a heuristic.

The quadratic program was again transformed, this time by incorporating the constraints into the objective function. Then the variables were automatically zero or one, but feasibility required large penalties whilst convexity required small penalties and a successful compromise was not always possible. The method is, however, a useful heuristic, especially when specifically programmed as described in section 3.8 instead of using the general quadratic programming package.

Chapter 4 was concerned with tree-search algorithms based on those suggested by Gilmore and Lawler. Since neither author has published any results, the aim was first to report computational experience for their algorithms and then to improve on them, particularly by calculating better lower bounds.

Section 4.4 described a tree-search algorithm which uses the bound proposed by Lawler and Gilmore, but a different branching strategy. This program has solved problems with as many as 14 machines, which is compatible with Gilmore's prediction. The time needed to solve a problem is very unpredictable.

The rest of chapter 4 investigated several other lower bounds, some weaker and some stronger than the original. Mathematical proofs have been given to show that these expressions are, in fact, lower bounds.

Evaluation of one of these bounds involves solving a special case of the linear assignment problem. This restricted ranking problem can also arise directly in various practical assignment situations. An extremely efficient algorithm for restricted ranking was developed and tested (sections 4.7 and 4.8).

Chapter 5 continued the analysis of alternative lower bounds using graph theory. The bounds of chapter 4 correspond to particular partial graphs and it was shown that no other partial graphs correspond to useful bounds. This appears to exhaust the possible usefulness of tree-search algorithms.

Most plant layout and other quadratic assignment applications are special cases in some sense, often having many zero coefficients. Chapter 6 has developed solution techniques for several specialised problems, again making use of graph theoretic concepts.

One particular case has been called the assembly line problem and this was formulated as a dynamic program. Another dynamic programming formulation was derived for a hierarchical "star-graph" problem and this is extremely efficient.

Chapter 7 has computationally compared all the methods - heuristics and algorithms, from the literature and this research.

The tree-search algorithm of chapter 4 is the only method which guarantees optimal assignments for more than 8 machines, excluding the highly specific dynamic programming methods. It has solved all problems with less than 15 machines that have been tested, as well as some larger specialised problems.

For large problems there are no known algorithms. Incomplete tree-search can be used as a heuristic and this has been done for as many as 36 machines. Such a method is very flexible in that it designs a good layout very quickly and then may improve it if given more computing time. Using the same time as the best alternative heuristics, incomplete tree-search has produced assignments of the same or marginally superior quality.

8.3 THE FUTURE

It is difficult to predict the direction of future research on the quadratic assignment problem. Many people are very interested in it but the way ahead seems blocked at many points.

The optimal procedures described in this thesis are limited to only about 14 machines and this is disappointing. And yet no one has been able to suggest an alternative to tree-search and chapter 5 seems to account for all useful bounds.

There has been little improvement in heuristics in the last 6 or 8 years and all the extensive research has concentrated on

refinement of just one technique. Further improvement seems to require a "break-through", a new concept on which to build another heuristic.

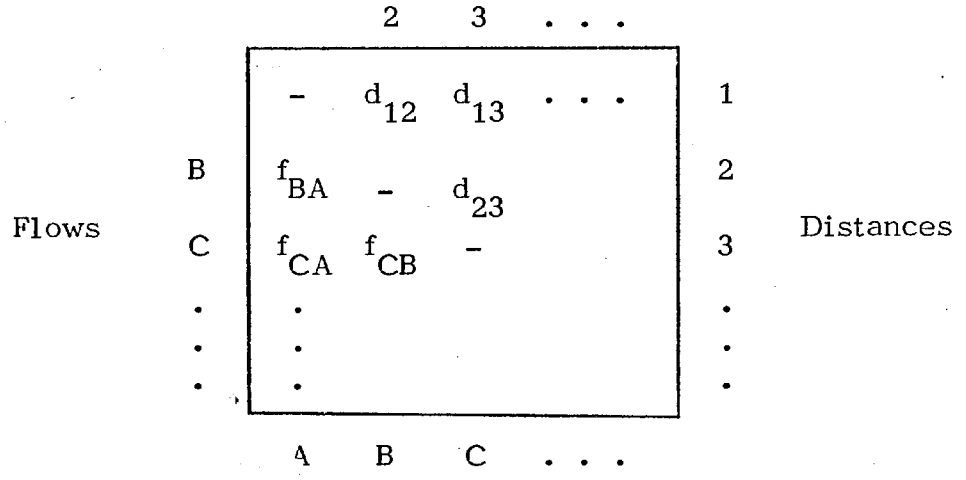
The most likely direction of research in the immediate future is probably that of special cases. Almost all applications of the quadratic assignment problem have a specialised pattern or structure and this can often simplify the tasks of finding both optimal and sub-optimal assignments.

APPENDICES

APPENDIX A DATA FOR TEST PROBLEMS

This appendix gives the data for all the problems mentioned in the text. Where known the optimum cost, Z^*, and an optimal assignment are also given. The problems are labelled according to the author's name and the number of machines. Thus LA7 is Lawler's 7-machine problem, and MG12 is the 12-machine problem proposed by this author.

If all the linear costs are zero, as for Nugent's problems, then the linear cost matrix is omitted. Since all problems have symmetric flow and distance matrices it is convenient to combine them as shown, putting the distances above the diagonal and the flows below.



MG3

		2	3	
	-	2	5	1
B	3	-	6	2
C	1	4	-	
	A	B		

Z*=29 : A3 B1 C2.

MG10

	2	3	4	5	6	7	8	9	10	
-	4	6	2	7	1	7	8	8	6	1
B 5	-	3	4	5	2	9	4	5	5	2
C 4	6	-	7	3	1	4	4	3	8	3
D 8	3	1	-	8	3	6	2	1	7	4
E 3	9	9	8	-	7	3	3	2	4	5
F 4	5	2	3	8	-	5	4	1	3	6
G 8	8	5	6	2	4	-	6	3	9	7
H 7	4	2	6	5	7	9	-	4	9	8
I 1	1	3	9	1	1	3	5	-	9	9
J 7	9	6	8	9	9	7	1	3	-	
	A	B	C	D	E	F	G	H	I	

Linear costs

	1	2	3	4	5	6	7	8	9	10
A	25	57	34	11	34	29	15	13	22	61
B	45	35	10	3	54	68	41	21	19	20
C	0	71	45	23	14	26	45	17	42	54
D	13	32	46	52	74	36	28	5	2	33
E	16	27	38	41	62	33	49	40	10	1
F	6	13	26	41	32	68	52	37	44	51
G	32	41	52	36	19	99	21	30	41	12
H	27	41	23	53	61	25	43	23	16	19
I	38	16	28	21	34	60	10	7	16	12
J	49	20	37	16	21	31	16	10	1	33

Z*=1092

A8 B10 C1 D9 E6
F2 G5 H3 I7 J4.

