

A LEXISEARCH APPROACH TO SOME COMBINATORIAL PROGRAMMING PROBLEMS

**A THESIS
SUBMITTED FOR THE DEGREE OF**

DOCTOR OF PHILOSOPHY

**By
M. RAMESH**



**Department of Mathematics & Statistics
SCHOOL OF MATHEMATICS & C.I.S.**

UNIVERSITY OF HYDERABAD

**P.O. Central University,
Hyderabad-500 046, India.**


January 1997




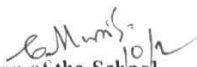
DEPARTMENT OF MATHEMATICS AND STATISTICS
SCHOOL OF MC1S
UNIVERSITY OF HYDERABAD
HYDERABAD

CERTIFICATE

This is to certify that the thesis entitled "*A LEXISEARCH APPROACH TO SOME COMBINATORIAL PROGRAMMING PROBLEMS*" submitted for the degree of doctor of philosophy in Statistics to the University of Hyderabad, Hyderabad, is a record of bonafide research carried out by Mr. M.Ramesh under the supervision of Professor S.N. Narahari Pandit and no part of the thesis has been submitted for any degree or diploma of any other University.


Supervisor
(Professor S.N.Narahari Pandit)


Head of the Department
(Professor K. Viswanath)


Dean of the School
(Professor C. Musili)

DECLARATION

The research work presented in this thesis has been carried out in the Department of Mathematics and Statistics, School of Mathematics and Computer/Information Sciences, University of Hyderabad, Hyderabad. The work is original and has not been submitted in part or full, for any other degree or diploma of any University.

Date : 10-2-1997

Place: Hyderabad



M.Ramesh

ACKNOWLEDGEMENTS

I express my deep sense of gratitude to Prof. S. N. Narahari Pandit, who has been my teacher and guide, and whose intellectual and scholarly guidance has enabled me to present this thesis.

I am thankful to Prf.C.Musili, Dean, School of MCIS, Prof K. Viswanath, Head, Department of Mathematics & Statistics, University of Hyderabad and Prof Sitaramaiah, Department of Mathematics & Statistics, University of Hyderabad, for the facility provided to me to complete my thesis work. I also thank full to other faculty members for their encouragement.

I am very grateful to Dr. C. Ragavendra Rao, University of Hyderabad, Dr. V.V. Haragopal, Osmania University, for their valuable suggestions and encouragement.

I offer my thanks to all the Non-Teaching staff of Department of Mathematics and Statistics for their help.

I offer my thanks, for the interest and encouragement accorded by my well-wishers and friends, Ms. Shanta Pandit, Dr. Ravi Kumar and Dr. Ashwini Kumar Datta.

I am grateful to my parents, brother and sister for their encouragement in prosecuting my studies. Their love and faith are my source of inspiration and strength.

Finally, I thank the CSIR for providing financial assistance for my research and the UGC-SAP programme for providing me the laser print outs.

M. Ramesh

CONTENTS

Page No

CHAPTER I

INTRODUCTION	001
1.1 The Shortest Route Problem	003
1.2 The Travelling Salesman Problem	003
1.3 The Job Scheduling Problem	003
1.4 Branch and Bound Algorithms	004
1.5 Lexi search	005
1.6 Chapter- wise details	005

CHAPTER II

THE LEXISEARCH APPROACH	010
2.1 Introduction	010
2.2 An Illustrative Example	013

CHAPTER III

THE TRAVELLING SALESMAN PROBLEM	019
3.1 Introduction	019
3.2 Vehicle Routing Problem	021
3.3 Computer Wiring Problem	022
3.4 Job Sequencing with switch over cost	023
3.5 Methods for solving the TSP	023
3.6 Heuristics	024
3.6.1 Nearest neighbour algorithm	024
3.6.2 Nearest Addition Algorithm	025
3.7 Exact Solutions	026

	Page No
3.7.1 Integer Programming Approach	026
3.7.2 Dynamic Programming approach	027
3.7.3 Branch and Bound algorithms	027
3.7.4 Eastman's algorithm	028
3.7.5 Lexisearch	028
3.8 Some variations of the classical TSP	029

CHAPTER IV

GENERALISED K- TRAVELLING SALESMAN PROBLEM	031
4.1 Introduction	031
4.2 k-Salesmen Problem with additional restrictions	032
4.3 Lexisearch Algorithm	032
4.3.1 Bound Calculations	036
4.3.2 Numerical Illustration	037
4.4 Computational Experience	044
4.5 Min- Max objective and lexisearch algorithm	049
4.6 Numerical Illustration	052
4.7 Computational Experience	054

CHAPTER V

MAXIMUM WEIGHTED CONNECTED SUB GRAPH	056
5.1 Introduction	056
5.2 Computation of Super Nodes	059
5.3 Reduction of the graph: Construction of super nodes	060
5.4 Reduction Algorithm: REDGRA	061

	Page No
5.5 Lexisearch algorithm: LEXIGRA	062
5.6 Numerical Illustration	065
5.7 Computational Experience	072
5.8 Comments	075

CHAPTER VI

QUADRATIC ASSIGNMENT PROBLEM	077
6.1 Introduction	077
6.2 A Survey of available algorithms	081
6.3 Sub- Optimal Procedures	082
6.3.1 Construction Methods	082
6.3.2 Improvement Methods	082
6.4 Optimal Solution Procedures	084
6.4.1 Single assignment algorithms	084
6.4.2 Pair assignment algorithms	085
6.4.3 Pair- Exclusion algorithms	086
6.4.4 Lexisearch	086
6.5 Lexisearch algorithm LEXIQAP	087
6.6 Numerical Illustration	090
6.7 Computational Results	111
6.8 Some Comments	113

APPENDICES

Appendix - 1	118
Appendix - 2	121
Appendix - 3	149
References	153

CHAPTER I

INTRODUCTION

Mathematical programming is concerned with finding optimal solutions rather than good solutions. Though the concept of optimisation is ancient, interest in this area has accelerated enormously with the development of computers and linear programming in the 1940s. Mathematical programming problems are broadly classified into two classes namely;

1. Continuous programming problems
2. Discrete programming problems

The problem of mathematical programming is to find a maximum or a minimum of an objective function whose variables are required to satisfy a set of well defined constraints.

If the objective function is continuous or partially continuous in the variable values that also lie in a feasible region which is compact, then it is a continuous programming problem. This type of problems are solved by some procedures which are based on concepts like continuity, convexity and neighbourhood. Some of them are the Simplex procedure, (Hadely,1994) Lagrangian multipliers method (Rao, 1984) and Steepest Ascent method (Rao, 1984).

If the decision variables take only discrete values, then it is a discrete programming problem.

If the set of solutions (solutions space) is a finite discrete set not necessarily of variables in the usual sense but may even be of other entities like permutations or combinations, then the problem is one of discrete programming.

If the solution space of a problem consists of combinatorial entities, then it is known as a combinatorial programming problem. Special techniques are available for solving different discrete programming problems. However, no general approach which is suitable for all discrete programming problems seems to be available and methods of finding optimum solutions are largely search methods (c.f. Pandit, 1963).

In the case of continuous programming problems, the existence of a solution may be doubtful, but in the case of combinatorial programming problems, the existence of an optimal solution is not in doubt. However, finding optimal solutions of these problems and recognising them as such is not trivial; one often obtains the optimal solution early in the process of solving the problem but can establish the optimality of the same only after considerable computational time and effort.

The general structure of combinatorial programming problems can be described as follows (Pandit, 1963):

“ There is a numerical function defined over the domain of arrangements (permutations or selections) of a set of elements. There are also feasibility criteria. The problem is to find the arrangements which are feasible and which optimise the numerical function.”

Following **are** some of the well-known combinatorial programming problems:

1.1 The Shortest Route Problem

There are n nodes connected with one another by 'links' (or arcs) of given lengths; the problem is to find the shortest route between node pairs in the above network.

1.2 The Travelling Salesman Problem

A set of n nodes ('cities'), with distance between every ordered node pair, is given. The problem is to find a least distance route for a salesman who must visit each of these cities starting and ending his journey in the same city.

1.3 The Job Scheduling Problem

There are n jobs to be completed and each job is processed through each of the ' m ' machines in the same order. A job cannot be processed on machine ' j ' until it is finished on machine ' $j-1$ '. Let the processing times of each job on each machine be given. The problem is to find a sequence of jobs (defined as a schedule) that minimises the total elapsed time (makespan) to complete all the jobs on all machines.

For many of the combinatorial programming problems the solution space is finite and hence it is theoretically possible to enumerate all the solutions. But often, even for problems of small size, **the number of** solutions can become very large and grows exponentially with the size of the problem making a complete enumerative solution not practicable.

Hence, there arises a need for implicit enumeration methods. In these methods, though theoretically **all** the solutions are examined, only a few are explicitly to be examined. The efficiency of the implicit enumeration algorithm basically depends **on how** quickly it eliminates subsets of solutions as not including any optimal solutions and ‘converge’ fast to an optimal feasible solution. There are two methods in this implicit Enumeration approach. They are 1) Lexicographic Search (Pandit, 1962) and 2) Branch and Bound (Little et al 1963).

1.4 Branch and Bound Algorithms

The Branch and Bound approach to combinatorial programming problems was first developed by Little et. al (1963) in the context of Travelling Salesman problem. Dakin (1965) gave a branch and bound algorithm to solve the general Integer Linear Programming problem. This technique has been successfully applied to solve both pure and mixed integer linear programming problems (Ravi Kumar, 1989).

The Branch and Bound method is characterised by two decision rules. One provides a procedure for the estimation of the upper bound for the value of the objective function at every stage, and the other specifies a ‘choice criterion’ for the selection of branching variable at the stage selected for further partitioning. In all the Branch and Bound algorithms the basic approach is the same, but the technique of selecting the nodes for further branching is different. A few such rules are LIFO (Last **in** first out), FIFO (first **in** first out) and LC (least cost) search.

A comprehensive surveys of Branch and Bound technique and approach, particularly in the context of travelling salesman problem and Quadratic Assignment Problem, can be found in Lawler and Wood (1966) and Agin (1966).

1.5 Lexisearch

The Lexisearch approach was first proposed by Pandit (1962) in the context of 'The Loading Problem'. From 1963 onwards this approach has been used to solve various combinatorial programming problems efficiently, e.g., The Assignment Problem (Jain, Mishra and Pandit, 1964), The Facility Location Problem (Das, 1976), The Travelling Salesman Problem (Pandit and Rajbongshi, 1976; Sundara Murthy, 1979; Subramanyam, 1980; Ramesh 1980; Chandrasekhar Reddy, 1987). The Job Scheduling Problem (Gupta , 1967; Rajabongshi, 1982; Bhanumurthy, 1986). In all these problems the lexicographic search was found to be more efficient than the branch and bound algorithms (c.f. Ravi Kumar, 1989).

It may be noted that in a general sense the Branch and Bound approach can be viewed as a particular case of lexicographic search.

1.6 Chapter-wise details

The thesis presents solution of some combinatorial programming problems using Lexisearch approach.

In Chapter II, some of the basic concepts of the Lexicographic Search approach are explained. Also this approach is explained with an illustrative example.

Chapter III discusses the Single Travelling Salesman Problem with the 'Standard Objective Function' of minimising the total tour cost. Some of the available algorithms are reported. Also the same problem, but with some structural constraints on the tour, is discussed. It is also shown that, contrary to expectation, change of scale of the costs c_{ij} can have an impact on the computational time of the lexiseach algorithm.

Chapter IV considers the k-Travelling salesmen Problem with constrained tours and with different types of objective functions. After reviewing the k salesman problem with structural constraints like the number of cities to be visited by each salesman, some non-standard objective functions are discussed.

In this same chapter is presented an algorithm for *single salesman* problem, where the objective is to minimise the maximum arc length in the tour.

Chapter V considers the problem of finding 'a maximum node weighted connected sub graph from a node weighted connected graph'. Situations which can be formulated as this problem can be exemplified by the following situation arising in the field of mining:

The 'area' to be mined out is divided into blocks, not only on the surface but underground as well. Each block has an identification number and is having a 'net value' (profit or loss as the case may be) associated

with it which is assumed to be known. However, to extract out a block it is necessary to ‘approach’ it through a neighbouring block which is already reached, except when the block of interest is a surface block. Cost of approaching a block is negligible, at least as compared to the ‘value’ of the block, and is ignored. The problem is to select the set of blocks which are ‘connected’ and have the total of their values a maximum.

The problem can be stated formally in Graph-theoretic terminology as follows:

Let $G = (A, E; V)$ be a symmetric connected graph with $A = \{1, 2, \dots, n\}$ as its node set, E the edge set while $V = \{v_1, v_2, \dots, v_n\}$ is the set of values associated with the corresponding nodes.

Let S be a subset of the node set A and let $G(S, E_S; V_S)$ be a connected sub graph with E_S and V_S as the corresponding edge set and value set respectively.

The value of this sub graph is the total of the values of the nodes in S and is denoted by $\text{Val}(G(S, E; V)) = \sum_{i \in S} v_i$

The problem is to select a connected sub set S of A such that it will have a value not less than the value of any other connected subset S of A .

It may be noted that unlike in the travelling salesman problem, here, the edges of the graph will not have any value attached to them but the nodes will have values v_i positive or negative as the case may be.

This problem was posed in 1988 by Caccetta and Giannini. They proposed a graph theoretical approach to solve the problem. They

have suggested 'spanning trees' as a tool for solving the problem, but have not developed a procedure to solve the problem completely. However, in Chapter V of the present thesis, *a procedure is developed to reduce where possible, the effective size of the problem by 'merging' appropriate blocks into connected superblocks.* Further, a lexisearch algorithm to solve this problem fully is presented.

A number of randomly generated problems, of varying sizes and varying levels of connectivity have been solved by this approach and the relevant computational experience is reported.

The Quadratic Assignment problem is studied in Chapter VI. The problem can be stated as follows:

The n facilities are to be assigned to n locations. Let f_{ik} be the flow between i^{th} facility to k^{th} facility and d_{jl} be the distance between j^{th} location and l^{th} . Here the objective is to assign the facilities to the locations such that the sum of pair wise interactions among the facilities weighted by the distance between their locations is minimised.

This problem can be stated formally as follows:

$$\begin{aligned} &\text{Minimise } \sum_{i,k} \sum_{j,l} f_{ik} d_{jl} x_{ij} x_{kl} \\ &\text{subject to } \sum_j x_{ij} = 1 \quad i=1,2,\dots,n \\ &\quad \sum_i x_{ij} = 1 \quad j=1,2,\dots,n \\ &\quad x_{ij} = 0 \text{ or } 1 \end{aligned}$$

In this chapter, some of the 'exact' and 'heuristic' algorithms available in the literature are reported. In particular, the Lexisearch

approach to this problem as developed by Das (1976) is also summarised. A new lexicographic search approach - with a different way of defining the alphabet table and the corresponding bounding algorithm is presented along with a report on relevant computational experience.

The thesis concludes with some general observations on the problems considered in the thesis. Some interesting open problems are also presented.

CHAPTER II

THE LEXISEARCH APPROACH

2.1 INTRODUCTION

The lexicographic search approach to combinatorial optimisation, as already pointed out, was developed by Pandit in the first instance for the knapsack problem (Pandit, 1962) and has since been applied to many other Combinatorial Programming Problems like Job Scheduling (c.f. Gupta, 1967) etc., in addition to the Assignment problem and Travelling Salesman Problem (c.f. Das, 1976) etc. Concepts involved in the lexicographic search are 'so natural' that the exact definitions connected with these concepts appear almost 'academic' if not pedantic. A detailed explanation of these concepts is given, for instance, in Ravi Kumar (1995).