MG12

	2	3	4	5	6	7	8	9	10	11	12	
	-	6	3	1	4	5	8	9	4	2	2	1	1
B	3	-	6	6	6	3	3	2	2	4	5	5	2
C	5	2	-	3	2	4	7	4	5	6	7	3	3
D	7	1	3	-	8	1	4	1	3	7	4	4	4
E	3	5	4	7	-	3	6	4	4	3	6	2	5
F	1	3	2	6	4	-	5	4	1	1	3	6	6
G	9	8	3	4	6	1	-	4	6	2	4	8	7
H	4	2	6	4	7	7	5	-	2	5	2	9	8
I	3	1	1	2	1	4	3	5	-	6	5	4	9
J	3	6	5	4	7	3	5	4	6	-	8	3	10
K	2	43	3	6	5	1	8	3	9	5	-	5	11
L	1	4	7	8	5	2	4	3	5	4	4	-	
	A	B	C	D	E	F	G	H	I	J	K		

	1	2	3	4	5	6	7	8	9	10	11	12
A	15	36	47	62	81	40	35	24	71	66	83	23
B	15	62	35	87	68	46	23	15	36	45	81	40
C	81	23	64	90	37	72	68	43	27	82	46	11
D	43	0	10	46	38	91	20	49	62	36	72	46
E	83	17	56	3	19	91	77	38	50	38	26	31
F	1	10	47	53	82	93	86	14	55	21	51	39
G	34	40	10	72	39	11	15	30	22	61	1	0
H	23	11	73	49	82	83	80	65	72	15	43	10
I	96	84	59	7	38	47	16	32	95	7	38	21
J	93	6	48	36	74	20	81	11	37	67	89	32
K	71	28	46	73	28	29	74	61	40	83	28	95
L	0	27	46	53	92	1	36	48	33	89	56	25

Z*=1241 : A1 B7 C9 D3 E4 F2 G11 H12 I10 J8 K5 L6.

MG14

	2	3	4	5	6	7	8	9	10	11	12	13	14	
	-	8	2	1	5	3	9	3	7	8	7	4	3	7	1
B	3	-	7	7	3	6	3	7	6	3	2	3	7	5	2
C	5	2	-	3	7	4	8	4	4	6	4	7	2	6	3
D	1	5	4	-	2	5	6	6	1	2	5	2	8	3	4
E	4	5	2	3	-	2	3	2	3	7	1	8	9	5	5
F	5	7	2	3	5	-	1	4	2	4	8	4	7	4	6
G	1	5	3	6	4	5	-	3	5	9	3	7	8	6	7
H	2	4	3	5	4	8	2	-	8	3	4	5	4	2	8
I	2	5	4	7	6	3	4	9	-	7	5	9	1	7	9
J	3	4	6	5	7	3	4	5	3	-	3	1	4	3	10
K	8	4	5	6	3	4	7	5	3	1	-	2	3	4	11
L	7	3	8	4	6	2	1	1	7	3	2	-	2	8	12
M	3	8	2	2	2	6	3	1	7	3	8	4	-	2	13
N	3	2	7	1	2	5	4	2	5	3	9	4	5	-	
	A	B	C	D	E	F	G	H	I	J	K	L	M		

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	81	27	36	47	58	91	70	37	26	38	46	51	26	37
B	50	48	10	26	38	64	72	38	49	71	1	39	20	57
C	72	47	38	74	63	19	3	47	65	28	39	71	8	39
D	1	27	38	46	28	49	88	42	74	56	91	22	36	48
E	83	46	58	16	37	28	37	49	71	40	30	91	46	58
F	92	47	58	37	61	38	40	86	28	49	33	10	6	59
G	89	44	37	28	19	73	46	58	29	37	64	51	83	59
H	28	37	46	19	47	77	33	28	46	10	82	64	52	89
I	21	90	32	89	56	37	83	21	89	96	37	48	19	31
J	74	36	28	39	16	59	37	18	28	39	47	61	28	42
K	20	38	47	59	81	47	39	28	46	37	82	21	27	29
L	90	48	39	31	34	32	18	64	1	83	49	96	58	47
M	84	38	29	47	38	19	38	64	58	73	28	22	21	28
N	64	57	37	44	24	46	48	75	53	47	62	38	47	49

Z*=1873 : A3 B11 C7 D2 E4 F12 G5 H10 I14 J8 K1 L9 M13 N6.

TSP10

	2	3	4	5	6	7	8	9	10	
	-	3	6	9	9	4	3	2	4	9	1
B	1	-	4	9	9	9	9	4	3	9	2
C	0	1	-	3	9	9	9	9	2	5	3
D	0	0	1	-	2	9	9	9	4	3	4
E	0	0	0	1	-	1	9	4	4	3	5
F	0	0	0	0	1	-	2	2	9	3	6
G	0	0	0	0	0	1	-	2	9	2	7
H	0	0	0	0	0	0	1	-	4	2	8
I	0	0	0	0	0	0	0	1	-	1	9
J	1	0	0	0	0	0	0	0	1	-	
	A	B	C	D	E	F	G	H	I		

Z*=22 : A1 B2 C9 D3 E4 F5 G6 H7 I10 J8.

GP4

	2	3	4	
	-	6	7	2	1
B	28	-	5	6	2
C	25	15	-	1	3
D	13	4	23	-	
	A	B	C		

Z*= 403 : A4 B1 C3 D2.

LA7

	2	3	4	5	6	7	
	-	5	0	5	0	5	4	1
B	0	-	9	7	3	8	6	2
C	12	2	-	9	4	4	4	3
D	2	0	16	-	1	1	9	4
E	2	6	16	14	-	5	5	5
F	16	2	8	12	0	-	4	6
G	8	6	4	8	12	18	-	
	A	B	C	D	E	F		

	1	2	3	4	5	6	7
A	51	27	14	9	0	18	0
B	0	1	22	17	0	41	13
C	2	0	13	22	2	12	27
D	38	11	0	0	22	13	14
E	62	56	0	67	1	0	5
F	61	0	3	14	9	1	67
G	41	12	23	0	18	41	0

Z*=559 : A7 B2 C1 D3 E5 F6 G4.

Nugent et al. used a slightly different notation, numbering the machines as well as the locations. Their five larger problems are reproduced here directly from their paper (57) and so the machine-labels are different, but the layout of the entries is as for the other problems.

NVR8

	1	2	3	4	5	6	7	8
1	—	1	2	3	1	2	3	4
2	5	—	1	2	2	1	2	3
3	2	3	—	1	3	2	1	2
4	4	0	0	—	4	3	2	1
5	1	2	0	5	—	1	2	3
6	0	2	0	2	10	—	1	2
7	0	2	0	2	0	5	—	1
8	6	0	5	10	0	1	10	—

Z*=107

A6 B5 C1 D7
E8 F4 G3 H2.