The lexisearch derives its name from lexicography, the science of effective storage and retrieval of information. The basis for this approach is the possibility of defining a structure in the solution space on the following lines: Elements in the solution space of the problem are arranged in a hierarchical order as blocks and subblocks within blocks, etc., like the words in a dictionary. For instance, many combinatorial programming problems admit of representing the set of all possible 'solutions' as 'words', say $(\alpha_1, \alpha_2, \dots, \alpha_n)$ based on an alphabet, so that the first few 'letters' define a block of solutions, all having the specified 'leader/incomplete word' in common. Thus the entire set of 'words' in this 'dictionary' (*viz.*, the elements in the solution space) is segregated (partitioned/arranged) into blocks. The number of letters in a block (i.e. the number of letters of the

leader which defines the block) is called length of the block. The block 'B' with a leader $(\alpha_1, \alpha_2, \alpha_3)$ of length three consists of all the words beginning with $(\alpha_1, \alpha_2, \alpha_3)$ as the first three letters; The block A with leader (α_1, α_2) , of length 2, is the immediate 'Superblock' of B and includes B as one of its subblocks. The block C with leader $(\alpha_1, \alpha_2, \alpha_3, \gamma)$ is a subblock of 'B'. This block B' consists of many subblocks $(\alpha_1, \alpha_2, \alpha_3, \gamma)$, one for each different γ . The block B is the immediate Superblock of block C. Similarly, A is the immediate Superblock for the block B.

By the structure of the problem, it is often possible to get bounds to the values of all the words in a block (i.e. the 'bound for the block') by an examination of its leader. The possible cases are:

- Case 1. In case the block 'may contain a better word' (i.e. when the bound is less than the trial solution value) one has to go *into subblocks*.
- Case 2. In case the value of this bound is greater than or equal to the trial solution value, *jump over* to the *next block* within the present superblock. If the current block happens to be the last block in the superblock, of course, one goes to the first block of the next superblock.
- Case 3. When the value of the partial word is greater than or equal to the trial solution, *jump out* to the next 'Superblock'.

Further when the exact values of the leaders of the same size within the block are **non-decreasing** and in this case, the exact value of a 'current' leader is already greater than the trial value, one need not check the

subsequent blocks **within** this Superblock at all but can move over to the next Superblock . This property of monotonicity in the leaders of the same size is not always present, for instance the travelling salesman problem, with 'usual' alphabet table shows this structure. However the quadratic assignment problem does not allow this possibility.

The above concepts and the approach are illustrated below, in the case of the well known travelling salesman problem :

Let a,b,c,d be four cities to be covered by a salesman. Then the set of possible partial and complete words is listed lexicographically as follows:

a ab abc **abcd** abdc ac acb acbd acdb ad adb adbc adcb
 b ba bac bacd badc bc bcd bcad bcda bd bda bdac bdca
 c ca cab cabd cadb cb cba cbad cbda cd cda cdab cdba
 d da dab dabc dacb db dba dbac dbca dc dca dcab dcba

The words starting with 'a' can be grouped together; such a group is called a 'block' and city 'a' is called a leader of the block. In a block there can be subblocks; for instance, block of words with leader 'b' has subblocks ba, be and bd. While the words bacd and badce, are the two words which constitutes the subblock ba of the block b. There would be blocks having only one word, for example, the block with leader bac contains only the word bacd. All the incomplete words can be used as leaders to define blocks.

For the blocks of size 2, with leaders ab, ac, ad, the block with leader 'a' is the immediate Superblock.

The logical flow of the lexisearch procedure at each step is **mentioned** under the remarks column of the illustrative example (Table 2.1); the notations (or abbreviations) used are described as follows.

GS : Go to subblock i.e., add the first available 'free' letter of the remaining ones; GS for 'dc' is 'dca'.

JB : Jump block, to go to next block of the same length i.e., replace the last letter of the current block by the letter next to it in alphabet table; JB at 'abd' is 'abe'.

JO : Jump out to the next higher order block, i.e., remove the last letter of the current leader and then jump block; JO for 'cdbe' is 'cde'.

The search is started with the first letter 'a' as the leader. As the search is for an optimum solution, one can jump the current block if its bound is greater than or equal to the 'current trial solution'. No complete word in the block can be better than the current one.

2.2 An Illustrative Example

The Lexisearch procedure is explained through the solution of a Travelling Salesman Problem of size 5.

Let $[d_{ij}]$ be the given distance (cost) matrix and d_{ij} be the distance of city j from city i ; $i, j = 1, \dots, 5$. Arrange the cities according to their distances from city i ($i=1, 2, \dots, 5$) in increasing order; the resulting table is called the *Alphabet Table*. Due to this ordering of the cities, if the value of the

partial word is greater than the trial solution there will not exist any better solution in the subsequent blocks in the same superblock.

The following Travelling Salesman Problem of Size 5 is considered for illustrating the lexisearch procedure.

Distance matrix

$$\begin{bmatrix} 9000 & 5 & 13 & 8 & 53 \\ 7 & 9000 & 44 & 11 & 22 \\ 77 & 3 & 9000 & 88 & 56 \\ 84 & 34 & 63 & 9000 & 22 \\ 26 & 74 & 83 & 55 & 9000 \end{bmatrix}$$

The alphabet table for this problem is given below.

The alphabet Table

1	2	3	4	5
2-5	1--7	2-3	5-22	1-26
4-8	4--11	5-56	2-34	4-55
3-13	5--22	1-77	3--66	2--74
5--33	3--44	4-88	1-84	3-83
1-9000	2-9000	3-9000	4-9000	5--9000

Thus for instance, the search starts with $\alpha_1=2$ with cumulative value of 5, the next letter is from column 2 and is $\alpha_2 = 4$, since $\alpha_2 = 1$ would complete the cycle at this stage itself. Thus incomplete word is $1 \rightarrow 2 \rightarrow 4$ with a cumulative total value $5+11=16$. The next node added is $\alpha_3 = 5$ taken from column 4, node 5 being the first available node in this column, leading to cumulative total $16+22= 38$. This leads to the entries in the rows 1, 2 and 3 of the search table.

However, at any level of search i.e. corresponding to a specified incomplete word as the leader, one can define a bound to the values that are to be added to the value of the incomplete word in order to complete a tour. One obvious such bound can be computed as follows: The leader $(1,2) = (1 \rightarrow 2)$ has a cumulative value $d_{12} = 5$. The bound is $7+56+22+26=111$, since the four columns not yet included are columns 2, 3, 4 and 5 and the first available nodes are: 1 with value $d_{21} = 7$, 5 with value $d_{25} = 56$ and node 5 with value $d_{45} = 22$ and node 1 with value $d_{51}=26$. Thus the additive bound value is $7+111=118$ and hence the bound to the values of solution set with leader $(1,2)$ is 118. Similarly, for leader $(1,2,5) \triangleq (1 \rightarrow 2 \rightarrow 5)$, the cumulative value is $5+22=27$ and the cumulative value of the bound is sum of the terms $d_{31} = 77$, $d_{43}=63$ and $d_{51}=26$ i.e. $(77+63+26) = 166$ and the additive bound is $27+166=193$. Thus from the search table we find the value for the leader $1 \rightarrow \alpha_1 \rightarrow \alpha_2 = 1 \rightarrow 2 \rightarrow 4$, the leader value is 16 with the additive bound is 120 which is less than 185. Hence, this block may contain a solution better than the trial solution. Thus

the remark column GS to indicate that one has to go to the subblock at this stage, namely the block $1 \rightarrow 2 \rightarrow 4 \rightarrow 5$ with $\alpha_3=5$. To get any complete word with this leader one has to take an entry from column 3 as well as column 5. The first available entry in the column 3 is the third entry that is node (city)1. Hence from this column a contribution of at least 77 is to become necessary for a complete word in this block. Similarly from the column 5 minimum contribution that has necessarily to come is $d_{51}=26$, since in this column the first available node which happens to be the first in that column. Thus the residual bound for the leader is $77+26=103$ and hence the additive bound is $103 + 38 = 141$. Since all the cycles(words) could start from any of the nodes, since the starting node has been chosen as node 1 after the leader (1,5) there is no other leader of size 2 that is to be examined. Hence when the search of leader (1,5) is over the search is also complete and one of the best solutions available at one stage is $1 \rightarrow 3 \rightarrow 2 \rightarrow 4 \rightarrow 5 \rightarrow 1$ with value 75.

In the above search table the values within the bracket represent the cumulative total of the incomplete word.

CHAPTER III

THE TRAVELLING SALESMAN PROBLEM

3.1 INTRODUCTION

Travelling salesman problem (TSP, for short) is one of the oldest combinatorial programming problems (Flood, 1956, and Croes, 1958) and can be stated as follows(c.f. Sundara Murthy,1979):

There are n cities and the distance between any ordered pair of cities is known. Starting from one of the cities a salesman has to visit the other cities only once and return to the starting city. The objective is to find a minimal length tour i.e. a tour which is of minimal total length'.

The TSP can be defined equivalently as the problem of finding a Hamiltonian cycle of shortest length in a connected weighted graph.

Merill Flood who was associated with the RAND Corporation gave wide publicity to the TSP. The importance of the TSP comes not from the wealth of applications but from the fact that it is typical of other problems of its genre : Combinatorial Optimisation. Another reason for the popularity of the problem was its intimate connection with prominent topics in combinatorial problems arising in the then new subject of Linear Programming Problem (LPP, for short) the Assignment Problem and more generally, the Transportation Problem. TSP has some similarity with those other problems but is much harder to solve.

The combinatorial structure of this problem is very close to that of the Assignment Problem. The only difference is that the solution set of the

Travelling Salesman Problem is more restrictive, viz., the permutation matrix, say X , which is a feasible solution of the Assignment Problem will also be a feasible solution of the TSP only if it is non decomposable. This additional restriction brings some complexity into solving this problem and for the same reason, many of the methods which solve the assignment problem fail to do so in this case.

Dantzig, Fulkerson and Johnson (1954) solved a sizeable TSP, by formulating it as an LPP. The requirement of indecomposability of the permutation matrix X can be checked through a set of linear constraints, though the number of constraints grows very fast as the size of the problem increases. Thus theoretically it is possible to treat the TSP as an LPP. However, in practice, due to the very large number of these special constraints which are to be checked for indecomposability, it is not practicable to solve the TSP by this approach, except when the size of the problem is relatively very small. The cutting plane method was used to solve the above LPP with integer constraints and they also seem to have used the concept of branch and bound in some sense to solve the same (c.f.Lawler et al.,1985).

It may be noted that Das.S (1976) applied the lexisearch for TSP and further modifications were considered by Sundara Murthy (1979) and Srinivas (1990). Before this approach to TSP and its various versions are developed, a brief discussion of the TSP and various other approaches of the same is in order.

The Branch and Bound method was developed by Little et al., in 1963 for solving the TSP. From then onwards it has been widely used to solve different problems in Operations Research.

The solution space of TSP is a sub set of the solution space of the assignment problem (AP for short) in that every permutation of order n is a solution for the AP while only indecomposable permutations are acceptable as solutions of TSP. Hence, any procedure to solve the AP can be modified to solve the TSP, if one can incorporate a procedure to check for cycles in the permutation proposed as a solution. This fact is exploited in converting the lexsearch algorithm for AP into one for TSP (c.f. Pandit(1964,1965)).

Despite the fact that the travelling salesman model applies directly to a very useful sounding solution, namely that of salesman wishing to minimise his travel distance, most of the reported applications are quite different; seemingly unrelated problems are solved by formulating them as instances of the TSP. Some of the applications are listed below.

3.2 VEHICLE ROUTING PROBLEM (c.f. Clerker and Wright 1964)

The n nodes are indexed $i=1,2,\dots,n$ and $i=1$ refers to the origin. There are m vehicles, indexed $k=1,2,\dots,m$. Node i has a demand q_i , the distance between the node i and node j is d_{ij} . The capacity of vehicle k is Q_k . Partition the nodes into sets such that each set contains the origin. An assignment is an one to many mapping of the vehicles to the nodes in

such a way that the sum of the requirements of the nodes in the set associated with a vehicle should not exceed the capacity of that vehicle. The optimal assignment is that assignment whose total tour length of all the vehicles is a minimum. The basic Vehicle Routing Problem is that of identifying the partition of nodes which has the shortest distance covered by the vehicles.

3.3 COMPUTER WIRING PROBLEM

The following problem occurs repeatedly in the design of computers and other digital systems.

A system consists of a number of modules and several pins are located on each module. The physical position of each module has been determined. A given subset of pins has to be inter-connected by wires. In view of possible future changes or connections and of the small size of a pin, at most two wires are to be attached to any pin. In order to avoid signal cross-talk and to improve ease and neatness of wiring, the total wire length should be minimised.

Let c_{ij} denote the actual distance between pins i and j . If any number of wires could be attached to a pin, then an optimal wiring would correspond to a minimum spanning tree which can be found efficiently by various algorithms (c.f. Lawler et al 1985). However, the condition on number of wires connected to a pin implies that we must find a minimum length Hamiltonian path. If we create a dummy pin O with $c_{oj} = c_{jo} = 0$ for all j , then the wiring problem becomes an $(n+1)$ city symmetric TSP(c.f Lenstra and Rinnooy Kan , 1975).

3.4 JOB SEQUENCING WITH SWITCH OVER COST

Consider the problem of sequencing n jobs on a single machine. The jobs can be done in any order and the objective is to complete **all** of them in the shortest possible time. After a particular job i is completed, it requires a certain amount of time or cost to prepare the machine to work on job j , the cost c_{ij} depending on i as well as on j . Let the time required to complete the job when the machine is ready to work on j be p_j . The total time required to complete the job j directly after job i is

$$t_{ij} = c_{ij} + p_j$$

Any job sequence for processing is obviously a cyclic permutation with symbols $\{1, 2, \dots, n\}$. The time required to complete all the jobs is

$$\sum_{i=0}^n (c_{i\pi(i)} + p_{\pi(i)}) = \sum c_{i\pi(i)} + \sum p_j$$

Here the sum of all p_j 's is a constant and is invariant over all permutations. Hence by considering c_{ij} as distance between city i and city j , it can be seen that the problem on hand is essentially a TSP (c.f. Lawler et al 1985).

3.5 METHODS FOR SOLVING THE TSP

Based on the approaches adopted, methods for solving the TSP can be classified into different categories as follows.

1. Exact Methods
2. Heuristic Methods

As already noted, one of the earliest attempts is perhaps of Dantzig et al (1954) who introduced the cutting plane approach for general integer programming problem and in particular, to the TSP. Another notable early

attempt to solve the **TSP** relying on techniques of integer programming is that of Miller et al(1960). **Bellman(1962)** proposed an algorithm based on dynamic programming approach. Lin and **Kernigham (1973)** proposed an algorithm based on the principles of heuristic search. Karp (1979) has proposed a patching algorithm, a polynomial time approximation algorithm which tends to give solutions with small relative error, and works on the principles of the solution of the Assignment problem (Srinivas, 1990).

The most popular computational methods for exact solutions were based on the branch and bound approach as developed by Little et al (1963) and subsequently modified by Shapiro (1966) basing on Eastman's algorithm (1958).

3.6 HEURISTICS

Algorithms that do not guarantee the optimum solutions to the given problem, but do find what one hopes and expects are 'near optimum solutions' are known as Heuristic algorithms.

Some of the Heuristic algorithms of TSP are given below:

3.6.1. Nearest neighbour algorithm: It is an appealing method for constructing a tour. It works by selecting the locally superior alternative at each step. This approach for the TSP leads to the following procedure:

Step 1: Start with a partial tour consisting of a single, arbitrarily chosen city a_1 .

Step 2: If the current partial tour is a_1, a_2, \dots, a_k , $k < n$, and let a_{k+1} be the city, not currently on the tour, which is closest to a_k and add a_{k+1} at the end of the partial tour.

Step 3: Repeat step 2 until all the cities have been visited.

Step 4: When the current tour contains all the cities, then include the city a_1 to the solution to make the tour(solution) complete and get the value of the tour and stop the procedure.

This is obviously a greedy algorithm and hence suffers from the obvious drawback of many greedy algorithms, namely, a choice of the cheapest steps in the beginning can force one to choose very costly items at the end and make the solution value far from optimal. However from the empirical evidence, it is claimed that the values of solutions obtained by this algorithm are generally within 20% nearness of the optimum value (c.f Lawler, et al., 1985)

3.6.2 Nearest Addition Algorithm (NAA. for short):

Step 1: Start with a partial tour consisting of a single, arbitrarily chosen city i .

Step 2: If the current partial tour T does not include all the cities, find those cities k and j , j on the tour and k not on the tour for which c_{kj} is minimal.