NVR12

	1	2	3	4	5	6	7	8	9	10	11	12
1	—	1	2	3	1	2	3	4	2	3	4	5
2	5	—	1	2	2	1	2	3	3	2	3	4
3	2	3	—	1	3	2	1	2	4	3	2	3
4	4	0	0	—	4	3	2	1	5	4	3	2
5	1	2	0	5	—	1	2	3	1	2	3	4
6	0	2	0	2	10	—	1	2	2	1	2	3
7	0	2	0	2	0	5	—	1	3	2	1	2
8	6	0	5	10	0	1	10	—	4	3	2	1
9	2	4	5	0	0	1	5	0	—	1	2	3
10	1	5	2	0	5	5	2	0	0	—	1	2
11	1	0	2	5	1	4	3	5	10	5	—	1
12	1	0	2	5	1	0	3	0	10	0	2	—

Z*=289

A5 B9 C1 D8 E6 F2 G5 H3 I7 J4

NVR15

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

Best known cost is 575.

NVR20

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

Best known cost is 1282.

NVR30

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
1	—	1	2	3	4	5	1	2	3	4	5	6	2	3	4	5	6	7	3	4	5	6	7	8	4	5	6	7	8	9
2	3	—	1	2	3	4	2	1	2	3	4	5	3	2	3	4	5	6	4	3	4	5	6	7	5	4	5	6	7	8
3	2	4	—	1	2	3	3	2	1	2	3	4	4	3	2	3	4	5	5	4	3	4	5	6	6	5	4	5	6	7
4	0	0	3	—	1	2	4	3	2	1	2	3	5	4	3	2	3	4	6	5	4	3	4	5	7	6	5	4	5	6
5	0	10	4	0	—	1	5	4	3	2	1	2	6	5	4	3	2	3	7	6	5	4	3	4	8	7	6	5	4	5
6	2	4	0	0	5	—	6	5	4	3	2	1	7	6	5	4	3	2	8	7	6	5	4	3	9	8	7	6	5	4
7	10	0	5	0	2	1	—	1	2	3	4	5	1	2	3	4	5	6	2	3	4	5	6	7	3	4	5	6	7	8
8	5	0	5	2	0	2	10	—	1	2	3	4	2	1	2	3	4	5	3	2	3	4	5	6	4	3	4	5	6	7
9	0	2	5	2	0	2	10	1	—	1	2	3	3	2	1	2	3	4	4	3	2	3	4	5	5	4	3	4	5	6
10	5	2	1	0	0	1	5	3	10	—	1	2	4	3	2	1	2	3	5	4	3	2	3	4	6	5	4	3	4	5
11	2	1	4	6	0	4	10	5	2	5	—	1	5	4	3	2	1	2	6	5	4	3	2	3	7	6	5	4	3	4
12	5	0	1	0	2	10	10	0	1	5	0	—	6	5	4	3	2	1	7	6	5	4	3	2	8	7	6	5	4	3
13	0	5	0	2	0	10	6	0	5	6	0	5	—	1	2	3	4	5	1	2	3	4	5	6	2	3	4	5	6	7
14	0	0	4	5	0	2	0	0	2	0	1	5	2	—	1	2	3	4	2	1	2	3	4	5	3	2	3	4	5	6
15	2	0	0	2	0	5	0	2	0	1	2	2	0	2	—	1	2	3	3	2	1	2	3	4	4	3	2	3	4	5
16	0	0	4	5	0	5	10	4	3	5	1	0	4	1	4	—	1	2	4	3	2	1	2	3	5	4	3	2	3	4
17	5	0	0	1	2	0	2	5	0	5	0	0	2	0	5	0	—	1	5	4	3	2	1	2	6	5	4	3	2	3
18	6	2	6	1	1	5	1	2	2	0	2	0	2	5	1	3	2	—	6	5	4	3	2	1	7	6	5	4	3	2
19	3	0	3	1	0	0	10	10	0	5	0	0	1	3	0	0	2	5	—	1	2	3	4	5	1	2	3	4	5	6
20	0	1	2	1	0	0	1	6	0	2	0	2	0	10	1	2	0	1	0	—	1	2	3	4	2	1	2	3	4	5
21	1	6	5	2	2	0	5	0	4	3	0	0	6	0	0	2	0	2	5	5	—	1	2	3	3	2	1	2	3	4
22	10	1	5	2	0	10	5	5	0	5	6	4	2	0	5	0	0	10	5	2	4	—	1	2	4	3	2	1	2	3
23	0	0	2	4	5	0	2	5	5	0	6	5	1	4	0	2	6	10	1	1	0	5	—	1	5	4	3	2	1	2
24	10	1	1	0	1	0	3	2	2	5	0	10	5	2	2	0	5	4	0	3	1	0	0	—	6	5	4	3	2	1
25	2	2	0	2	0	0	5	5	0	2	4	1	5	0	0	5	3	0	5	1	0	4	4	5	—	1	2	3	4	5
26	1	2	0	0	2	4	0	0	5	10	5	0	0	0	0	0	5	0	2	5	0	4	4	5	1	—	1	2	3	4
27	1	5	3	2	1	0	2	5	2	10	3	0	0	4	5	5	0	5	1	6	0	5	1	0	0	0	—	1	2	3
28	1	1	1	2	0	10	0	5	2	1	2	0	1	2	1	2	0	0	2	5	5	0	0	1	10	0	0	—	1	2
29	0	10	0	5	2	1	1	0	5	5	2	0	5	5	1	5	5	0	10	5	0	2	2	0	1	0	0	2	—	1
30	1	5	2	5	1	1	3	2	2	2	10	1	5	5	0	10	1	0	10	3	0	5	2	0	0	0	10	2	2	—

Best known cost is 3086.

ST36

The 34 modules must be placed in 4 rows of 9, with any two locations empty. The distance between locations is the straight-line or Euclidean distance. The "flows" in the matrix are the numbers of wires between each pair of modules.

	E1	E2	E3	E4	E5	E6	E7	E8	E9	E10	E11	E12	E13	E14	E15	E16	E17	E18	E19	E20	E21	E22	E23	E24	E25	E26	E27	E28	E29	E30	E31	E32	E33	E34	TOTAL	*
E1				2	1	7	9		4	75	7	12	22	7	1					23															170	12
E2							4	16		9			16					6		4															54	6
E3						4	16	20					20							4															64	5
E4					29	5	19	47	23	2	4		118		4					25															207	11
E5						18	12	25			4		25		3					18		3													138	10
E6							4	2		1	23	2	19						2	19															106	12
E7									14	72	7	8	39	8	40	8		8	4	7								28	8						314	19
E8									10	71	2							41									7	8							249	11
E9										14			18																						83	6
E10											11	1	17		1			17		15															305	13
E11												316	33	8	2				8	34			6				10			6					481	16
E12													157	25	11			1							22		1								549	11
E13														11	6			6		5	8	3	10			7	11	2				1			486	22
E14															3			1	1	21		1		2			5			3	2	5	5	4	112	17
E15																19		2	2	12								7	3						109	15
E16																		6		1															34	4
E17																		40																	40	1
E18																				26															154	11
E19																				13	9		7					27	16	3		20		4	116	13
E20																					11	4	36				16	13	9	10	1	28	6	2	368	26
E21																						36	6		8		2								80	7
E22																											4								51	6
E23																										12	9								86	7
E24																									26		5								53	5
E25																										35	2								93	5
E26																											4								51	3
E27																																			74	12
E28																												10	22	4	6	4	12		157	12
E29																													19	12					79	8
E30																														19	4	5	8		99	10
E31																																3	13		54	7
E32																																18	24		106	8
E33																																	20		61	7
E34																																			87	8

TOTAL NUMBER OF WIRES =2620

*NUMBER OF ELEMENTS TO WHICH EACH IS CONNECTED

Best known cost is 4327.84 .