Step 3: Let (i, j) be either one of the two edges involving j in T and replace it by (i, k) and (k, j) to obtain a partial tour including k and go to step 2 if the tour is not completed.

The Nearest insertion Algorithm that follows is slightly better than NAA. It chooses k in step 2 above but instead of pulling k next to j , it finds the best place to insert it. (c.f Lawler et al, 1985).

The cheapest insertion algorithm chooses the city k which is not in the partial tour T in the step 2 of NAA in such way that the resultant partial tour will have the cheapest cost (length).

3.7 EXACT SOLUTIONS

The following approaches lead to exact solution of the TSP

1. Integer programming approach
2. Dynamic programming approach.
3. Branch and Bound algorithms:
 - a) Eastman's algorithm
 - b) Low cost branch and bound algorithm.
4. Lexisearch Method.

3.7.1 INTEGER PROGRAMMING APPROACH

The TSP can be formulated as a integer programming problem as follows.

In addition to the linear programming formulation of AP as

$$\text{Minimise } \sum_{i=1}^n \sum_{j=1}^n d_{ij} x_{ij}$$

subject to the constraints

$$\sum_{j=1}^n x_{ij} = 1 \quad i=1,2,\dots,n \quad \sum_{i=1}^n x_{ij} = 1 \quad j=1,2,\dots,n$$

cycle formation is excluded by introducing the constraints

$$\sum_{i \in S} \sum_{j \in S} x_{ij} \leq |S| - 1$$

where S ranges over all possible subsets of size $|S|$, $S=2,3,..n$.

Thus, for a 10×10 problem, these additional restrictions will be $2^{10-1} - 1 = 2^9 - 1 = 511$ in number.

3.7.2 DYNAMIC PROGRAMMING APPROACH

The recursive technique of dynamic programming has also been applied to the TSP (Bellman 1962, Held & Karp 1962). This can be used by viewing the solution process for the problem as the result of a sequence of decisions. In this method an optimal sequence of decisions is arrived at, making explicit appeal to the principle of optimality. This principle states that whatever the initial state and decisions are, the remaining decisions must constitute an optimal sequence with regard to the state resulting from the first decision. Owing to its enormous storage requirements dynamic programming can only solve relatively small size problems. However, the method is perhaps useful in obtaining some theoretical running time characteristics and is the basis for an effective bounding procedure in practical routing algorithms (c.f Srinivas, 1990).

3.7.3 BRANCH AND BOUND (B B for short)ALGORITHMS

Little et al developed an algorithm to solve the TSP using Branch and Bound approach. The main idea of this approach is to break up the set of all tours into smaller and smaller subsets and to calculate for each of them a lower bound on the cost of the best tour therein. The bound guides

the algorithm in deciding whether a given **subset** needs to be further scrutinised for optimality or whether it can be rejected as containing no solution better than the one at hand.

The subsets of tours are represented as the nodes of a tree and the process of partition as a branching of the tree. Hence, this method is known as Branch and bound.

3.7.4 EASTMAN'S (1958) ALGORITHM

Depending on the procedure used for getting the bounds at each branch, many BB algorithms were developed, for instance by Little, Murthy, Sweeney & Karel(1963), Shapiro(1966), Christofides(1970), Bellmore & Malone(1971), Smith, Srinivasn & Thompson(1977). One such algorithm is due to Eastman. In this algorithm the bounding is based on solving an AP. In this approach the bound is calculated based on the optimal solution of the Assignment Problem(AP). It solves the easier AP that allows sub tours and then systematically forbids sub tours until finally a tour is obtained that is optimal.

3.7.5 LEXISEARCH

As already noted this approach was developed and applied for solving discrete optimisation problems (Pandit, 1962) before the BB technique was developed first, in the context of TSP. This approach has already been presented in Chapter II and this (i.e. lexisearch) approach to different ‘versions’ of the TSP is presented in the next section, as well as in the next chapter.

At this point, one more observation about the travelling salesman problem is perhaps in order. It has been suggested that the TSP can be formulated as an equivalent longest path problem (c.f. Hardgrave and Nemhauser 1962) and that if the latter can be solved, the TSP is also obviously solved. However, as pointed out in Pandit (1965), the situation is not that simple; longest route problems with specified steplength are 'equivalent' to the shortest route problems with the same specified steplength, but longest or shortest paths (not allowing for routes) are in general non-markovian and hence their solution is not easy.

3.8 SOME VARIATIONS OF THE CLASSICAL TSP

Some of the many interesting generalisations of the travelling salesman problem are given below:

'The Truncated Travelling Salesman problem' can be stated as follows: There are n cities $i=1,2,\dots,n$ and $N = \{1,2,3,\dots,n\}$. The distance $d(i,j)$ between any ordered pair of cities (ij) is known. A subset HQ of size $< n$ of the n cities constitutes the potential places for setting up a Headquarters. A Salesman has to visit only m out of the n cities with the restriction that his tour should include at least one city from the subset HQ . The problem is to find a feasible tour of m cities with a minimum length'.

A lexicographic search Algorithm was developed to solve the above problem (c.f. Sundara Murthy, 1979).

The travelling salesmen problem with the following restrictions was solved by the Scroggs and Therp(1972) and Das (1976):

(1). Some ordered pairs of cities are given. The salesman should visit the first city in each pair before the second, not necessarily immediately ; this is called the '***Precedence***' constraint .

(2). Some cities are specified to be visited at specific steps from Headquarters; this is called '***fixed position***' constraint.

These restrictions occur frequently in practical travelling salesman situations. For example, an executive may have to visit plant X before plant Y, since information obtained at plant X may influence his discussions at plant Y.

CHAPTER IV

GENERALISED K-TRAVELLING SALESMEN PROBLEM

4.1 INTRODUCTION

The present chapter deals with a specific generalisation of the standard TSP where the number of salesmen need not be one only.

An obvious form of the objective function requirement is to minimise the total length of all the sub tours. This can be solved by using any existing algorithms by introducing dummy head quarters.

Frederickson, Hecht & Kim (1978) have considered the following k- Salesmen problem:

'Let a network consist of n cities, of which city '1' is the Head Office. k (>1) salesmen are stationed at the Head office. Each salesman must follow a route that takes him from head office through some subset of the cities and then back to the head office. Each city is visited by at least one salesman. The problem is to minimise the length of the sub tour containing more number of cities'.

Frederickson et al. suggested a heuristic approach which is claimed to give the 'near optimum solution'.

The present chapter considers the k-salesmen problem with additional restriction on number of cities visited by each salesman and also that each city is visited by only one salesman. The problem is to determine an optimal tour which consists of k sub tours with specified sub tour length (i.e. number of nodes in the subtour).

Another objective function considered in this chapter is 'to minimise the maximum step value in a tour', in the case of the *single salesman*.

4.2 k-Salesmen Problem with additional restrictions

In the usual k-salesmen problem, the problem is to determine the optimal tour which has the minimum length, but it may consist of only a tour with the minimal total length, where a tour is defined as the union of all the subtours, one for each salesman; However, the step lengths of subtours are not specified; some of them may even be zero, the case where some salesman may 'simply' sit at the headquarters. But allowing 'idle salesman' could be very expensive. Thus if the subtour of salesman a has n_a nodes in it other than the headquarters, one has the conditions on n_i as $\sum n_i = N-1$, n_i 's being non negatives (i.e. some of them may be zero also) but unspecified.

The modified problem has the following additional restriction that the n_i 's are part of the problem specification and have prescribed values only.

4.3 LEXISEARCH ALGORITHM

Let $D = [d_{ij}]$ $i, j = 1, 2, \dots, N+k-1$, where d_{ij} be the distance from i to j . The problem is reduced to single salesman problem by introducing $K-1$ dummy origins. For convenience of computation the $k-1$ images' of the headquarters (node 1) are given the names 2, 3, 4, ..., k , where the headquarter node is given the name 1, the remaining, non headquarters nodes are

labelled $k+1, k+2, \dots, k+N-1$. Thus, in this extended distance matrix the first k columns are mutually identical. The same is true with the first k rows. Let the trial solution be

$1 \rightarrow \alpha_{1,1} \rightarrow \alpha_{1,2} \rightarrow \dots \rightarrow \alpha_{1,n1} \rightarrow 1 \rightarrow \alpha_{2,1} \rightarrow \alpha_{2,2} \rightarrow \dots \rightarrow \alpha_{2,n2} \rightarrow 1 \dots 1 \rightarrow \alpha_{k,1} \rightarrow \alpha_{k,2} \rightarrow \dots \rightarrow \alpha_{k,nk} \rightarrow 1$
with value TR.

The lexiseach algorithm to solve this k-TSP is given briefly below.

Algorithm LEXIGTSP

The following notations are used in this algorithm.

W : Current / partial solution

TR : Trial value associated with a trial solution

$1 \rightarrow \alpha_{1,1} \rightarrow \alpha_{1,2} \rightarrow \dots \rightarrow \alpha_{1,n1} \rightarrow 1 \rightarrow \alpha_{2,1} \rightarrow \alpha_{2,2} \rightarrow \dots \rightarrow \alpha_{2,n2} \rightarrow 1 \dots 1 \rightarrow \alpha_{k,1} \rightarrow \alpha_{k,2} \rightarrow \dots \rightarrow \alpha_{k,nk} \rightarrow 1$

α_{ij} : The city visited by the i^{th} salesman as j^{th} city in his tour.

n_i : The number of cities visited by i salesman $i=1, 2, \dots, k$ and

$$\sum n_i = N-1$$

Step 1. Arrange all the cities according to their distances from the city $i=1, 2, \dots, N+(k-1)$. This arrangement consists of $N+(k-1)$ columns and $N-1$ rows since the origins are removed. Each column represents a city, say, 'A' and the elements in that column are the cities arranged in increasing order according to their distance from the city 'A'.

Step 2. Include the first reachable city from the origin in the partial solution W. If the distance itself is greater than or equal to TR then stop. Otherwise go to next step.

Step 3. Calculate the Bound.

Step 4. If the (Bound + Partial Solution Value) is greater than or equal to trial solution then drop the city added in step 2. Now go to step 2 i.e. JB. Otherwise go to next step i.e., GS.

Step 5: Include the next reachable city (from the last city included in the partial solution W) into the partial solution.

Step 6: If Partial Solution Value is greater than or equal to the TR, drop the city added in step 5. Also drop the last city in W. Go to step 5 (i.e. JO).

Otherwise go to step 7.

Step 7: If the (Partial Solution Value + Bound) is greater than or equal to TR, then drop the newly added city in step 5 (i.e. JB). Go to step 5. Otherwise go to step 8.

Step 8: If the Partial Solution contains $\sum_{i=1} n_i + (i-1)$ cities, add the dummy origin to the partial solution . Calculate the value of this Partial Solution.

If it is greater than or equal to the TR, drop the dummy origin and last two cities from the Partial Solution (i.e. JO).

If W contains only one city, go to step 2;

Otherwise go to step 5

Otherwise go to the next step.

Step 9: Calculate the Bound.

Step 10: If the (Partial Solution Value + Bound) is greater than or equal to TR then drop the dummy origin and also last city from W. Go to step 5 (i.e. JB).

Otherwise go to step 11.

Step 11: Include the latest possible city from the dummy origin i in W.

Step 12: If (Partial Solution Value) is greater than or equal to TR, drop the last dummy origin and also the city from which the i^{th} dummy origin was reached. Go to step 5.

Otherwise goto next step.

Step 13: Calculate the Bound.

Step 14: If (Partial Solution Value + Bound) is greater than or equal TR, drop the recently added city in W and go to step 11.

Otherwise go to next step.

Step 15: Include the latest reachable city from the last city in W.

Step 16: If (Partial Solution Value) is greater than or equal to TR, drop the last two cities; if dummy origin happens to be one of these two cities, also drop the city from which the dummy origin was reached, reduce the value of i by 1 i.e., i becomes $i-1$.

Otherwise go to next step.

Step 17: Calculate the Bound.

Step 18: If (Partial Solution Value + Bound) is greater than or equal to TR, drop the latest city i.e., JB. Go to step 15.

Otherwise go to next step.

Step 19: If the number of elements in W is less than $(\sum n_i + k)$ go to step 15.

Otherwise go to step 20.

Step 20: Add the dummy origin to W and calculate the value of W. If it is greater than or equal to the TR drop the dummy origin and also the last two cities in W. Go to step 15.

Otherwise go to step 21.

Step 21: Replace TR by Partial Solution Value and trial solution by W.

Drop the dummy origin and the last two cities in W. Go to step 15.

4.3.1 BOUND CALCULATIONS

Bound is the sum of distances of the columns (cities) (which is not in the word, excluding latest city) to the first reachable city (excluding latest city) within the first $(M-1)$ cities if any; otherwise, take the distance to M^{th} city in that column. The value of M may be $(1/4)(N+k-1)$, $(1/3)(N+k-1)$ etc. The value of M is arbitrary; it may be chosen as 1/3, 1/4, or any other

convenient fraction of the number $N+k-1$. The larger the value of M , the better (more 'efficient') is the bound obtained but the computations also will be higher, while a smaller M requires less computations for bound setting but requires a more intensive search. In other words, this way of computing the bound may result in increasing the number of solutions to be searched, but reduces the computational time on computing the bound as it is calculated for every partial solution.

4.3.2 Numerical Illustration

As an illustration, consider a network of 6 cities including the head office with $k=2$ salesmen. One salesman is to visit $n_1=2$ cities and the other salesman is to visit the remaining $n_2=3$. Since $k=2$, one dummy head office is to be introduced. Thus, one has nodes 1,2 for head office and nodes 3,4,...,7 for the other cities. The corresponding distance matrix is

$$\begin{bmatrix} 999 & 999 & 22 & 8 & 9 & 15 & 7 \\ 999 & 999 & 22 & 8 & 9 & 15 & 7 \\ 22 & 22 & 999 & 23 & 15 & 19 & 1 \\ 8 & 8 & 23 & 999 & 5 & 25 & 22 \\ 9 & 9 & 15 & 5 & 999 & 10 & 19 \\ 15 & 15 & 19 & 25 & 10 & 999 & 15 \\ 7 & 7 & 1 & 22 & 19 & 15 & 999 \end{bmatrix}$$

The alphabet table is the arrangement of the cities (excluding the origin '1' and dummy origin '2') from city i ($i=1,2,\dots,7$), in increasing order of distances.

The alphabet table

1	2	3	4	5	6	7
7-07	7-07	7-01	5-05	4-05	5-10	3-01
4-08	4-08	5-14	7-22	6-10	7-15	6-15
5-09	5-09	6-19	3-23	3-15	3-19	5-19
6-15	6-15	4-23	6-25	7-19	4-25	4-22
3-22	3-22	3-999	6-999	5-999	6-999	7-999

The value of M is taken as 3 and the first 3 cities (i.e. $M=3$) which have the least distances from the city $i=1,2,\dots,N+k-1$ are listed below along with their distances and are considered for bound calculations.

1	2	3	4	5	6	7
7-07	7-07	7-01	5-05	4-05	5-10	3-01
4-08	4-08	5-15	2-08	2-09	2-15	2-07
5-09	5-09	6-19	1-09	1-09	1-15	1-07

The search table

$1 \rightarrow \alpha_{11}$	$\alpha_{11} \rightarrow \alpha_{12}$	$\alpha_{12} \rightarrow 2$	$2 \rightarrow \alpha_{21}$	$\alpha_{21} \rightarrow \alpha_{22}$	$\alpha_{22} \rightarrow \alpha_{23}$	$\alpha_{23} \rightarrow 1$	Remarks
$1 \rightarrow 7,7$	$7 \rightarrow 3,1$ 8,43	$3 \rightarrow 2,22$ 30,28	$2 \rightarrow 4,8$ 38,24	$4 \rightarrow 5,5$ 43,24	$5 \rightarrow 6,10$ 53,15	$6 \rightarrow 1,15$ TR=68 X	GS
							GS
							GS
							GS
							GS
							GS
							JO
					X		
				$4 \rightarrow 6,25$ 63,19 X			JB
			$2 \rightarrow 5,9$ 39,28				GS
				$5 \rightarrow 4,5$ 44,23			JB
				$5 \rightarrow 6,10$ 49,23 X			JB
			$2 \rightarrow 6,15$ 45,20				GS
				$6 \rightarrow 5,10$ 55,13			JB
				$6 \rightarrow 4,25$ 70			JO
		X	X				
	$7 \rightarrow 6,15$ 22,43						GS
		$6 \rightarrow 2,15$ 37,33 X					JB

$1 \rightarrow \alpha_{11}$	$\alpha_{11} \rightarrow \alpha_{12}$	$\alpha_{12} \rightarrow 2$	$2 \rightarrow \alpha_{21}$	$\alpha_{21} \rightarrow \alpha_{22}$	$\alpha_{22} \rightarrow \alpha_{23}$	$\alpha_{23} \rightarrow 1$	Remarks
1→4,8	7→5,19 26,55 7→4,22 29,48 X						JB
							JB
	4→5,5 13.33						GS
		5→2,9 22,24					GS
			2→7.7 29,35				GS
				7→3,1 30,34			GS
					3→6,19 49,15		GS
						6→1,15 TR=64 X	JO
					X		
				7→6,15 44,34 X			JB
			2→6,15 37,17				GS
				6→7,15 52,20 6→3,19 56,8 X			JB
			2→3.22 44,23 X				JB
	4→7,22 30,44 4→3,23 31,34 4→6,25 33,28	X					JB
							JB
							GS

$1 \rightarrow \alpha_{11}$	$\alpha_{11} \rightarrow \alpha_{12}$	$\alpha_{12} \rightarrow 2$	$2 \rightarrow \alpha_{21}$	$\alpha_{21} \rightarrow \alpha_{22}$	$\alpha_{22} \rightarrow \alpha_{23}$	$\alpha_{23} \rightarrow 1$	Remarks
$1 \rightarrow 6, 15$	$5 \rightarrow 3, 15$ 24, 38	$3 \rightarrow 2, 22$ 46, 37					GS
							JB
	$5 \rightarrow 7, 19$ 28, 51 X						JB
							GS
	$6 \rightarrow 5, 10$ 25, 22						GS
		$5 \rightarrow 2, 9$ 34, 17					GS
		$2 \rightarrow 7, 7$ 41, 28 $2 \rightarrow 4, 8$ 42, 10					GS
		$4 \rightarrow 7, 22$ 64					JO
		$2 \rightarrow 3, 22$ 56, 16 X					JB
		X					
$1 \rightarrow 3, 22$	$6 \rightarrow 7, 15$ 30 34	$3 \rightarrow 2, 22$ 56, 24 X					JB
	$6 \rightarrow 3, 19$ 34, 28						GS
							JB
	$6 \rightarrow 4, 25$ 40, 18						GS
			$4 \rightarrow 2, 8$ 48, 18 X				JB
	X						GS
	$3 \rightarrow 7, 1$ 23 35						GS

$1 \rightarrow \alpha_{11}$	$\alpha_{11} \rightarrow \alpha_{12}$	$\alpha_{12} \rightarrow 2$	$2 \rightarrow \alpha_{21}$	$\alpha_{21} \rightarrow \alpha_{22}$	$\alpha_{22} \rightarrow \alpha_{23}$	$\alpha_{23} \rightarrow 1$	Remarks
		7→2,7 30,28	2→4,8 38,24	4→5,5 43,24 4→6,25 63,19 X			GS
							GS
							JB
							JB
			2→5,9 39,28 2→6,15 45,20 X				GS
							JB
	3→5,15 37,42 3→6,19 41,34 3→4,25 45,38	X					JB
							JB
							STOP

In the above search table, search starts with $\alpha_{11}=7$ with the cumulative value $d_{17}=7$. The next letter (or city) in the word is the first reachable city from 7, i.e. city 3 which is the first element in the alphabet table under column 7. Value of this partial word $1 \rightarrow 7 \rightarrow 3$ is $7+1=8$. Since the first salesman is restricted to visit only 2 cities, he has to return to the head office, the value of the partial word $1 \rightarrow 7 \rightarrow 3 \rightarrow 2$ is $8+22=30$. From the dummy origin 2, the first reachable city(i.e. the city which is not visited by the first salesman) is 4, i.e. $\alpha_{21}=4$, the value of $1 \rightarrow 7 \rightarrow 3 \rightarrow 2 \rightarrow 4$ is 38.

In the above table the bold letters represent the bound values of the partial word. For instance the bound value of the partial solution $1 \rightarrow 7 \rightarrow 3$ is the sum of the distance of the first reachable city from 3 and the distance of first reachable city (among the first 3 cities) from the cities which are not in the word i.e. $15+8+5+5+10=43$. Similarly for the partial word $1 \rightarrow 4 \rightarrow 5 \rightarrow 2 \rightarrow 7 \rightarrow 6$, the bound is the sum of the distance of the first reachable city from 6 (i.e. the city 3) and the first reachable city from the city 3. Since $M=3$ and no other city is reachable within the first 3 cities from the city 3, for bound calculations the distance between the city 3 and city 6 is considered. Hence the bound for this partial word is $15+19=34$. Similarly the bound for $1 \rightarrow 4 \rightarrow 5 \rightarrow 2 \rightarrow 6 \rightarrow 7$ is $1+19=20$.

4.4 COMPUTATIONAL EXPERIENCE

The procedure developed and illustrated above is essentially the same as the lexsearch algorithm for the usual TSP, with appropriate modifications. A FORTRAN programme to implement this procedure is listed in appendix 2.

However, the question as to whether the choice of the ordering of the subtours either of same step lengths or of different step lengths n_i , as the case may be, remains not yet answered. A computational experiment, by solving a same set of problems by the same algorithm but changing the indexing of the n_i 's and comparing the solution times was carried out for this purpose.

A paired t statistic was calculated for each of the pair sequences {2&11,11&2},{3&10,10&3}, {4&9,9&4}, {5&8,8&5}, {6&7,7&6}, and {2&13,13&2},{3&12,12&3},{4&11,11&4},{5&10,10&5},{6&9,9&6}, and {7&8, 8&7} for problems of sizes 14+1 and 16+1 respectively. Details of the results namely timings(in seconds), the optimum solution value obtained and relevant statistics reported in the following tables for 10 problem data sets are generated for each of sizes 14+1 and 16+1 ,

Problem size : 14 +1
TABLE 4.1.1
OPTIMUM SOLUTION TIMINGS

Ordering of subtours

P.No	2& 11	3&10	4 & 9	5 & 8	6 & 7	7 & 6	8 & 5	9 & 4	10& 3	11& 2
1	0.49	0.49	0.38	0.43	0.49	0.52	0.54	0.49	0.60	0.71
2	0.49	0.49	0.54	0.60	0.65	0.71	0.76	0.76	0.65	0.54
3	0.49	0.54	0.60	0.60	0.60	0.65	0.76	0.87	0.87	0.82
4	0.65	0.60	0.65	0.71	0.82	0.93	0.98	0.98	0.87	0.98
5	0.54	0.49	0.49	0.60	0.60	0.71	0.71	0.65	0.60	0.71
6	0.54	0.49	0.49	0.49	0.49	0.60	0.71	0.82	0.87	0.87
7	0.43	0.49	0.49	0.54	0.60	0.61	0.82	0.65	0.60	0.93
8	0.49	0.54	0.54	0.60	0.71	0.76	0.76	0.65	0.87	0.60
9	0.49	0.54	0.60	0.54	0.54	0.60	0.76	0.82	0.82	0.71
10	0.49	0.43	0.43	0.43	0.54	0.60	0.60	0.60	0.82	0.98

TABLE 4.1.2
DIFFERENCE IN TIMINGS

P.No	2&11 11&2	3&10 10&3	4&9 9&4	5&8 8&5	6&7 7&6
1	0.22	0.11	0.11	0.11	0.03
2	0.05	0.16	0.22	0.16	0.06
3	0.33	0.33	0.27	0.16	0.05
4	0.33	0.27	0.33	0.27	0.09
5	0.17	0.11	0.16	0.11	0.11
6	0.33	0.38	0.33	0.22	0.11
7	0.50	0.11	0.16	0.28	0.01
8	0.11	0.33	0.11	0.16	0.05
9	0.22	0.28	0.22	0.22	0.06
10	0.49	0.39	0.17	0.17	0.06
Mean	0.26	0.21	0.21	0.19	0.06
S.D	0.14	0.11	0.08	0.05	0.03
Paired t	5.2	6.25	7.8	9.5	6

TABLE 4.1.3
OPTIMUM VALUES

P.No	2&11	3 &10	4 & 9	5 & 8	6 & 7	Mean	S.D	Single Salesman
1	421	411	391	406	400	405.8	10.11	382
2	367	379	385	388	389	381.6	08.09	364
3	332	327	337	335	333	332.8	03.37	307
4	350	345	351	349	353	349.6	02.65	322
5	377	366	380	386	386	379.0	07.38	349
6	299	300	298	283	287	293.4	07.01	268
7	320	311	306	322	314	314.6	05.85	280
8	278	292	282	290	289	286.2	05.31	264
9	324	334	336	302	319	323.0	12.23	296
10	343	332	323	302	325	325.0	13.46	293

Problem Size 16+1

TABLE 4.2.1**OPTIMUM SOLUTION TIMINGS**

Ordering of subtours

P.No	2&13	3&12	4&11	5&10	6&9	7&8	8&7	9&6	10&5	11&4	12&3	13&2
1	0.65	0.71	0.87	0.82	0.98	1.26	1.59	1.70	2.41	3.18	2.41	3.73
2	0.87	0.71	0.83	1.04	1.20	1.69	1.92	2.58	3.24	3.89	2.91	4.11
3	0.93	0.65	0.82	0.93	1.09	1.86	2.58	2.19	2.74	1.92	1.81	2.03
4	0.54	0.73	0.76	0.65	0.71	0.87	1.15	1.09	1.31	1.48	1.42	1.48
5	0.93	1.09	0.82	0.76	1.20	1.09	1.20	1.64	1.15	1.31	2.25	1.48
6	0.87	0.60	0.71	0.82	0.98	1.64	1.92	2.03	1.59	2.19	1.37	2.96
7	0.65	0.71	0.54	0.65	0.71	0.76	0.82	1.05	1.09	1.09	1.92	1.70
8	0.98	0.98	0.87	1.09	1.09	1.26	1.59	3.52	2.47	2.74	3.68	5.49
9	0.93	0.98	0.93	0.65	1.20	1.37	1.81	2.08	2.08	2.63	3.02	4.11
10	0.82	0.93	0.65	0.76	0.93	1.26	1.53	1.59	1.37	1.53	3.57	1.86

TABLE 4.2.2
DIFFERENCE IN TIMINGS

P.No	2&13 13&2	3&12 12&3	4&11 11&4	5&10 10&5	6&9 9&6	7&8 8&7
1	3.08	1.70	2.31	1.59	0.72	0.33
2	3.24	2.20	3.06	2.20	1.38	0.33
3	1.10	1.16	1.10	1.81	1.10	0.72
4	0.94	0.69	0.72	0.66	0.38	0.28
5	0.55	1.16	0.49	0.39	0.44	0.11
6	2.09	0.77	1.48	0.77	1.05	0.28
7	1.05	1.21	0.55	0.44	0.34	0.06
8	4.51	2.70	1.87	1.38	2.43	0.33
9	3.18	2.04	1.70	1.43	0.88	0.44
10	1.04	2.64	0.88	0.61	0.66	0.27
Mean	2.06	1.63	1.42	1.13	0.94	0.32
S.D	1.27	0.70	0.79	0.60	0.59	0.17
paired t	4.61	6.61	5.13	5.38	4.55	5.33

TABLE 4.2.3
OPTIMUM VALUES

Ordering of subtours

P.No	2&13	3&12	4 &11	5 &10	6 & 9	7 & 8	Mean	S.D.	Single Salesman
1	350	343	349	345	340	343	345.0	3.51	311
2	372	367	376	373	371	368	371.2	3.02	333
3	402	395	401	405	405	417	404.2	6.64	377
4	339	344	345	340	336	341	340.8	3.02	336
5	365	373	362	359	370	365	365.7	4.68	345
6	357	333	348	340	349	356	347.2	8.47	333
7	429	433	420	420	418	415	422.5	6.34	412
8	422	409	407	408	409	406	410.2	5.39	390
9	390	381	381	360	377	377	377.6	9.01	360
10	366	380	361	354	363	368	365.3	7.91	331

Note: The minimum of the optimal values is presented in bold face for each problem. All the above problems executed on Wipro Super Genius E-590

- i). From 4.1.1 and 4.2.1, it is seen that as n_1 increases(& n_2 decreases), the time for solution increases. However, the choice of n_1 , n_2 is problem specified in general.
- ii). From 4.1.2 and 4.2.2, it is clear that subtour of smaller step length should be preferred as the first subtour in the solution process i.e. if a, p are the sizes of the two subtours, n_1 should be taken as the smaller of a, P . In fact, even the t test is not needed for this

inference; in all the ten cases timing for $n_1 < n_2$ is smaller than for the cases $n_2 < n_1$.

- iii a). An Optimal solution value for single salesman case is always smaller than any of the two salesman cases, for each of the problem. Of course, this is what is only to be expected, since each salesman has to return to head quarter.
- iii b). There does not seem to be any trend in regard to the break up into subtour of different sizes, hence, one can not conclude that among the many partitions as n_1, n_2 any particular choice is aprior likely to give smaller optimal value.

4.5 Min-Max objective and a lexisearch algorithm.

In this and the next section, the single salesman problem with a different objective is considered. A tour may have a total length relatively **small**, but may have one or more of the laps rather very larger, where other tours, with larger total lengths may have all the laps of the tour small values. Hence another meaningful criterion for optimality of the tours is minimality of the largest arc length in a tour. The present and the next section consider this objective function.

A formal statement of the objective function follows:

Let $1 \rightarrow \alpha_1 \rightarrow \alpha_2 \rightarrow \dots \rightarrow \alpha_{n-1} \rightarrow 1$ be a tour. The 'tour value' is now defined as $\max(d_{\alpha_i, \alpha_j})$, where $\alpha_0 = \alpha_n = 1$. Here the objective is to choose a tour which has minimum tour **value**. For instance in the case of illustrative problem given below the tour $1 \rightarrow 2 \rightarrow 4 \rightarrow 5 \rightarrow 3 \rightarrow 1$, has the consecutive

arc lengths 5,1,25,85,and 77. The total **tour length** is 203; but the **tour** value is 85.

Let $D = [d_{ij}]$ $ij = 1,2, \dots, n$ where d_{ij} be the distance from city i to city j .

The lexisearch algorithm to solve the above problem is given briefly below.

Algorithm LEXIMMT

The following notations are used in this algorithm.

W : Current / partial solution

TR : Trial value associated with the trial solution

$$1 \rightarrow \alpha_1 \rightarrow \alpha_2 \rightarrow \alpha_3 \dots \dots \rightarrow \alpha_{n-1} \rightarrow 1$$

Step 1: The alphabet table to this network is an arrangement of all the cities to city i ($i = 1,2,\dots,n$) in increasing order according to their distances.

Step 2: Trial value **TR** is the maximum value of distances of two successive cities in the tour $1 \rightarrow 2 \rightarrow 3 \dots \rightarrow n \rightarrow 1$.

Step 3: Start with the first city (first column) and go to the first possible city from it and let the distance be **DIST**. If it is greater than or equal to **TR** then Stop.

Step 4: **BOUND** is the maximum value of the distances among the first available cities from each city which is not in the partial solution. If this value is greater than or equal to the **TR** proceed in the same column i.e. **JB**.

Otherwise include this city into the partial solution W i.e., GS.

- Step 5: Go to the next possible city from the last city in the partial solution. Now, DIST is the maximum among the distances in the partial solution W and the distance between the last city in W and the first reachable city from it.
- Step 6: If DIST is less than the TR go to step 7. Otherwise drop the last city in the partial solution . If the number of cities in the partial solution is one go to step 2.
Otherwise go to step 5.
- Step 7: Calculate the BOUND as in Step 5. If the BOUND is less than the TR go to step 8.
Otherwise go to step 5.
- Step 8: Include the city in the partial solution. If the partial solution contains n cities go to step 9.
Otherwise go to step 5.
- Step 9: If the DIST is greater than distance between the last city in the partial solution W to city 1 go to step 10.
Otherwise replace the DIST by the distance between the last city in W and the city 1. Goto next step.
- Step 10: If DIST is greater than the TR drop the last two cities in W. Go to step 5.