APPENDIX B BIASED QUADRATIC PROGRAMMING EXAMPLES

An optimal assignment for NVR8 is A6 B5 C1 D7 E8 F4 G3 H2 with cost 107. The iterations of the biasing procedure described in section 3.5 are given below.

With no bias the QPS package gives Z = 101.57,

$$X = \begin{array}{c} \\ A \\ B \\ C \\ D \\ E \\ F \\ G \\ H \end{array} \begin{array}{c} \begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \\ \begin{bmatrix} & & & & .20 & .80 & & \\ .19 & & & & .61 & .20 & & \\ .81 & & & & .19 & & & \\ & .20 & .04 & & & & .77 & \\ & & & .30 & & & & .70 \\ & & & .70 & & & & .30 \\ & .20 & .80 & & & & & \\ & .60 & .16 & & & & .23 & \end{bmatrix} \end{array}$$

There are 19 fractional variables and so the bias coefficients should be $\frac{5(107-101.57)}{19} = 1.39$. (see flow-chart in figure 3-5). For convenience the value 1.0 was used, with the following result:

$$B = \begin{array}{c} \\ A \\ B \\ C \\ D \\ E \\ F \\ G \\ H \end{array} \begin{array}{c} \begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \\ \begin{bmatrix} & & & & 1 & 1 & & \\ 1 & & & & 1 & 1 & & \\ 1 & & & & 1 & & & \\ & 1 & 1 & & & & 1 & \\ & & & 1 & & & & 1 \\ & & & 1 & & & & 1 \\ & 1 & 1 & & & & & \\ & 1 & 1 & & & & 1 & \end{bmatrix} \end{array} \qquad X = \begin{array}{c} \begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \\ \begin{bmatrix} & & & & & 1 & & \\ & & & & 1 & & & \\ 1 & & & & & & & \\ & .17 & & & & & .83 & \\ & & & .23 & & & & .77 \\ & & & .77 & & & & .23 \\ & .16 & .84 & & & & & \\ & .67 & .16 & & & & .17 & \end{bmatrix} \end{array}$$

$$Z = 104.05$$

Now there are 11 fractional values and this means that the bias coefficients should be increased by 1.32 ; again 1.0 was used.

$$B = \begin{array}{c} \\ A \\ B \\ C \\ D \\ E \\ F \\ G \\ H \end{array} \begin{array}{c} \begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \\ \begin{bmatrix} & & & & 1 & 1 & & \\ 1 & & & & 1 & 1 & & \\ 1 & & & & 1 & & & \\ & 2 & 1 & & & & 2 & \\ & & & 2 & & & & 2 \\ & & & 2 & & & & 2 \\ & 2 & 2 & & & & & \\ & 2 & 2 & & & & 2 & \end{bmatrix} \end{array} \qquad X = \begin{array}{c} \begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \\ \begin{bmatrix} & & & & & 1 & & \\ & & & & 1 & & & \\ 1 & & & & & & & \\ & .18 & & & & & .82 & \\ & & & .14 & & & & .86 \\ & & & .86 & & & & .14 \\ & .05 & .95 & & & & & \\ & .77 & .05 & & & & .18 & \end{bmatrix} \end{array}$$

$$Z = 105.60$$

For this solution the same values as before are fractional, but most of them have become nearer to being integers. The bias should be increased by 0.67, but 1.0 was more convenient.

$$B = \begin{array}{c} \\ A \\ B \\ C \\ D \\ E \\ F \\ G \\ H \end{array} \begin{array}{c} \begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \\ \begin{bmatrix} & & & & 1 & 1 & & \\ 1 & & & & 1 & 1 & & \\ 1 & & & & 1 & & & \\ & 3 & 1 & & & & 3 & \\ & & & 3 & & & & 3 \\ & & & 3 & & & & 3 \\ & 3 & 3 & & & & & \\ & 3 & 3 & & & & 3 & \end{bmatrix} \end{array} \qquad X = \begin{array}{c} \begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \\ \begin{bmatrix} & & & & .19 & .81 & & \\ & .19 & & & .81 & & & \\ 1 & & & & & & & \\ & & & & & .19 & .81 & \\ & & & & & & & 1 \\ & & & 1 & & & & \\ & & 1 & & & & & \\ & .81 & & & & & .19 & \end{bmatrix} \end{array}$$

$$Z = 106.32$$

Now some of the 0-1 variables have become fractional and vice versa. The formula for bias increment gives 0.43, but since all fractions are so near zero or one, the effect of additional bias is small and so a larger value can be used - e.g. the convenient 1.0.

$$B = \begin{array}{c} \\ A \\ B \\ C \\ D \\ E \\ F \\ G \\ H \end{array} \begin{array}{c} \begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \\ \begin{bmatrix} & & & & 2 & 2 & & \\ 1 & 1 & & & 2 & 1 & & \\ 1 & & & & 1 & & & \\ & 3 & 1 & & & 1 & 4 & \\ & & & 3 & & & & 3 \\ & & & 3 & & & & 3 \\ & 3 & 3 & & & & & \\ & 4 & 3 & & & & 4 & \end{bmatrix} \end{array} \qquad X = \begin{array}{c} \begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \\ \begin{bmatrix} & & & & & 1 & & \\ & & & & 1 & & & \\ 1 & & & & & & & \\ & .10 & & & & & .90 & \\ & & & & & & & 1 \\ & & & 1 & & & & \\ & & 1 & & & & & \\ & .90 & & & & & .70 & \end{bmatrix} \end{array}$$

$$Z = 106.75$$

Again the formula suggests a low value, 0.31, for the bias increment. But the extreme values of the fractions mean that the bias needed to give this solution the cost 107 is actually much higher, 0.70. Hence 1.0 was used again.

$$B = \begin{array}{c} \\ A \\ B \\ C \\ D \\ E \\ F \\ G \\ H \end{array} \begin{array}{c} \begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \\ \begin{bmatrix} & & & & 2 & 2 & & \\ 1 & 1 & & & 2 & 1 & & \\ 1 & & & & 1 & & & \\ & 4 & 1 & & & 1 & 5 & \\ & & & 3 & & & & 3 \\ & & & 3 & & & & 3 \\ & 3 & 3 & & & & & \\ & 5 & 3 & & & & 5 & \end{bmatrix} \end{array} \qquad X = \begin{array}{c} \begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \\ \begin{bmatrix} & & & & & 1 & & \\ & & & & 1 & & & \\ 1 & & & & & & & \\ & & & & & & 1 & \\ & & & & & & & 1 \\ & & & 1 & & & & \\ & .08 & .92 & & & & & \\ & .92 & .08 & & & & & \end{bmatrix} \end{array}$$

$$Z = 106.96$$

This is very nearly integral and only one further iteration was needed to give the optimal solution.