Otherwise replace TR by DIST and include the city in 1 in W, replace trial solution by W. Drop the last three cities (including recently added city 1) from W . Go to step 5.

4.6 Numerical Illustration

The problem of minimising the maximum step length of a tour is illustrated through a problem with a network of 5 cities.

Distance matrix

$$\begin{bmatrix} 9999 & 5 & 13 & 8 & 33 \\ 7 & 9999 & 34 & 11 & 32 \\ 77 & 8 & 9999 & 90 & 60 \\ 80 & 38 & 65 & 9999 & 25 \\ 32 & 74 & 85 & 55 & 9999 \end{bmatrix}$$

For the solution (1→2→4→5→3 →1), the ‘value’ of the steps are 5, 11, 25,85 and 77 respectively. Hence the value of the tour is the maximum among them, viz. 85. Some other solutions and their values are

(1 2 3 4 5 1)and 90

(1 4 2 5 3 1)and 85

Alphabet table

1	2	3	4	5
2-5	1-7	2-8	5-25	1-32
4-8	4 -11	5-60	2-38	4-55
3-13	5-32	1-77	3-65	2-74
5-33	3-34	4-90	1-80	3-85
1-9999	2-9999	3-9999	4-9999	5-9999

The search table for this problem is

Search Table

$1 \rightarrow \alpha_1$	$\alpha_1 \rightarrow \alpha_2$	$\alpha_2 \rightarrow \alpha_3$	$\alpha_3 \rightarrow \alpha_4$	$\alpha_4 \rightarrow 1$	Remarks
$1 \rightarrow 2-5, 60$	$2 \rightarrow 4-11, 60$	$4 \rightarrow 5-25, 77$	$5 \rightarrow 3-85, 77$	$3 \rightarrow 1-77$ TR=85 X	GS
					GS
					GS
					GS
$1 \rightarrow 4-8, 32$	$2 \rightarrow 5-32, 77$ $2 \rightarrow 3, 34, 60$	$3 \rightarrow 5-60, 80$ $3 \rightarrow 4-90$	$4 \rightarrow 3-65, 60$ $3 \rightarrow 5-60, 32$	$5 \rightarrow 1-65$ TR=65 X	
$1 \rightarrow 3-13, 32$	$4 \rightarrow 5-25, 32$ $4 \rightarrow 2-38, 60$ $4 \rightarrow 3-65$	$5 \rightarrow 2-74$	$3 \rightarrow 5-60, 80$ $3 \rightarrow 4-90$	$5 \rightarrow 1-32$ TR=32 X	
$1 \rightarrow 5-32$	$2 \rightarrow 4-11, 32$ $3 \rightarrow 2-8, 32$ $2 \rightarrow 5-32$	$4 \rightarrow 5-25, 32$	$5 \rightarrow 3-85, 77$	$3 \rightarrow 1-77$ TR=85 X	

In **the** above search table search starts with $\alpha_1=2$ with the value 5. The next letter in the word is the first reachable city from 2 i.e. 4 with **the** value 11. **The** partial solution $1 \rightarrow 2 \rightarrow 4$ value is maximum of the distances between the city 1 to city 2 and city 2 to city 4 i.e. 11. In the above table the bold letters represent the bound values of the partial word. For instance the bound value of the partial solution $1 \rightarrow 2 \rightarrow 4$ is the maximum of the distance of the first reachable city (including city 1) from 4 and the distances of the first reachable city (including city 1) from the cities which are not in the word i.e. maximum of (25,60,32) =60. Similarly the bound for $1 \rightarrow 3 \rightarrow 2 \rightarrow 4$ is 32.

A FORTRAN programme to implement the **algorithm** LEXIMMT is listed in appendix 2

4.7 Computational experience

Computational experience of minimising the maximum step length in a tour is given in the following table for 10 data sets generated randomly on Wipro Super Genius E 590 for each of the problem sizes 11,13, 15,17 and 19.

TABLE 4.7.1**Execution time in Seconds**

P.No.→ Size↓	1	2	3	4	5	6	7	8	9	10	Mean	S.D.
11	0.16	0.16	0.16	0.16	0.16	0.27	0.16	0.21	0.16	0.16	0.176	0.035
13	0.21	0.16	0.16	0.21	0.98	0.16	0.21	0.16	0.21	0.16	0.262	0.241
15	0.82	0.16	0.38	0.49	0.32	0.43	0.16	0.49	0.60	0.38	0.423	0.187
17	0.65	0.38	1.59	2.96	1.75	0.49	0.38	2.41	2.08	1.81	1.45	0.877
19	0.32	0.60	0.71	2.30	1.59	0.93	2.08	3.57	0.38	2.36	1.484	1.021

From the above table, it is clear that, as is to be expected, the mean time increases with problem size. With the limited data available, the model $T = \exp(A + Bn)$ where n is the problem size, appears good enough, with A and B as -4.7948, 0.2696, giving the predicted timings as given below.

TABLE 4.7.2

Problem Size	Mean Time	Predicted Time
11	0.176	0.161
13	0.262	0.275
15	0.423	0.472
17	1.450	0.809
19	1.484	1.388

Remark: The A and B values are computed without using the data of problem size 17.

CHAPTER V

MAXIMUM WEIGHTED CONNECTED SUB GRAPH

5.1 INTRODUCTION

This chapter considers the problem of finding a maximal node weighted connected sub graph of a connected **node-weighted** graph.

In most of the network optimisation problems like the Shortest Route Problem and the Travelling Salesman Problem, the edges which, for example, represent the connectivity of two places have the distance between the two places as their weights. Similarly, in Maximal Flow problems, the edge weight represents the flow capacity between two points. However, there are other optimisation problems in which the vertices, not edges, have weights and one is to find a connected sub graph with a maximum total node-weight.

One such problem occurs in the mining industry : The problem is to choose a set of 'blocks' for mining activity which will give maximum total weight(c.f.Caccets and Giannini,1988). The mining site (not just the surface of the mining area) is divided conceptually into blocks of specific (approximately uniform) volume. These blocks are numbered consecutively for identification. By exploratory studies the net money value that will accrue to the miner if a given block is mined out is assessed (estimated). This value, particularly in blocks near the surface, may well be negative. However, one cannot simply choose a valuable block well below the ground and mine it directly. Physical and technological constraints may require that a block of interest can be mined only if already one of its **specified** neighbours has been mined or, though a

rather rare case, the valuable block itself is on the surface and is directly reachable. Transit cost of moving from one block to another which could be approached directly from the former, is negligible and can be ignored. Mining has to start from a 'Surface block'. Then, a mining block selection strategy is equivalent to naming the blocks to be mined out; the value of this activity is the total of the net values of these blocks while the 'activity set' is sterile unless one of the surface blocks is included in the set. Also for the set to be meaningful, one should be able to approach all the selected nodes ; since the approachability is not directed, it is sufficient if the set of blocks is a connected (not necessarily directly) set. Different such connected sets will mean different Mining Strategies. It may be noted that if two sets have a common block, they are no longer isolated sets and they coalesce into a single connected set.

One need not treat separately the trivial case of having more than one mutually isolated mining strategies, each starting with a 'separate starting block'; one can always 'get them connected' by introducing a fictitious starting block of value say, 0 which is directly connected to each of the possible starting blocks i.e., each of the surface blocks.

The problem can be stated formally in Graph-theoretic terminology as follows:

Let $G = (A, E ; V)$ be a symmetric connected graph with $A = \{ 1, 2, \dots, n \}$ as its node set, and E the edge set; let $V = \{v_1, v_2, \dots, v_n\}$ be the set of values associated with the corresponding nodes.

Let S be a subset of the node set A and let $G(S, E_S; V_S)$ a connected sub graph with E_S and V_S as the corresponding edge set and value set respectively. The value of this sub graph is the total of the values of the nodes in S and is denoted by $Val(G(S, E_S; V_S)) = \sum v_i$

The problem is to select a connected subset S which has the maximum value,

i.e. such that $Val(G(S^*, E_{S^*}; V_{S^*})) = \text{Sup} (Val(G(S_r, E_{S_r}; V_{S_r})))$

As already noted, unlike in the travelling salesman problem, here, the edges of the graph will not have any value attached to them but the nodes will have values v_i , positive or negative as the case may be.

This problem was posed in 1988 by Cacetta and Giannini[1988]. They proposed a graph theoretic approach to solve the problem but except for making some observations and mentioning some procedures to move from one spanning tree to another, do not seem to have solved the problem as such. No solution procedure, let alone computational experience in solving this problem, seems to have appeared so far in the literature. In the present chapter, a lexisearch **algorithm** is developed to solve this problem and also a procedure is developed to reduce wherever possible, the effective size of the problem by ‘merging’ appropriate connected blocks into super blocks, which makes the lexisearch algorithm more effective. A number of randomly generated problems, of varying sizes and varying levels of connectivity have been

solved using this algorithm and the relevant computational experience is reported.

The concepts and the approach of the procedure used to solve this problem are explained in the sequel.

5.2 Computation of Super Nodes

At a first glance, one may think that the problem is trivial: exclude all negative nodes and divide the residual graph into mutually unconnected sub graphs and select the most valuable one among these connected sub graphs. However, this approach is not workable; two connected sub graphs with only positive nodes which are mutually isolated may get connected through a negative node giving a more valuable, bigger connected graph but with a negative node. Also it is often necessary to require that the acceptable sets S should necessarily contain at least one of a prescribed set of (possibly negative) nodes. In what follows, we shall insist that the S of interest must contain at least one negative node. Some of the basic concepts which are used in solving this problem are also explained in the sequel.

Without loss of generality it can be assumed that all the surface blocks are connected with each other since each of them can be directly mined and no transit cost is involved.

One can have many mutually disjoint connected subsets of blocks containing at least one surface block in each of them. All these subsets of **blocks** can be treated effectively as one connected set by assuming links to exist between every pair of surface **blocks** since **any block that can be**

involved in the mining strategy has to be connected to a surface block. This assumption of connectivity of surface blocks makes it possible to consider every set of mutually disconnected subsets with at least one surface block in each as essentially a connected set. Hence one needs to consider only one connected subset of blocks to be mined.

The Maximum node weighted Connected Subgraph problem is solved in two stages. In the first stage, the size of the graph is reduced by constructing supernodes-connected subsets with atleast one negative node; in the second stage, a lexisearch procedure is applied to get a maximal node weighted sub graph from the reduced graph.

5.3 Reduction of the graph : Construction of super nodes

In a given graph there may be clusters of positive nodes which are connected within themselves. However, two such clusters (which are not connected directly with one another) may be connected through a negative node. One can replace all the positive nodes of a cluster by a single node called positive super node. The value of this super node is the total of all the values of the nodes in that cluster. If any negative node is connected to at least one of the nodes in a super node, then it is connected to that positive super node. In this way , the size of the graph is reduced.

The identification of super nodes is done by raising the connectivity sub matrix defined by the positive nodes of the given connected graph to various '**minad**' powers(Pandit,1961) ; this operation is explained in the Appendix 1. After some stage, the pattern gets stabilised. From this stage

onwards no zero will become positive and positives will remain as positive; the minad power sequence will have reached satiety (see Appendix 1). From this matrix one can identify the positive super nodes. This will be illustrated by an example.

5.4 REDUCTION ALGORITHM : REDGRA

Let $G(A, E, V)$ be a connected graph (network) with node-weights, which may be either positive or negative. 0 in the corresponding connectivity matrix represents direct connection between the nodes and 1 stands for 'no direct connection'.

Step 1. Consider the sub graph of nodes whose weights are only non-negative and obtain the submatrix of connectivity for this subgraph.

Step 2. Use the 'compact storage algorithm' to get the satiated matrix. From this matrix, identify the mutually disjoint sets of nodes which are connected within themselves.

Step 3. Construct the new graph with positive super nodes and original negative nodes.

FURTHER REDUCTION

step 4. If any positive node (not the super node) is connected to only one negative node, then both these nodes can be merged and the value of this node is the total of the two node-values.

- Step 5. If a negative node is connected to only one negative node, and dropping of this node not effecting the connectivity then drop it from the graph.
- Step 6. If a negative node (3 is not directly connected to any of the positive super nodes and removal of (3 is not affecting the connectivity of the remaining graph then drop it from the graph.
- Step 7. Repeat Step 4, Step 5 and Step 6 till no further merging and dropping of nodes is possible.

5.5 LEXISEARCH ALGORITHM : LEXIGRA

The following notation is used in the sequel.

- W** A (dynamically changing) set of negative valued nodes
- L** The list of all negative nodes.
- AT** Alphabet Table; defined through columns, one for each negative node, consisting of nodes connected to the negative nodes, arranged in the decreasing order of their values.
- TR** Value of a trial solution. Initially, it may be set at $-\infty$.
- B** Bound, at any stage, B is the total of the values of positive supernodes not included in W.

The LEXIGRA algorithm follows:

- Step 1. Start with first available negative node from L. If list is exhausted, then stop.
Otherwise, take next node in the list and put this in the array W.
- Step 2. Take all the positive nodes which are connected to this negative node, if any. The value of set W, $V(w)$ is the total of all the positive super node weights (Value) and the value of this negative node.
- Step 3. Obtain B = the total of all positive super nodes which are not in the W.
- Step 4. If $(B + V(W))$ is less than TR , drop this negative node and go to step 1.
Otherwise go to next step
- Step 5. Take the first available negative node in the column which represents the last node in the W and put it in W. If there is no negative node available in that column, go to step 10.
Otherwise go to next step
- Step 6. Calculate $V(W)$ as explained in the step 2.
- Step 7. If the $V(W)$ is greater than TR , then $TR = V(W)$; go to step 8.

Step 8. Calculate the bound (as in step 3).

Step 9. If $(V(W) + B)$ is greater than or equal to TR , go to step 5. Otherwise drop the recently added two negative nodes (and positive nodes connected to these negative nodes), reduce the word value appropriately. If the word contains no negative node then go to the step 1. Otherwise go to the step 5.

Step 10. Drop the last letter from W . If the word contains no negative node then go to the step 1. Otherwise go to step 5.

As the 'positive(valued)' nodes are nodes numbered 12,13,.....,24, we have, by taking the submatrix defined by rows and columns 12,13,.....,24, the connectivity matrix for the subgraph consisting only of the positive nodes and the edges of the graph which are connecting them as the positive nodes submatrix

12	13	14	15	16	17	18	19	20	21	22	23	24
0	1	0	1	1	1	1	1	1	1	1	1	1
1	0	1	1	0	0	1	1	1	1	1	1	1
0	1	0	0	1	1	0	1	1	1	1	1	1
1	1	0	0	1	1	1	1	0	1	1	1	1
1	0	1	1	0	1	1	0	1	1	1	1	1
1	0	1	1	1	0	1	0	1	1	1	1	1
1	1	0	1	1	1	0	1	0	1	1	1	1
1	1	1	1	0	0	1	0	1	1	1	1	1
1	1	1	0	1	1	0	1	0	1	1	1	1
1	1	1	1	1	1	1	1	1	0	0	1	0
1	1	1	1	1	1	1	1	1	0	0	0	1
1	1	1	1	1	1	1	1	1	1	0	0	0
1	1	1	1	1	1	1	1	1	0	1	0	0

These Super nodes along with the corresponding weights are presented below.

Supernode	Members of Supemode	weights
A	21,22,23,24	$12+13+14+15=54$
B	12,14,15,18,20	$3+5+6+9+11=34$
C	13,16,17,19	$4+7+8+10 =29$

The reduced graph connectivity matrix is

	A	B	C	1	2	3	4	5	6	7	8	9	10	11
A	0	1	1	1	0	0	1	1	1	1	1	1	1	1
B	1	0	1	1	1	0	0	1	1	1	1	1	1	0
C	1	1	0	1	1	1	0	0	1	1	1	1	1	1
1	1	1	1	0	1	1	1	1	1	1	1	1	0	1
2	0	1	1	1	0	1	1	1	1	1	1	1	0	1
3	0	0	1	1	1	0	1	1	1	0	1	1	1	1
4	1	0	0	1	1	1	0	1	1	1	1	1	1	1
5	1	1	0	1	1	1	1	0	1	1	0	1	1	0
6	1	1	1	1	1	1	1	1	0	1	1	1	0	0
7	1	1	1	1	1	0	1	1	1	0	1	1	0	1
8	1	1	1	1	1	1	1	0	1	1	0	1	1	1
9	1	1	1	0	1	1	1	1	1	1	1	0	1	0
10	1	1	1	1	0	1	1	1	0	0	1	1	0	1
11	1	0	1	1	1	1	1	0	0	1	1	0	1	0

The initial trail solution can consist of the first negative node and positive nodes connected to it if any, in this illustration $W = \{1\}$, since no positive node is connected to it; its value is $V(W) = -1$

The alphabet table for this Problem is

1	2	3	4	5	6	7	8	9	10	11
	A 54	A 54 B 34	B 34 C 29	B 34 C 29					B 34	B 34
9 -13	3 -4 10 -14	2 -2 4 -5 5 -7 6 -8 7 -9 10-14 11 -15	3 -4 5 -7 6 -8 10-14 11-15	3 -4 4 -5 8 -11 10-14 11 -15	3 -4 4 -5 10-14	3 -4 10-14	5 -7	1 -1 11 -15	2 -2 3 -4 4 -5 5 -7 6 -8 7 -9 11-15	3 -4 4 -5 5 -7 6 -8 9 -13 10-14

In the above alphabet table the nodes written in bold font are connected to the negative node of that column through a positive super node. For instance , in the column 2 the entry 2 i.e. the node 3 is not directly connected to node 2 but connected through the supernode A.

The Search Table for the illustration problem follows:

The Search Table

1	2	3	4	5	6	7	Remarks
1=-1 -1,117							GS
	9 = -13 -14,117						GS
		11=-15 -29+34=5 5,83					GS
			3=-4 1+54=55 55,29				GS
				2=-2 53,29			GS
					10=-14 39,29		GS
						4=-5 34+29=63	TR=63 JO
						5=-7 32,29	JO
					X		
				4=-5 50+29=79 79,0			TR=79 JO
			4=-5 0,85				
				3=-4 -4+54=50 50,29			JO
			5=-7 -2+29=27 27,45				GS
				3 =-4 23+54 77,0			JO
			6=-8 -3,83				GS
				3=-4 -7,83			JO

1	2	3	4	5	6	7	Remarks
			9=-13 -8,83				JO
2=-2 -2+54=52 52,63							GS
	3=-4 48+34=82 82,29						GS
		4=-5 77+29=106 106,0					TR=82 GS TR=106 JO
	10=-14 38,63						JO
3=-4 84,29							GS
	4=-5 79+29=108 108,0						TR=108 JO
4=-5 58,54							GS
	5=-7 51,54						JO
5=-7 56,54							GS
	8=-11 45,54						JO
6=-8 -8,117							GS JO
	10=-14 -23+34=12 12,73						
7=-9 -9,117							JO
	10=-14 -23+34=11 11, 73						
8=-11 -11,117							STOP

In any row or any column of the above search table, the letter on left-hand side of the equality represents the negative node and the letter on right-hand side represents the value of that node. The first number under the negative node is the cumulative value of the solution $V(W)$ and the

bold number to the right of cumulative value is B the value of the bound.

For instance, in the third row and third column of the search table the solution is $1 \rightarrow 9 \rightarrow 11$ and its value is value of node 1 + value of node 9 + value of node 11 plus the sum of the Positive **Supernodes** connected to at least one of these negative nodes. In this solution node 11 is connected to Positive Supernode **B**, of value 34. Hence, the value of the solution (1,9,11) is $-1-13-15+34=5$. The bound to this solution is the sum of the values of the positive super nodes A and C i.e. $54 + 29 = 83$. Similarly the value of the solution $1 \rightarrow 2 \rightarrow 3$ is $(-1) + (-2) + (-4) + (54+34) = 82$ and the bound is 29

The optimal solution to this problem is A B C 3 4 and its value is

$$54 + 34 + 29 + (-4) + (-5) = 108$$

A listing of the FORTRAN programme for this algorithm is given as appendix 2,

5.7 Computational Experience

In order to assess the efficiency of the proposed algorithm, a number of networks, of different sizes (i.e., number of nodes with different number of positive and negative nodes) and with different levels of connectivity were solved on Wipro Super Genius E 590 with the FORTRAN Programme listed in the appendix, and the times required for obtaining solution of the same were recorded and analysed. Methods adopted

for generating a network with a specified structure will be **explained first; the** times taken for solving the different problems with different **specified** structures will then be presented in appropriate tables. A statistical analysis of these data and the conclusions drawn from the same are also presented.

After deciding the level of connectivity ,generate a uniform random number for each pair of the nodes , if it is less than the level of connectivity then there exists a direct connection between that node pair ,which is represented by 0; otherwise, there is no direct connection between them, which fact is represented by 1. Similarly the percentage of positive nodes in a graph is also decided. For each node a uniform random number is generated to decide the nature of the node. For instance, the graph with 60% positive nodes is generated by generating uniform random number for each node; if it is less than 0.6, then the node will be positively weighted; otherwise the node is negatively weighted. The weights are themselves taken as uniform random numbers in the range 0-200. Thus, if weight-random number is 70 and the node is a negative node, the weight for that node is taken as -70

10 node weighted graphs (networks) were generated for each of the different combinations of size, connectivity and % of positive node. The optimum values and time(in seconds) taken for each of these 10 problems **are recorded**. The details are given in appendix 3. A summary analysis of **these data** (on times for solution) by ANOVA is given below, treating the

data as coming from a 3-Factor experiment with levels 2x3x3, with 10 replicates for each 3-Factor combination.

TABLE 5.7.1

Size	Percentage of		Time		No of super nodes	
	connectivity	Positive Nodes	Mean	S.D	Min	Max
75	10	10	0.29	0.0245	3	7
75	10	20	0.29	0.0245	2	5
75	10	30	0.29	0.0245	2	10
75	20	10	0.304	0.035	1	7
75	20	20	0.273	0.031	2	5
75	20	30	0.263	0.0303	1	2
75	30	10	0.292	0.044	1	3
75	30	20	0.246	0.029	1	1
75	30	30	0.27	0.0	1	1
100	10	10	0.366	0.0418	4	8
100	10	20	0.393	0.0558	2	7
100	10	30	0.378	0.025	2	5
100	20	10	0.471	0.036	2	5
100	20	20	0.443	0.034	2	5
100	20	30	0.295	0.025	1	1
100	30	10	0.426	0.128	1	2
100	30	20	0.311	0.02	1	1
100	30	30	0.285	0.023	1	1

TABLE 5.7.2**ANOVA TABLE**

S.V	Sum of Squares	d.f	Mean Square	Variance Ratio
$\alpha \dots$	0.4014	1	0.4014	191.1429
$\dots \alpha \dots$	0.0403	2	0.0202	9.6190
$\dots \alpha$	0.1129	2	0.0565	26.9048
$\alpha \times \alpha \times \alpha \dots$	0.0254	2	0.0127	6.0476
$\alpha \dots \times \alpha$	0.0563	2	0.0282	13.4286
$\dots \alpha \times \alpha$	0.1078	4	0.0270	12.8571
$\alpha \times \alpha \times \alpha \times \alpha$	0.0351	4	0.0088	4.1905
Error	0.0351	162	0.0021	
Total	1.1229	179	0.0021	

5.8. COMMENTS

- i. Even with relatively large number of nodes and low connectivity the number of super nodes turns out to be very small.
- ii. For connectivity less than 5%, graphs that were generated were un-connected, details of this case have not been reported.
- iii. Time taken, for a particular combination of size, connectivity etc., appears to be varying very little, over the randomly generated problems, C.V. being not more than 15.0 except for one case (100,30,10) whose C.V. is 30%.
- iv. With increasing connectivity number of super nodes reduces, almost to 1.

- v. With increasing number of positive nodes, number of super nodes reduces, almost to 1.
- vi. With increasing number of nodes, number of super nodes seems to increase slightly.

At first appearance, one is tempted to conclude that

- vii. With increasing connectivity, no effective change in time.
- viii. With increasing number of positive nodes no effective change in time.
- ix. With increasing number of nodes (size) average computational time is more.

It should be noted that except in the case (100,30,10) (with S.D. = 1.28) and (75,30,30) (with S.D.=0) , the S.D. in each of the various factor combinations is of order 0.03. However, from the ANOVA table for time for solution, it is obvious that compared to the error variance, all other variances are quite large, the smallest ratio being 4.19 for 3 factor interaction, the next higher variance ratio is 6.0. Hence , though one can not use the F ratio test here, it is numerically evident that at least upto 2nd order interactions these factors do have an influence in time for solution.

CHAPTER VI

QUADRATIC ASSIGNMENT PROBLEM

6.1 INTRODUCTION

In this chapter, the Quadratic Assignment Problem(QAP,for short) is discussed. The earlier approaches for solving the QAP are briefly presented along with a critical appraisal of the same. A new implicit enumeration approach for tackling the same is presented. This chapter ends with a report on computational experience regarding this algorithm.

Many practical optimisation problems concerning the assignment of discrete facilities to discrete locations are combinatorial in nature. An important problem of this type which arises in a diversity of contexts is known as the Quadratic Assignment Problem. For convenience this type of problem is often referred as facility location problem so that it indicates one of the common contexts in which it is used as a layout problem.

Quadratic Assignment has many applications in different fields, some of which are listed below: (c.f. Burkard and Stratmann, 1978)

- 1.**Blackboard** wiring problems.
- 2.Plant location problems.
- 3.**Development** of new typewriter keyboard.

These problems differ from the classical linear assignment problem in that the objective function is quadratic while the constraints are the same as in the case of the ‘linear’ assignment problem.

The Q.A.P. as given by Koopmans and Beckman (1957) can be formulated as follows.

$$\text{Minimise } \sum \sum \sum \sum f_{ik} d_{jl} x_{ij} x_{kl} \quad (6.1)$$

$$\text{subject to } \sum_{i=1}^n x_{ij} = 1, \sum_{i=1}^n x_{ji} = 1, j=1,2,\dots,n \quad (6.2)$$

$$x_{ij}^2 = x_{ij} \quad i,j = 1,2,\dots,n \quad (6.3)$$

The above problem can be interpreted as follows:

There are 'n' indivisible facilities to be assigned to n indivisible locations; f_{ik} is the flow or interaction from facility i to facility k and d_{jl} is the distance from location j to location l. The objective is to assign the facilities to locations such that the sum of pair-wise interactions among facilities, weighted by the distance between their respective locations, is minimised.

In the above definition of the problem, the distance between a pair of locations j and l is not necessarily assumed to be the same in both the directions i.e. in general $d_{jl} \neq d_{lj}$. Similarly, the flow or interaction between facilities i and k is also not necessarily symmetrical i.e. $f_{ik} \neq f_{kl}$.

If at least one of the matrices, D or F, is symmetric then the above QAP is known as symmetric QAP.

The QAP can be illustrated through the following example.

A medical complex has 4 facilities. Among them there will be flow of patients. Each of these facilities is to be located in one of the four locations selected in the site of the complex. The distances among the four locations are given and also the flow (or movement) of patients among facilities in a typical day i.e. the flow pattern between the facilities is also specified.

The total movements and hence the total cost of the 'flow' depends upon the pattern of allocations of these facilities to the locations.

Let the matrix of distances (equivalently ,the cost) between locations and the matrix of flows from facility to facility be D and F respectively as given below.

Distance matrix

	a	b	c	d
a	X	6	7	2
b	6	X	5	6
c	7	5	X	1
d	2	6	1	X

Flow matrix

	1	2	3	4
1	X	10	20	5
2	18	X	9	4
3	5	6	X	8
4	8	0	15	X

Let the assignment of the facilities to locations be $\begin{pmatrix} f & a & b & c & d \end{pmatrix}$

Thus, ten patients are to be moved from facility 1 to facility 2 **but** these facilities are located at locations a and b; hence the total of the distance travelled by these patients is $10 \times 6 = 60$ units. Similarly **for the**

distance travelled by the other patients. Hence, with this allocation, the **total** distance travelled by all the patients is $(10 \times 6 + 20 \times 7 + 5 \times 2) + (18 \times 6 + 9 \times 5 + 4 \times 6) + (5 \times 7 + 6 \times 5 + 8 \times 1) + (8 \times 2 + 0 \times 6 + 15 \times 1) = 461$.

If the same facilities were located as per the allocation $\begin{pmatrix} a & b & c & d \\ 1 & 4 & 2 & 3 \end{pmatrix}$ the total distance would be $(5 \times 6 + 10 \times 7 + 20 \times 2) + (8 \times 6 + 0 \times 5 + 15 \times 6) + (18 \times 7 + 4 \times 5 + 9 \times 1) + (5 \times 2 + 8 \times 6 + 6 \times 2) = 477$.

Thus, for the same physical space with same flow pattern among the facilities, the total distance (or equivalently, cost) depends on the actual allocation of individual facilities to individual locations. This allocation can be conveniently represented by a permutation or equivalently, by a permutation matrix $[x_{ij}]$; when the locations as well as the facilities could be identified with the same set of symbols 1, 2, 3, 4. Thus, in the above

example, the allocations $\begin{pmatrix} a & b & c & d \\ 1 & 4 & 2 & 3 \end{pmatrix}$ can be represented as permutation

$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 4 & 2 & 3 \end{pmatrix}$, where the 1, 2, 3, 4 of the first row stand for the locations a, b

c, d respectively, while the symbols in second row stand as earlier for the facilities. This same assignment (allocation) can be represented by the following permutation matrix:

$$X = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

The value of the objective function can be written as

$$\sum f(i,k) d(j,l) x_{ij} x_{kl} \quad (6.4)$$

6.2 A SURVEY OF AVAILABLE ALGORITHMS

Since n facilities can be assigned to n locations in a finite number of ways, solution by complete enumeration, which consists of evaluating all possible layouts and choosing a best, may be considered as a solution method. However, as is true about combinatorial problems in general, it can be quickly dismissed as computationally infeasible for problems of even moderate size which are of practical interest.

It is to be noted that 'heuristic, manual, intuitive approaches attempted in early 1950's' (c.f. Nugent et al 1968) are not practicable even for problems of very moderate size; hence they do not require further elaboration.

Among the other methods, the algorithms can be classified as those based on branch and bound approach and those based essentially on linear programming.

Some of the algorithms based on branch and bound approach were developed by Gilmore (1962), Lawler (1963), Land (1963), Gravett and Plytter (1966), Graves and Whinston (1970), Pierce and Crowston (1971),

Burkad and Stretmann (1978), Roucairol (1978), Bazaraa and Elshafei (1979), and Edwards (1980).

The second approach is based on converting the problem by reformulating the same as L.P.P through an appropriate transformation. The algorithms of Love and Wong (1976), Kufman and Brockx (1978), and Bazaraa and Sherali (1980), belong to this category.

Owing to the complexity of the QAP., in general, none of the above algorithms can solve problems of size $n > 15$ effectively (c.f. Bazaraa and Kirca 1983). Hence, interest has been focused on suboptimal procedures.

6.3 SUB-OPTIMAL PROCEDURES

Suboptimal methods can be divided into the following classes.

6.3.1 Construction Methods

Starting with a partial solution or the null assignment, a complete assignment is reached iteratively by locating one or more facilities at each iteration.

6.3.2 Improvement Methods

Starting with a complete assignment of facilities, an improvement over the incumbent objective function value is sought by interchanging the locations of several facilities. The procedure is terminated when no further improvements are possible.

Hillier(1963) algorithm is based on a 'Move Desirability Table'(M.D.T). The algorithm is restricted to making 2-location exchange between adjacent locations. Subsequently, Hillier and Connors (1966) modified the Hillier's(1963) algorithm. In this latter algorithm, they consider the possibility of exchanges of non-adjacent locations too.

Armour and Buffa (1963) suggested random selection of starting solutions and improving them. Their algorithm evaluates the effect on the cost function of all $n(n-1)/2$ 2-department exchanges and selects the one which leads to the greatest cost reduction from the starting solution.

Nugent et al (1968) developed a procedure where they suggested random selection of starting solutions; improvement is sought by randomly selected exchanges of locations, the selection probability being proportional to a monotonic function of cost reduction.