$$B = \begin{array}{c} \\ A \\ B \\ C \\ D \\ E \\ F \\ G \\ H \end{array} \begin{array}{c} \begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \\ \begin{bmatrix} & & & & 2 & 2 & & \\ 1 & 1 & & & 2 & 1 & & \\ 1 & & & & 1 & & & \\ & 4 & 1 & & & 1 & 5 & \\ & & & 3 & & & & 3 \\ & & & 3 & & & & 3 \\ & 4 & 4 & & & & & \\ & 6 & 4 & & & & 5 & \end{bmatrix} \end{array} \qquad X = \begin{array}{c} \begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \\ \begin{bmatrix} & & & & & 1 & & \\ & & & & 1 & & & \\ 1 & & & & & & & \\ & & & & & & 1 & \\ & & & & & & & 1 \\ & & & 1 & & & & \\ & & 1 & & & & & \\ & 1 & & & & & & \end{bmatrix} \end{array}$$

$$Z = 107.$$

Here is another example of the heuristic biasing procedure given in section 3.5 . The problem is GP4, Gavett and Plyter's 4-machine example, for which the unique optimal assignment is A4 B1 C3 D2 with cost 403. The formula given in section 3.5 is followed accurately and so no detailed working need be shown.

With no bias the QPS package gives:

$$X = \begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} & .14 & & .86 \\ .36 & & .64 & \\ .30 & .28 & .28 & .14 \\ .34 & .58 & .08 & \end{bmatrix} \end{array} \qquad Z = 341.47$$

$$B = \begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} & 20 & & 20 \\ 20 & & 20 & \\ 20 & 20 & 20 & 20 \\ 20 & 20 & 20 & \end{bmatrix} \end{array} \quad X = \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} .27 & .29 & .45 & \\ & .71 & & .29 \\ .45 & & .55 & \\ .28 & & & .71 \end{bmatrix} \end{array} \qquad Z = 390.72$$

$$B = \begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 6 & 26 & 6 & 20 \\ 20 & 6 & 20 & 6 \\ 26 & 20 & 26 & 20 \\ 26 & 20 & 20 & 6 \end{bmatrix} \end{array} \quad X = \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} & & & 1 \\ .38 & & .62 & \\ .42 & .20 & .38 & \\ .20 & .80 & & \end{bmatrix} \end{array} \qquad Z = 377.60$$

Note that Z has decreased and so the biased function was sufficiently non-convex at the previous iteration to have a non-optimal local minimum.

$$B = \begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 6 & 26 & 6 & 20 \\ 34 & 6 & 34 & 6 \\ 40 & 34 & 40 & 20 \\ 40 & 34 & 20 & 6 \end{bmatrix} \end{array} \qquad X = \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} & .21 & & .79 \\ & & 1 & \\ .38 & .42 & & .21 \\ .62 & .38 & & \end{bmatrix} \end{array} \qquad Z = 396.79$$

$$B = \begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 6 & 30 & 6 & 24 \\ 34 & 6 & 34 & 6 \\ 44 & 38 & 40 & 24 \\ 44 & 38 & 20 & 6 \end{bmatrix} \end{array} \qquad X = \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} & & & 1 \\ .64 & & .36 & \\ .36 & & .64 & \\ & 1 & & \end{bmatrix} \end{array} \qquad Z = 395.40$$

This solution is almost optimal.

$$B = \begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 6 & 30 & 6 & 24 \\ 44 & 6 & 44 & 6 \\ 54 & 38 & 50 & 24 \\ 44 & 38 & 20 & 6 \end{bmatrix} \end{array} \qquad X = \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} & .39 & .59 & .02 \\ & .59 & .41 & \\ & .02 & & .98 \\ 1 & & & \end{bmatrix} \end{array} \qquad Z = 385.81$$

The objective has been reduced again!

$$B = \begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 6 & 42 & 18 & 36 \\ 44 & 18 & 56 & 6 \\ 54 & 50 & 50 & 36 \\ 44 & 38 & 20 & 6 \end{bmatrix} \end{array} \qquad X = \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} & & & 1 \\ & & 1 & \\ .64 & .36 & & \\ .36 & .64 & & \end{bmatrix} \end{array} \qquad Z = 407.62$$

The cost of this solution is actually greater than that of the optimal assignment and the procedure is clearly out of control anyway since the sequence of solutions oscillates wildly. The procedure was continued

using somewhat arbitrary bias increments for the final four iterations. The resultant assignment is quite unsatisfactory.

$$B = \begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 10 & 62 & 30 & 44 \\ 44 & 30 & 68 & 6 \\ 58 & 62 & 50 & 44 \\ 52 & 42 & 24 & 6 \end{bmatrix} \end{array} \qquad X = \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} & & & 1 \\ & & 1 & \\ & 1 & & \\ 1 & & & \end{bmatrix} \end{array} \qquad Z = 445.00$$

- - - - - - - - - - -

The unique optimal assignment for Lawler's 7-machine example, LA7, is A7 B2 C1 D3 E5 F6 G4 with cost 559. The biased quadratic programming procedure found this most easily by starting with all bias coefficients set to 10. This leads to:

$$B = \begin{array}{c} \\ A \\ B \\ C \\ D \\ E \\ F \\ G \end{array} \begin{array}{c} \begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \\ \begin{bmatrix} 10 & 10 & 10 & 10 & 10 & 10 & 10 \\ 10 & 10 & 10 & 10 & 10 & 10 & 10 \\ 10 & 10 & 10 & 10 & 10 & 10 & 10 \\ 10 & 10 & 10 & 10 & 10 & 10 & 10 \\ 10 & 10 & 10 & 10 & 10 & 10 & 10 \\ 10 & 10 & 10 & 10 & 10 & 10 & 10 \\ 10 & 10 & 10 & 10 & 10 & 10 & 10 \end{bmatrix} \end{array} \qquad X = \begin{array}{c} \begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \\ \begin{bmatrix} & & & .36 & & & .64 \\ & 1 & & & & & \\ 1 & & & & & & \\ & & 1 & & & & \\ & & & & 1 & & \\ & & & & & 1 & \\ & & & .64 & & & .36 \end{bmatrix} \end{array}$$

$$Z = 546.60$$

Note that simple rounding at this stage would give the optimal assignment. Continuing with the biasing procedure :

$$B = \begin{array}{c} \\ A \\ B \\ C \\ D \\ E \\ F \\ G \end{array} \begin{array}{c} \begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \\ \begin{bmatrix} 10 & 10 & 10 & 35 & 10 & 10 & 35 \\ 10 & 10 & 10 & 10 & 10 & 10 & 10 \\ 10 & 10 & 10 & 10 & 10 & 10 & 10 \\ 10 & 10 & 10 & 10 & 10 & 10 & 10 \\ 10 & 10 & 10 & 10 & 10 & 10 & 10 \\ 10 & 10 & 10 & 10 & 10 & 10 & 10 \\ 10 & 10 & 10 & 35 & 10 & 10 & 35 \end{bmatrix} \end{array} \quad X = \begin{array}{c} \begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \\ \begin{bmatrix} & & & & & & 1 \\ & 1 & & & & & \\ 1 & & & & & & \\ & & .99 & .01 & & & \\ & & & & 1 & & \\ & & & & & 1 & \\ & & .01 & .99 & & & \end{bmatrix} \end{array}$$