Heider (1973) gives a two-variable branching process which considers the placement of facility and location simultaneously at every step. However, the computations required to calculate the bound are rather heavy.

A comprehensive survey of heuristics methods can be found in the works of Burkard and Stratmann (1978)and Sherali (1979).

In symmetric case, for any solution there will be another mirror image solution which will have the same value. Using this fact Bazarra and Kirca (1983) developed a heuristic based on the branch and bound approach.

6.4 OPTIMAL SOLUTION PROCEDURES

Optimal solution procedures based on branch and bound approach for solving the Q.A.P. can be classified as follows:

6.4.1 Single assignment algorithms

In these algorithms, at each stage, one unassigned facility is assigned to an unoccupied location according to some criteria.

Gilmore (1962) developed a direct branch and bound algorithm for this problem. However, the bound setting procedure requires listing of a large number of quantities in ascending and descending order, the set of these numbers itself changing as the algorithm proceeds. The procedure, therefore, appears to require an inordinate amount of bookkeeping and computation. This appears to be the main reason for the computational infeasibility of the procedure when n is above 15.

Though this method is applied to symmetric case, it can be adapted to the asymmetric case also. However, the computational efforts will become much heavier.

The works of Lawler (1963), and Burkard and Stratmann (1979) are some further examples of the single-assignment algorithm.

6.4.2 Pair assignment algorithms

These algorithms proceed by simultaneously locating two unassigned facilities to two unassigned locations. Land (1963) proposed an algorithm based on this principle. Gavett and Plyter (1966) suggested conversion of an n^{th} order 'symmetric distances' location assignment problem into an 'equivalent' $N = n(n-1)/2^{\text{th}}$ order assignment problem; the latter is solved by a branch and bound technique. The size of the equivalent assignment problem increases very rapidly with the increment of the number of facilities. Also, it is to be noted that in the equivalent $N \times N$ assignment matrix, there are $N! \gg n!$ permutations to be dealt with. As noted by Das (c.f. Das, 1976) not all the N^{th} order assignments are acceptable as n^{th} order facility location allocations and hence, the optimal solution for the N^{th} order assignment problem need not be even a facility location solution. Then next best solutions of N order assignment problems are to be generated sequentially (which itself is a challenge) and tested for acceptability.

The problem of the large size of N is made worse when the distance matrix is asymmetric. Then the value of N becomes $n(n-1)$ i.e. double the value of N in the symmetric case.

It is therefore, obvious that computationally, even a complete enumeration may be more practicable than attempt at solution by such conversion of facility location problem to an 'equivalent' 'linear' assignment problem.

6.4.3 Pair - Exclusion algorithms

Pierce and Crowston (1971) presented a tree search algorithm. However, the algorithm requires 'construction and solution' of assignment problems at every 'decision step', which also involves heavy computations.

6.4.4 Lexisearch

Before Branch and Bound method (1963) another implicit enumeration approach viz. lexicographic search (shortly lexisearch) had been developed and used (Pandit, 1962) in the context of the Loading (Knapsack) problem. Compared to lexisearch BB method requires heavy computation and a large amount of memory. The disadvantage with BB when compared to lexisearch is that it lacks natural hierarchy and hence requires computations for back-tracking.

Das(1976) developed a lexisearch algorithm for the facility location problem. She divided the objective function into two parts, one of them is independent of and the other is depending on the permutation for an allocation. After removing the constant factor from the objective function, she developed an alphabet table. She calculated the bound for every newly added facility in the word, which would require more time. Keeping in view these disadvantages, in the proposed lexisearch algorithm a new alphabet table is constructed which is totally different from that of Das. In the proposed algorithm the locations are fixed; Under these locations, the facilities are filled in a lexicographic order. *The bounds both conditional and unconditional which are used in the present case are not to be computed dynamically but are computed once for all and require relatively*

less time. The computational experience shows the present algorithm is superior to that of **Das(1976)**.

6.5 Lexisearch Algorithm LEXIQAP

TR The value of the trial solution $\begin{pmatrix} 1 & 2 & 3 & . & . & . & n \\ \alpha_1 & a_2 & a_3 & . & . & . & a_n \end{pmatrix}$

FW Current word/partial solution: subset of the facilities.

V(FW) Value of FW

UB Unconditional Bound

CB Conditional Bound

Here the alphabet table consists of the facilities in natural order, i.e., each column consists the elements in the order 1,2,..n.

Now for the algorithm:

Step 1: Start with the first available facility under the first location and keep this facility in the(incomplete solution) word **FW**. If there is no facility available in the first column, stop.
Otherwise go to the next step.

Step 2: Look for the first available facility under the second column. If it is available keep this facility under location 2; go to step 3.
Otherwise drop the first facility in the word **FW** and go to step 1.

- Step 3. Calculate the actual value $V(FW)$ of this partial solution. If it is greater than or equal to TR , drop the second facility from FW and proceed in the second column.
Otherwise go to the next step.
- Step 4. If $(V(FW) + UB)$ is less than TR , go to step 5; Otherwise, drop the recent facility and continue in the same column. If no facility is available in that column then drop the last facility in FW and go to Step 1.
- Step 5. Find the first available facility under the column of the alphabet table which is represented by the last facility in the word FW ; then go to step 6. If no facility is available in that column, drop the last facility in the word FW ; go to step 2.
- Step 6. If $(V(FW) + CB(\text{for that specified facility}))$ is less than TR , go to the next Step 7.
Otherwise, go to Step 5.
- Step 7. Include this facility in the word FW . Calculate $V(FW)$. If the word FW contains n elements then go to step 13.
Otherwise, go to step 8.
- Step 8. If $(V(FW) + UB)$ is less than TR then go to step 10.

Otherwise continue in the same column which is represented by the last facility in FW. If no facility is available in that column then go to Step 11.

Step 9. Find the first available facility under the last facility in the word in the corresponding column of the alphabet table. Go to step 10. If no facility is available in that column, go to step 11.

Step 10. If $(V(FW) + CB)$ is less than the TR, go to step 7. Otherwise, go to step 9.

Step 11 . Drop the last facility in the FW. If the FW contains only one element the go to Step 2. Otherwise, go to next step.

Step 12. Find the first available facility under the alphabet column for the last facility in FW. Go to step 10. If no facility is available, go to step 11.

Step 13. If the $V(FW)$ is less than the TR, replace TR by $V(FW)$ and the trial solution by word FW . Otherwise, drop the last two facilities in the FW and go to step 9.

6.6 Numerical Illustration

The lexisearch algorithm is illustrated with the following example.

Let the following be the distance and flow matrices for a QAP of size 5.

$$D = \begin{bmatrix} 9999 & 9 & 5 & 4 & 7 \\ 9 & 9999 & 8 & 3 & 6 \\ 5 & 8 & 9999 & 3 & 9 \\ 4 & 3 & 3 & 9999 & 7 \\ 7 & 6 & 9 & 7 & 9999 \end{bmatrix}; F = \begin{bmatrix} 9999 & 7 & 5 & 9 & 3 \\ 7 & 9999 & 1 & 8 & 4 \\ 5 & 1 & 9999 & 2 & 9 \\ 9 & 8 & 2 & 9999 & 8 \\ 3 & 4 & 9 & 8 & 9999 \end{bmatrix}$$

The alphabet table consists of n rows and n columns. The ordering of each column is the natural alphabet order namely 1,2,..., n in each. In the search table, the first row and first column the entry is α_1 , the contribution of this partial solution to objective function is nil. In the second row second column, the partial solution is α_1, α_2 the contribution of this solution to objective function is calculated sequentially

$$\begin{pmatrix} 1 & 2 \\ \alpha_1 & \alpha_2 \end{pmatrix} \text{ gives } d(1,2)*f(\alpha_1, \alpha_2) = C_2, \text{ by definition } C_1 = 0$$

Cumulative value corresponding to the incomplete word $\alpha_1, \alpha_2, \alpha_3$ is

$$C_2 + d(1,3)*f(\alpha_1, \alpha_3) + d(2,3)*f(\alpha_2, \alpha_3) = C_3$$

In general for an incomplete word $\alpha_1, \alpha_2, \alpha_3, \dots, \alpha_k$, C_k is the cumulative value for the leader of the super block $(\alpha_1, \alpha_2, \dots, \alpha_{k-1})$ plus the total of $k-1$ terms due to allocation of α_k at k^{th} location.

$$C_k = C_{k-1}(\alpha_1, \alpha_2, \dots, \alpha_{k-1}) + \sum_{r=1}^{k-1} d(r,k)*f(\alpha_r, \alpha_k)$$

The minimum value to be added to the given incomplete word $\alpha_1, \alpha_2, \dots, \alpha_{k-1}$ when α_k is placed under the location k is sums of the products of the k elements of the distance matrix arranged in decreasing order and k smallest elements of α_k arranged in increasing order. Thus, for instance, in the illustration, when facility 3 is placed under location 3 the minimum to be added to the objective function is

$$(8,5) * \begin{pmatrix} 1 \\ 2 \end{pmatrix} = 8*1 + 5*2=18$$

Hence, for any given incomplete word of length k , the contribution to be got from the remaining columns consists of sum of the products of $k, k+1, \dots, n-1$ pairs, the entries being the super diagonal terms in the $k+1^{\text{th}}$ column of distance matrix arranged in decreasing order and the smallest k elements of the $(k+1)^{\text{th}}$ column of F arranged in increasing order and so on. So, each of these have the corresponding lower bound the super diagonal terms arranged in decreasing order in the distance matrix D and the smallest k elements in increasing order in column $(k+1)$ of the F matrix respectively. Hence to compute unconditional bound components the following procedure is adopted.

Construct the matrices A_1 , A_2 as follows:

The columns of A_1 consist of the super diagonal entries in the corresponding column of the distance matrix written in the decreasing order. The columns of A_2 are the entries of the corresponding column of the F matrix arranged in increasing order.

Thus, for our illustration example we have

A_1 ,

A_2

$$\begin{bmatrix} 9999 & 9 & 8 & 4 & 9 \\ & 9999 & 5 & 3 & 7 \\ & & 9999 & 3 & 7 \\ & & & 9999 & 6 \\ & & & & 9999 \end{bmatrix} \quad \begin{bmatrix} 3 & 1 & 1 & 2 & 3 \\ 5 & 4 & 2 & 8 & 4 \\ 7 & 7 & 5 & 8 & 8 \\ 9 & 8 & 9 & 9 & 9 \end{bmatrix}$$

Hence, the contribution to the objective function from column 4(i.e.location 4) irrespective of what facility is placed under the location 4 is at least 25(i.e., minimum among 48,37,25,56). If α_4 is 2 the minimum contribution is 37. Thus we have unconditional bounds for the contribution for $\alpha_2, \alpha_3, \alpha_4, \alpha_5$ respectively as 9,18,25,112. Hence the set of cumulative bounds in case of words which terminate at $k = 4,3,2$ are 112,137,155,164 respectively.

However, another set of unconditional bounds can also be worked out by considering the entire set of $n(n-1)$ items from each of the F and D matrices. A global lower bound taking all the ten pairs into account is got as $3*9 + 3*9 + \dots 9*1 = 282$.

When α_2 is specified and others are yet to be specified i.e. when the incomplete word is of length 2, the unspecified cost of the objective function should consist of sums of products of $10-1 = 9$ pairs of elements and is obviously given by recomputing the sums of product by omitting the

largest entries in each of D and F values. In the present case, this value is
 $3*9+3*8+...+9*1=222$

Distance entries 3 3 4 5 6 7 7 8 9 9

Flow entries 9 9 8 8 7 5 4 3 2 1

Similarly, the cumulative bound for the incomplete word of length 3 can be obtained as the sum of the products of $10-3=7$ pairs by omitting the next largest entries in the two sets D and F of the matrices and is given by
 $3*8+3*7+.....+7*1 = 124$

Thus, the present problem has two sets of unconditional bounds for words of Length : 2 3 4 5

Bound1 : 164 155 137 112

Bound2:282 222 124 34

Hence, these being lower bounds only, a better lower bound set is obtained by taking the larger of the pair in Bound1 and Bound2 to get the unconditional bound

UB:282 222 137 112

Table A_3

Under location

2	3	4	5
27	49	48	165
9	28	37	134
9	18	25	112
18	56	56	184
27	44	48	165

Table A_4

Under location

2	3	4	5
249	198	160	165
231	187	149	134
231	167	159	112
240	205	168	184
249	193	160	165

This array of bounds can be used as unconditional bounds for an unspecified letter α_k in position k . However, when α_k is specified for the incomplete word of length k , the contribution from that column as a lower bound equals the corresponding entry in the table A_3 plus the cumulative bound corresponding to the next column(right to it), this can be worked out as shown in table A_4 .

Here, the bound corresponding to a particular k will exclude the cumulative bound for the same index in the column $k+1$ etc.,. Hence, the conditional bound for a given α_k if the cumulative bound for the same index at column $k+1$ is to be excluded and hence the corresponding cumulative bound for α_k is the bound for the column plus the minimum of the bound column to the right excluding the index α_k in the column $k+1$. Thus for instance, $\alpha_5=3$ gives the minimum entry under column 5 in A_3 . Hence, when α_4 is 3, the bound for 5 is the smallest entry in that column excluding the value 112 i.e. 134. Similarly the conditional cumulative bounds can be computed. The result is shown in table A_4 .

Further, it is noted that one of the bounds can be larger than all the entries of a column in which case the entries in the table A_4 will be inoperative as bounds and the best available bound, namely the entry from the row Bound B itself may be used. After doing this manipulation, the best conditional bound is given by

Conditional Bound Matrix

$$\begin{bmatrix} 222 & 160 & 165 \\ 222 & 149 & 134 \\ 222 & 159 & 112 \\ 222 & 168 & 184 \\ 222 & 160 & 165 \end{bmatrix}$$

The first allocation that naturally occurs to mind namely $\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 4 & 5 \end{pmatrix}$ is taken as trial solution with the value $(63+25+36+21) + (8+24+24) + (6+81) + (48) = 344$

In the first row and first column the permutation is $\alpha_1 = 1$. Before going to the second allocation, we compare the global bound 282 with the trial solution value 344, since trial solution value is greater than the global bound we proceed to column 2. In the second row and second column the allocation is $\{1 \ 2\}$. Here the unconditional bound is 222 and the value of

this partial solution is 63. Similarly for the allocation 1 2 3 the value of this is $63+33=96$ and the unconditional bound is 149. When the facility 4 is specified, the conditional bound is the minimum contribution to the objective is 168(which is (4,3)th element in A_3). **Before** going to calculate the value of the solution 1 2 3 4, we compare the value (96+168) which is less than TR. Since it is less than TR we calculate the actual value of {1 2,34 }.

In the following search table, the bold letters represent the conditional bound. The search table for the illustration is given below.