$$Z = 558.986$$

$$B = \begin{array}{c} \\ A \\ B \\ C \\ D \\ E \\ F \\ G \end{array} \begin{array}{c} \begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \\ \begin{bmatrix} 10 & 10 & 10 & 35 & 10 & 10 & 35 \\ 10 & 10 & 10 & 10 & 10 & 10 & 10 \\ 10 & 10 & 10 & 10 & 10 & 10 & 10 \\ 10 & 10 & 15 & 15 & 10 & 10 & 10 \\ 10 & 10 & 10 & 10 & 10 & 10 & 10 \\ 10 & 10 & 10 & 10 & 10 & 10 & 10 \\ 10 & 10 & 15 & 40 & 10 & 10 & 10 \end{bmatrix} \end{array} \quad X = \begin{array}{c} \begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \\ \begin{bmatrix} & & & & & & 1 \\ & 1 & & & & & \\ 1 & & & & & & \\ & & 1 & & & & \\ & & & & 1 & & \\ & & & & & 1 & \\ & & & 1 & & & \end{bmatrix} \end{array}$$

$$Z = 559.$$

APPENDIX C PROGRAM "PENALTY"

The program reads the problem date from TAPE 1 and minimises Z with the penalties zero and starting with the infeasible solution, $x_{\propto i}=0$; if all cost coefficients are non-negative this zero solution is optimal already. Then it prints Z and awaits directives from the following list.

Change	Changes specified variables from 0 to 1 or vice versa.
Alter	Increases specified penalties by specified amounts.
Alter +	Increases all penalties by specified amount.
Go	Minimises with the current parameters. A blank line has the same effect.
Step	Increments all penalties associated with unsatisfied constraints by a specified amount, minimises, and repeats until a feasible solution is found or the specified limit is reached.
Monitor	Resets print control to give more or less information.
Tape	Further directives are read from specified tape.
Finish	Terminates program.
Print (list)	Prints selected information according to the sub-directives in the list. Possible sub-directives are:
Solution	Current variables and Z.
Penalties	Current machine and location penalties.
Critical	The minimum increase and decrease in the specified penalties which will change the solution.

Derivative The partial derivatives of Z with respect to the specified variables; also the change in penalty needed move the variable in or out of the solution.

Many directives can be typed on the same line, but "finish", "go" and "step" are terminal. As an abbreviation, only the first three letters of each directive need be typed.

The example below shows the difficulty of choosing suitable penalties for the 15-machine problem. The notes on the right have been added later. The cost of the best known solution is 575.

```
lgo.,,nvr15
 INFEASIBLE     Z =          0
? alter + 20 go
 INFEASIBLE     Z =    392.000
? alter + 25 go
 INFEASIBLE     Z =    618.000
? print solution
 CURRENT SOLUTION     A 1   B 2   C 3   D 4   E11   F 6   G 8
   K13   L10   014                                H12   I 7   J 9
 Z =    618.000
? print critical m n 5 15
 ZEROS FOR MACH M  ARE  61.000(LOCN 7)*-.9E+10(LOCN 0)
 ZEROS FOR MACH N  ARE  33.000(LOCN 5)*-.9E+10(LOCN 0)
 ZEROS FOR LOCN 5  ARE  33.000(MACH N)*-.9E+10(MACH  )
 ZEROS FOR LOCN15  ARE  35.000(MACH N)*-.9E+10(MACH  )
? alter n 34 m 30  5 10   15 12 go
 INFEASIBLE     Z =    622.000
? print solution
 CURRENT SOLUTION     A 1   B 2   C 3   D 4   F13   G 6   H12
    N 5   014                                 I 7   J 9   L10
 Z =    622.000
? fin
STOP
/
```

Penalty = 20 is too small. But 45 is too large and so the variables in this infeasible solution are rather arbitrary.

M & N are missing.

Obviously nowhere near a good assignment and so this run was abandoned.

PENALTY being used for NVR8. Optimum is 107.

```
lgo,,,nvr8
 INFEASIBLE     Z =          0
? step 100 3
  78 INCREMENTS NOT NEEDED.
   FEASIBLE     Z =    116.000
? print solution penalties
 CURRENT SOLUTION     A 1   B 7   C 5   D 6   E 8   F 4   G 3   H 2
 Z =    116.000
 MACHINE PENALTIES          A 15.000   B  9.000   C  9.000   D 21.000
   F 21.000   G 15.000   H 42.000
 LOCATION PENALTIES         1 21.000   2  6.000   3 12.000   4 18.000
   6 18.000   7 21.000   8 30.000
? alter + -2 go
 INFEASIBLE     Z =    113.000
? print solution
 CURRENT SOLUTION     B 7   C 5   D 6   E 8   F 4   G 3   H 2
 Z =    113.000
? print critical a 1
 ZEROS FOR MACH A  ARE    3.000(LOCN 1)*-.9E+10(LOCN 6)
 ZEROS FOR LOCN 1  ARE    2.000(MACH C)*-.9E+10(MACH   )
? alter 1 3  go
 INFEASIBLE     Z =    115.000
? print solution
 CURRENT SOLUTION     B 7   C 1   C 5   D 6   E 8   F 4   G 3   H 2
 Z =    115.000
? print critical a c
 ZEROS FOR MACH A  ARE   14.000(LOCN 2)*-.9E+10(LOCN 6)
 ZEROS FOR MACH C  ARE    1.000(LOCN 1) -10.333(LOCN 2)
? alter c 2 go
 INFEASIBLE     Z =    116.000
? print solution
 CURRENT SOLUTION     B 7   C 5   D 6   E 8   F 4   G 3   H 2
 Z =    116.000
? print critical a
 ZEROS FOR MACH A  ARE   12.000(LOCN 2)         0(LOCN 1)
? alter a 13 go
 INFEASIBLE     Z =    110.000
? step 20 1
  16 INCREMENTS NOT NEEDED.
   FEASIBLE     Z =    116.000
? print solution
 CURRENT SOLUTION     A 1   B 7   C 5   D 6   E 8   F 4   G 3   H 2
 Z =    116.000
? alter + -3 go
 INFEASIBLE     Z =    106.000
? step 50 1
  32 INCREMENTS NOT NEEDED.
   FEASIBLE     Z =    117.000
? print solution
 CURRENT SOLUTION     A 1   B 5   C 6   D 7   E 8   F 4   G 3   H 2
 Z =    117.000
? finish
STOP
/
```

APPENDIX D INTERACTIVE USE OF "LOCATE"

As mentioned in section 4.4, program LOCATE can be run interactively. All manual input is via a subroutine called WATCH which is entered at the beginning of the program and then later in accordance with a parameter called IWATCH. If IWATCH = 0, WATCH is never entered. If $\geqslant 1$, it is entered on completion of the search. If $\geqslant 2$, when an improved solution is found. If $\geqslant 4$, when a node is about to be deleted. If $\geqslant 6$, when a newly active node is about to have its bound calculated. Higher values of IWATCH are only used for debugging.

Once it has been called, WATCH types a message indicating the reason for the call and then awaits further direction in the form of key-words typed by the user. The meaning of the 15 key-words is summarised below.

Flow	Print the flow matrix.
Distance	Print the distance matrix.
Cost	Print the linear cost matrix.
Parameters	Print the current value of IWATCH and two other parameters and accept new values for them.
Limit	Sets the number of nodes to be generated before the next call to WATCH.
Summary	Prints the number of nodes generated and also the number not yet discarded, the active node and the current minimum.
Best	Prints the current best assignment.
Time	Prints the computer time used in seconds.

Tree	Prints all the nodes in the tree below a specified node.
Single	Prints a specified node, giving its bound and the nodes immediately above and below itself.
Up	Prints all the nodes directly above a specified node.
Down	Prints one node from each level below a specified node.
Active	Declares a specified node to be active.
(blank)	Continues running program normally.
Stop	Terminates execution.

Most of these directives allow the user to watch the search process, but the "active" instruction gives the power to change the course of the search.

IWATCH is preset to 1 and so the program runs without interruption unless either "limit" or "parameters" is used at the beginning of execution. The output below shows NVR8 being solved without interruption. Lower case letters are typed by the user, upper case by the computer.

```
get,tape1=nvr8
/x,fortran,i=locate,e=4,k

START
?
**END**     TIME NOW IS   6.562     SINCE LAST CALL IS   1.555
         TOTAL NUMBER OF NODES IS      142


 SOLUTION IS       107.0000

     A  6      B  5      C  1      D  7      E  8      F  4      G  3
STOP
/
```

H2

The following output is for Gavett and Plyter's 4-machine problem and demonstrates a few of the key-words. The notes on the right-hand side were added later.

```
lgo,,,gp4

START
? flows                                              Print flow matrix.
N.B. ENTRIES ARE TOTAL PAIRWISE FLOWS

          A      B      C      D
 A        0     28.    25.    13.
 B       28.     0     15.     4.
 C       25.    15.     0     23.
 D       13.     4.    23.     0
?

distances                                            Print distance matrix.
          1      2      3      4

 1        0      6.     7.     2.
 2        6.     0      5.     6.
 3        7.     5.     0      1.
 4        2.     6.     1.     0
? parameters
IWATCH=   1    MAXEXCL=   4    DIMENS= 490
? 6 1 100                                            IWATCH = 6
?
START ANALYSIS OF ACTIVE NODE                        Node at level 0.
? time
            TIME NOW IS   7.101    SINCE LAST CALL IS    .034
?
START ANALYSIS OF ACTIVE NODE                        First inclusion node.
? time
            TIME NOW IS   7.124    SINCE LAST CALL IS    .023

                                                     Calculation of bound
                                                     for node at level 0
                                                     needed 0.023 seconds.
```

Continued on next page.

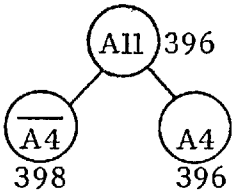
Total number of nodes generated.

Number of nodes not yet discarded.

```
? summary
NODES=     3 (  3)    ACTIVE=   2    BEST=999999.00
? tree
NODE   ASSIGNMENT    BOUND     PARENT   CHILDREN
? 1
   1       0   0       396.50       0        2     3
   2       A   4       396.00       1       -1     0
   3       A  -4       398.00       1       -1     0
? paremeters
IWATCH=   6    MAXEXCL=   1    DIMENS=  100
? 2 1 100
?
IMPROVED SOLUTION
? summary
NODES=     7 (  7)    ACTIVE=   6    BEST=    403.00
? best
    A  4    B  1    C  3    D  2
? tree
NODE   ASSIGNMENT    BOUND     PARENT   CHILDREN
? 2
   2       A   4       397.50       1        4     5
   3       A  -4       398.00       1       -1     0
   4       D   2       403.00       2        6     7
   5       D  -2       441.00       2       -1     0
   6       B   1       403.00       4       -1     0
   7       B  -1       419.00       4       -1     0
? time
            TIME NOW IS   7.198     SINCE LAST CALL IS    .074
?
**END**     TIME NOW IS   7.223     SINCE LAST CALL IS    .025
        TOTAL NUMBER OF NODES IS       8

 SOLUTION IS      403.0000

    A  4     B  1     C  3     D  2
STOP
/
```



The first branch.

The tree when the first assignment was found.

Only one more node was generated.

REFERENCES

1. Armour, G.C. and Buffa, E.S. (1963) "A heuristic algorithm and simulation approach to the relative location of facilities", Man. Sci. 9, 294-309.

2. Artle, R. and Varaiya, P.P. (1975) "Economic theories and empirical models of location choice and land use - survey", Proc IEEE 63, 421-427.

3. Bowman, V.J., Pierce, D.A. and Ramsey, F. (Nov. 1971) "A linear programming formulation of a special quadratic assignment problem", unpublished report, Carnegie-Mellon Univ.

4. Buffa, E.S. and Vollmann, T.E. (1966) "The facilities layout problem in perspective", Man. Sci. 12, 450-468.

5. Burkard, R.E. (Aug. 1973) "A perturbation method for solving quadratic assignment problems", presented at mathematical programming symposium, Stanford Univ.

6. ———— (1973) "Quadratische Bottleneckprobleme", O.R. Verfahren 18, 26-41.

7. ———— (Jan. 1975) "Numerische Erfahren mit summen und bottleneck-zuordnungsproblemen", report 75-1, Mathematishes Institut, Universitat zu Koln.

8. CERN Program Library, deck H300, Geneve, Switzerland.

9. Cabot, A.V., Francis, R.L. and Stary, M.A. (1970) "A network flow solution to a rectangular distance facility location problem", AIIE Trans. 11, 132-141.

10. Christofides, N. and Eilon, S. (1972) "Algorithms for large scale travelling salesman problems", O.R.Q. 23, 511-518.

11. Christofides, N. and Mitra, S.K. (1974) "A method for 0-1 programming", submitted for publication in O.R.Q.

12. Dearling, P.M. and Francis, R.L. (1974) "A network flow solution to a multifacility location problem involving Rectilinear distances", Transp. Sci. 8, 126-141.

13. Edwards, H., Gillett, E.E. and Hale, M.E. (1971) "Modular Allocation Technique (MAT)", Man. Sci. 17, 161-174.

14. Francis, R.L. (1967) "Some aspects of a minimax location problem", O.R. 15, 1163-1171.

15. ———— (1973) "A minimax facility configuration problem involving lattice points", O.R. 21, 101-118.

16. ———— and Goldstein, J.M. (1974) "Location theory: a selective bibliograpy" O.R. 22, 400-413.

17. ———— and White, J.A. (1974) "Facility layout and location", New Jersey, Prentice-Hall.

18. Garside, R.C. and Nichoson, T.A. (1968) "Permutation for the backboard wiring problem", Proc. IEEE 56, 27-30.

19. Gass, I.S. (1964) "Linear programming" New York, McGraw-Hill.

20. Gavett, J.W. and Plyter, N.V. (1966) "The optimal assignment of facilities to locations by branch-and-bound", O.R. 14, 210-232.

21. Gilmore, P.C. (1962) "Optimal and suboptimal algorithms for the quadratic assignment problem" J. SIAM 10, 305-313.

22. Graves, G.W. and Whinston, A.B. (1970) "An algorithm for the quadratic assignment problem", Man. Sci. 16, 453-471.

23. Glover, K.L. (1969) "An evaluation of plant layout algorithms", MBA research report, Seattle, Washington.

24. Hanan, M. and Kurtzberg, J. (1972) "A review of the placement and quadratic assignment problems", SIAM Rev. 14, 324-342.

25. Heider, C.H. (1973) "An N step, 2-variable search algorithm for the component placement problem", NRLQ 20, 699-724.

26. Held, M. and Karp, R. (1962) "A dynamic programming approach to sequencing problems", J. SIAM 10, 196-210.

27. ——— and ——— (1970) "The travelling salesman problem and minimum spanning trees", O.R. 18, 1138-1162.

28. ——— and ——— (1971) "The travelling salesman problem and minimum spanning trees: part II", O.R. 19, 1-23.

29. Hillier, F.S. (1963) "Quantitative tools for plant layout analysis", J. Ind. Eng. 14, 33-40.

30. ——— and Connors, M.M. (1966) "Quadratic assignment problem algorithms and the location of indivisible facilities", Man. Sci. 13, 42-57.

31. ——— and Lieberman, G.J. (1967) "Introduction to operations research", Houlden-Day.

32. Hitchings, G.G. (1973) "Analysis and development of techniques for improving the layout of plant and equipment", Ph.D. thesis, Univ. of Wales.

33. Khalil, T.M. (1973) "Facilities Relative Allocation Technique - (FRAT)", Int. J. Prod. Res. 11, 183-194.

34. Koopmans, T.C. and Beckmann, M.J. (1957) "Assignment problems and the location of economic activities", Econometrica 25, 53-76.

35. Kruskal, J.B. (1956) "On the shortest spanning subtree of a graph and the travelling salesman problem", Proc. Am. Math, Soc. 1, 48-50.

36. Kuhn, H.W. (1955) "Hungarian method for the assignment problem", NRLQ 2, 83-97.

37. Kurtzberg, J.M. (1964) "Algorithms for backplane formation", Proc. ONR symp. on microelectronics and large systems, Washington D.C.

38. Land, A.H. (1963) "A problem of assignment with inter-related costs", O.R.Q. 14, 185-199.

39. Lawler, E.L. (1963) "The quadratic assignment problem", Man. Sci. 9, 586-599.

40. ——— (1975) "The quadratic assignment problem: a brief review", unpublished report.

41. ——— and Wood, D.E. (1966) "Branch-and-bound methods: a survey", O.R. 14, 699-719.

42. Levin, P.H. (1964) "Use of graphs to decide the optimum layout of buildings", Architects J., 908-917.

43. Lin, S. (1965) "Computer solutions of the travelling salesman problem", Bell System Tech. J. 44, 2245-2269.

44. Loberman, H. and Weinberger, A. (1957) "Formal procedures for connecting terminals with minimum total wire length", J. ACM 4, 428-437.

45. Love, R.F. (1974) "The dual of a hyperbolic approximation to the generalised constrained multi-facility location problem with l_p distances", Man. Sci. 21, 22-33.

46. ——— and Morris, J.G. (1975) "Solving constrained multi-facility location problems involving l_p distances using convex programming", O.R. 23, 581-587.

47. Maxwell, W.L. (1964) "The scheduling of single machine systems : a review", Int. J. Prod. Res. 3, 177-199.

48. Mitten, L.G. (1970) "Branch-and-bound methods: general formulation and properties", O.R. 18, 24-34.

49. Moore, J.M. (1962) "Plant layout and design", New York, Macmillan.

50. ——— (1974) "Computer-aided facilities design: an international survey", Int. J. Prod. Res. 12, 21-44.

51. Morris, J.G. (1975) "A linear programming solution to the general rectangular distance Weber problem" NRLQ 22, 155-164.

52. Morse, P.M. (1972) "Optimal linear ordering of information items", O.R. 20, 741-751.

53. Munita, J. (1972) "Suboptimal solution procedures for the layout problem", M.Sc. thesis, Dept. of Man. Sci., Imperial College.

54. Munkres, J. (1957) "Algorithms for the assignment and transportation problems", J. SIAM. 5, 32-38.

55. Muther, R. (1955) "Practical plant layout", New York, McGraw-Hill.

56. Neghabat, F. (1974) "An efficient equipment-layout algorithm", O.R. 22, 622-628.

57. Nugent, C.E., Vollmann, T.E. and Ruml, J. (1968) "An experimental comparison of techniques for the assignment of facilities to locations", O.R. 16, 150-173.

58. Pierce, J.F. and Crowston, W.B. (1971) "Tree-search algorithms for the quadratic assignment problem", NRLQ 18, 1-36.

59. Pomentale, T. (1965) "An algorithm for minimising backboard wiring functions" CACM 8,699-703.

60. ———— (1967) "On minimisation of backboard wiring functions" SIAM Rev. 9, 564-568.

61. Raghavachari, M. (1969) "On the 0-1 integer programming problem", O.R. 17, 680-684.

62. Ritzman, L.P. (1972) "The efficiency of computer algorithms for plant layout", Man. Sci. 18, 240-248.

63. Scriabin, M. and Vergin, R.C. (1975) "~~Relative effectiveness of computer and manual~~ Comparison of computer algorithms and visual based methods for plant layout", ~~submitted for publication in~~ Man. Sci. 22, 172-181.

64. Simmons, D.M. (1969) "One-dimensional space allocation: an ordering algorithm", O.R. 17, 812-826.

65. Steinberg, L. (1961) "The backboard wiring problem: a placement algorithm", SIAM Rev. 3, 37-50.

66. Stoer, J. and Witzgall, C. (1970) "Convexity and optimisation in finite dimensions I", Berlin, Springer-Verlag.

67. Taha, H.A. (1973) "Concave minimisation over a convex polyhedron", NRLQ 20, 533-548.

68. Vollmann, T.E., Nugent, C.E. and Zartler, R.L. (1968) "A computerised model for office layout", J. Ind. Eng. 19, 321-336.

69. Wesolowsky, G.O. (1972) "Rectangular distance location under the minimax optimality criterion", Transp. Sci. 6, 142-155.

70. Yaspan, A. (1966) "On finding a maximal assignment", O.R. 14, 646-651.