The search table for the illustration is given below:

The Search Table
Locations

1	2	3	4	5	Remarks
1	2--63 63,222	3-33 96,149	96,168 4-66 162,112	162.165 5-182 344>TR X	GS GS GS
			96,160 5-59 147,112	147,184 4-185 332=TR X	GS
		63,222 4-109 172,149	X		GS
			172,159 3-29 201,168	201,165 X	JB JB
		63,222 5-47 110,149	172,168 X		
			110,159 3-50 160,112	160,184 X	GS JB
			110,168 4-84 194,112		GS

Locations

1	2	3	4	5	Remarks
				194,112 3-136 330 =TR X	JB
	3-45 45,222	X	X		GS
		45,222 2-43 88,149	88,184 4-66 154,112		GS
				154,165 5--167 321=TR X	GS
			X		JB
		45,222 4-61 106,149			GS
			106,149 2-55 161,112		GS
				161,165 X	JB
			106,160 5-63 169,112		GS
				169,134 2-155 324>TR X	JB
			X		
		45,222 5-87 132,149			GS
			132,149 2-43 175,112		GS

Locations

1	2	3	4	5	Remarks
				175,184 X	JB
			132,168 4-65 198,112		GS
			X	198,134	JB
		X			GS
	4--81 81,222	81,222 2--99 180,149 81,222 3-41 122,149			JB
			122,149 2-55 177,112		GS
				177,165 X	JB
			122,160 5-63 185,112		GS
				185,134 2-134 319=TR X	JB
			X		
		81,222 5-79 160,149 X			JB
	5-27 27,222				GS
		27,222 2--67 94,149			GS

Locations

1	2	3	4	5	Remarks
2	X 1-63 63,222	27,222 3--97 124,149	94,159 3-30 144,112	144,184	GS JB
			94,168 4-84 178,112		GS
				178,112 3-112 290=TR X	JB
			X		
			124,149 2--43 167,112		GS
				167,184 X	GS JB
		27,222 4-109 136,149	124,168 X		JB
					GS
			136,149 2--64 200,112		JB
			136,159 X		JB
			X		
			63,222 3-45 108,149		GS
	108,168 4-65 173,112		GS GS		

Locations

1	2		4	5	Remarks
				173,165 X	JB
			108,160 5-52 160,112		GS
				160,184 X	JB
			X		
		63,222 4--112 175,149 63,222 5-44 107,149			JB
			107,159 3-46 153.112		GS
				153,184 X	GS JB
			107,168 4-83 190,112 X		JB
		X			
	3-9 9,222				GS
		9,222 1-75 84.149			
			84,168 4-65 149,112		GS JB
				149,165 X	
			84,160 5-52 136,112		GS JB
			X	136,184 X	
		9,222 4-56 65,149			GS

Locations

1	2	3	4	5	Remarks
			65,160 1-70 135,112	135,165 X	GS JB
			65,160 5-67 132,112	132,165 X	GS JB
			X		
		9,222 5--92 101,149			GS
			101,160 1-52 153,112	153,184 X	GS JB
			101,168 4-62 163,112	163,165 X	GS JB
			X		
	4--72 72,222 5--36 36,222	X			JB
					GS
		36,222 1-59 95,149			GS
			95,151 3-46 141,112	141,184 X	GS JB

Locations

1	2	3	4	5	Remarks
3	X	36,222 3--77 113,149	95,168 4-83 178,112 X		JB
			113,160 1-52 165,112		GS
				165,184 X	GS JB
			113,168 4--42 175,112		GS JB
			X	175,165 X	
			36,222 4-104 140,140		GS JB JB
			140,160 140,159 X		
		X			
	1--45 45,222	45,222 2-61 106,149	106,168 4-59 165,112	165,165 X	GS GS JB

Locations

1	2	3	4	5	Remarks
			106,160 5-57 163,112	163,184 X	GS JB
		45,222 4-82 127,149	X 127,149 2-49 176,112		GS
			127,160 5-69 196,112 X	176,165 X	GS JB
		45,222 5-69 114,149	114,149 2-37 151,112		JB
			114,168 4-59 173,112	151.184 X	GS JB
		X	X	173,134 X	GS JB
	2-9 9,222				GS

Locations

1		3	4	5	Remarks
		9,222 1-81 90,149	90,168 4--59 149,112	149,165 X	GS JB
			90,160 5--57 147,112	147,184 X	GS JB
			X		
		9,222 4--74 83,149			GS
			83,160 1--68 151,112		GS JB
				151,165 X	
			83,160 5--72 155,112		GS JB
				155,165 X	
			X		
		9,222 5-77 86,149			GS

Locations

1	2	3	4	5	Remarks
			86,160 1--50 136,112	136,184 X	GS JB
			86,168 4--56 142,112	142,165 X	GS JB
		X	X		
	4--18 18,222				GS
		18,222 1-97 115,149			GS
			115,149 2-49 164,112	164,165 X	GS JB
			115,160 5-69 184,112 X		JB
		18,222 2-69 87,149			GS
			87,160 1-68 155,112	155,165 X	GS JB
			87,160 5-72 159,112		GS

Locations					
1	2	3	4	5	Remarks
4				159,165 X	JB
		18,222 5-109 127,149	X		GS
			127,160 1--56 183,112		JB
			127,149 2--40 167,112		GS JB
		X	X	167,165 X	
	5-81 81,222 X				JB
	1-81 81,222				JB
	2--72 72,222				JB
	3--18 18,222				GS
		18,222 1-85 103,149			GS
			103,149 2-46 159,112		GS JB
			103,168 5-68 171,112	159,165 X	GS JB
			X	171,134 X	GS JB

Locations

1	2	3	4	5	Remarks
5		18,222 2--48 66,149	66,160 1--72 138,112	138,165 X	GS GS JB
			66,160 5--71 137,112	137,165 X	GS JB
			X		
		18.222 5-112 130,149	130,160		GS GS JB
			130,149 2-47 177,112	X	GS JB
			X	177,165 X	
		X			
	5-72 72,222 X				JB
	1-27 27,222				GS
		27,222 2-76 103,149			GS

Locations

1	2	3	4	5	Remarks
			103,159 3-54 157,112		GS
			103,168 4-83 186,112 X	157,184 X	JB
		27,222 3-85 112,149	112,149 2-40 152,112		JB
			112,168 4--65 177,112		GS
			X	152,184	GS JB
		27,222 4-112 139,149	139,149 2-61 200,112 139,159 X	177,134 X	GS JB
		X			GS
	2-36 36,222	36,222 1-107 107,149	107,159 3--54 161,112		GS
					GS

Locations

1	2	3	4	5	Remarks
				161,184 X	JB
			107,168 4-83 190,112 X		JB
		36,222 3-53 89,149			GS
			89,160 1--48 137,112		GS
				137,184 X	JB
			89,168 4-62 151,112		GS
				157,165 X	JB
			X		
		36,222 4--113 149,149			GS
			149,160 149,159 X		JB JB
		X			
	3-81 81,222 4-72 72,222 X				JB
					JB
					STOP

6.7 Computational Results

To test the performance of the proposed algorithm LEXIQAP, problem data sets for each of sizes 8,9 and 10 are generated randomly. Both the algorithms i.e. Das algorithm as well as LEXIQAP are coded in FORTRAN. The programmes are tested on these data sets using Wipro super Genius E-590 and the execution time(in seconds) are reported in the following tables

Table No 6.7.1 Problem Size 8

P.No	Das Alg t_1	Lexiqap t_2	t_1-t_2	t_1/t_2	Optimum Value
1	1.48	1.15	0.33	1.29	1885
2	1.15	1.04	0.11	1.11	1279
3	1.64	1.42	0.22	1.16	2086
4	1.48	1.04	0.44	1.42	2118
5	1.59	1.20	0.39	1.33	2278
6	1.86	1.31	0.55	1.42	1976
7	1.42	1.04	0.38	1.37	1725
8	1.48	0.87	0.61	1.70	1406
9	2.41	1.09	1.32	2.22	2030
10	1.94	1.20	0.74	1.62	2189
Mean	1.645	1.136	0.509	1.463	
S.D	0.331	0.148	0.321	0.303	

The 95% confidence limits for the mean time taken by Das algorithm is (1.382,1.908) and the same for the proposed lexiqap algorithm is (1.018,1.254)

Table 6.7.2 Problem Size 9

P.No	Das Alg t_1	Lexiqap t_2	t_1-t_2	t_1/t_2	Optimum Value
1	13.29	4.88	8.41	2.72	2916
2	11.58	4.11	7.47	2.82	2204
3	12.35	5.16	7.19	2.39	2034
4	12.19	5.93	6.26	2.06	2528
5	09.00	4.94	4.06	1.82	1915
6	08.45	5.16	3.29	1.64	1525
7	13.01	7.30	5.71	1.78	2664
8	05.38	3.13	2.25	1.72	1864
9	08.89	4.11	4.78	2.16	2086
10	09.72	5.60	4.12	1.74	2227
Mean	10.386	5.032	5.354	2.085	
S.D	2.393	1.080	1.886	0.407	

The 95% confidence limits for the mean time taken by Das algorithm is (8.484,12.288) and the same for the proposed lexiqap algorithm is (4.174,5.59).

Table 6.7.3 Problem Size = 10

P.No	Das Alg t_1	Lexiqap t_2	t_1-t_2	t_1/t_2	Optimum Value
1	115.89	41.24	74.65	2.81	2226
2	120.45	55.30	65.15	2.18	2304
3	110.45	57.94	52.51	1.91	3707
4	149.89	52.78	97.11	2.84	3148
5	151.42	58.82	92.60	2.57	3646
6	133.02	34.32	98.70	3.88	2550
7	093.92	24.82	69.10	3.78	3107
8	149.72	55.69	94.03	2.69	3326
9	117.26	48.00	69.26	2.44	3035
10	129.34	61.51	67.83	2.10	2777
Mean	127.136	49.042	78.094	2.72	
S.D	18.186	11.371	15.326	0.627	

The 95% confidence limits for the mean time taken by Das algorithm is (112.682,141.59) and the same for the proposed lexiqap algorithm is (40.005,58.079).

Since in all the cases the 95% confidence range for the time taken by the present algorithm is non overlapping with entirely left of the confidence range of times required by the Das algorithm. It is very clear that the present algorithm is time wise superior to the Das algorithm. In fact a look at the values suggest that computational time is reduced by half.

6.8 SOME COMMENTS

Some interesting questions that arose in the context of lexiseach algorithm to some Combinatorial Programming Problems developed in this thesis, but could not be elaborated / answered are listed below.

It is observed that during the process of obtaining the optimum solution, the confirmation of a solution to be 'optimal' takes a lion's share in the total computation time. Heuristics gives most of the times "near optimal" (if not optimal) solutions quickly. The question of estimating the minimum value in a population by a sequential sampling strategy should be of some relevance in this context. This situation is analogous to the problem of reliability improvement as a 'learning process' in developing fault free systems by means of learning through failures in prototypes (c.f. **Srivastava,1976**). Investigations reported in **Sambhasiva Rao (c.f.Sambasiva Rao 1990)** appear to be an attempt in this direction.

In the context of k- salesmen problems, another interesting objective function is equalising as much as possible the efforts of salesmen i.e., minimising the difference between maximum subtour length and minimum subtour length. Though this appears to an interesting objective function , this case could not be investigated, even for the case 2-salesmen problem

In the Maximum Weighted sub-graph, we could not generate the 'problem-data sets' using the Random data that can lead to a large number of positive super nodes. The little computational experience reported in the chapter-5 could be biased as they are 'cooked-up' examples. Since the problem is interesting and has practical applications, it is worthwhile to develop a "problem-data-sets generation procedure" so that the algorithm described can thoroughly be tested.

Bound setting being, essentially not linear in the strict sense, one is tempted to inquire whether a 'shift of origin or change of scale' can have an effect on the time taken for solution (apart from such 'trivia' as: the time for search increases with the size of the problem). Hence, a small investigation was conducted to assess the effect of change of scale on the performance of the lexisearch algorithm in the context of the conventional travelling salesman's problem . A set of ten problems (of size 17) generated in the first instance. The times taken for solution by our algorithm were noted. The

'distances' were multiplied by a scale factor a : $d^*(i,j) = d(i,j) \times a$, for values of $\alpha=1,10,50,75,&100$ and the times for solution of the same problems but with scaling up by a factor a were also noted. The table 6.8.1 gives the times for solution for different values of a . It is obvious that, on the whole, for the first three cases $\alpha=1,10$ and 50, timings are essentially the same, after crossing 50, the time taken seems to be clearly larger than in the case of $\alpha=1$. Only in the case of problems 1 and 8 this appears to be not true.

TABLE 6.8.1

P.No	$\alpha=1$	$a=10$	$a=50$	$a=75$	$\alpha=100$
1.	3.51	3.51	2.19	3.13	3.62
2.	3.95	3.95	4.11	4.22	4.28
3.	1.75	1.75	1.75	2.63	3.18
4.	1.26	1.26	1.31	2.52	3.46
5.	1.26	1.31	1.15	2.19	2.85
6.	2.69	2.74	2.74	4.00	4.55
7.	1.53	1.48	2.19	4.00	4.88
8.	5.21	5.21	3.73	5.16	5.71
9.	3.89	3.89	3.02	4.50	5.16
10.	1.64	1.64	1.86	2.58	3.13

The table 6.8.2 gives the ranking of the problems according to their execution time for solution.

TABLE 6.8.2

P.No	$\alpha=1$	a=10	$\alpha=50$	$\alpha=75$	$\alpha=100$
1.	7	7	5.5	5	5
2.	9	9	10	8	6
3.	5	5	3	4	3
4.	1.5	1	2	2	4
5.	1.5	2	1	1	1
6.	6	6	7	6.5	7
7.	3	3	5.5	6.5	8
8.	10	10	9	10	10
9.	8	8	8	9	9
10.	4	4	4	3	2

TABLE 6.8.3

ANOVA TABLE

S.V.	d.f	Sum of Squares	Mean Square
Between Problems	4	362.1	90.52
Error	45	48.9	1.086
Total	49	411	

By concordance analysis it is found that rank order of the problem with respect to time for solution is effectively invariant with respect to change of scale. This conclusion follows from the ANOVA table (i.e. table 6.8.3) of the ranking of the timings. However, it does not follow either that the actual time for solving problems can be invariant or that it is systematically changing with respect to the scale. The actual figures given in table 6.8.1 of the time for solution, shows some evidence that increase in the scale is associated with increase in the time for solution.

APPENDICES

APPENDIX - 1

Connectivity Matrix:

Let $G(A,V,E)$ be the node weighted connected graph with node set $A=\{1,2,..n\}$ and E the edge set. Let V be the set of values associated with the corresponding nodes. The connectivity matrix of this graph is denoted by C and is defined as

$$c_{ij} = 0 \text{ if node } i \text{ is connected to node } j \text{ i.e. } (i,j) \in E \\ = 1 \text{ Otherwise.}$$

MIN ADDITION:

Let A and B two be two conformable matrices. The Minadd operation is defined as follows:

$$C = A \oplus B \quad \text{where } c_{ij} = \min \{ a_{ix} + b_{xj} \}.$$

Let $A [a_{ij}]$ be the matrix with entries as the distance of j from i . Then $A^2 = A \odot A$ gives the shortest distance from i to j in exactly 2 steps and similarly for $A^k = A \oplus A \odot \dots \odot A$ (k times) will give shortest route from i to j in exactly k steps(including possibly steps leading to cycles). However, when $a_{ij} > 0$ and $a_{ii} = 0$ for $ij=1,2,..,n$ the term $a_{ij}^{(k)}$,

the (ij) entry of A^k , obviously gives the shortest distance from i to j in k steps or less. Obviously $A^r = A^{r-1}$ for $r > n$. In fact, A^r may become equal to A^s for some $r > s > n$, it is called as satiated matrix which gives the shortest possible distance between nodes i and j without any restriction on the number of steps involved.

In the context of connectivity, if one defines $a_{ij} = 0$ when i is directly connected to j and 1 otherwise, the satiated matrix will have 0 for a_{ij} if node j is directly or indirectly connected to node i ; $a_{ij} = 1$ if i and j are not connected, either directly or through other, intermediate, nodes.

To get a satiated matrix, one has to raise the connectivity matrix of a graph of size n not more than $\log_2 n$ times but even this amount of computation can be reduced by using compact storage algorithm (Pandit 1961).

It consists of a forward and backward pass involving in each case a row-column **min** addition for each element. Let a_{ij} be the entry for (i, j) in the first pass. It is defined as follows;

$a_{ij}^{(1)} = \min \sum (a_{ix} + a_{xj})$ where a_{ix} and a_{xj} represent the current values of the matrix $A(1)$ $i, j = 1, 2, 3, \dots, n$.

For the second pass the $a_{ij}^{(2)}$ is defined as follows;

$a_{ij}^{(2)} = \min \sum (a_{ix}^* + a_{xj}^*)$ where a_{ix}^* and a_{xj}^* represent the current values of the matrix $A(2)$ $i, j = n, n-1, \dots, 2, 1$.

The compact storage algorithm will be illustrated through an example.

Consider a network consists of 5 nodes with the following connectivity matrix A.

$$\begin{bmatrix} 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \end{bmatrix}; \text{ First pass matrix } A(1) = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \end{bmatrix}$$

In the first pass, the $(1,3)^{\text{th}}$ element becomes zero

$$\begin{aligned} \text{ie } a(1,3) &= \text{Min}\{(a(1,3)+a(1,1)), (a(1,2)+a(2,3)) (a(1,3)+a(3,3)) \\ &\quad (a(1,4)+a(4,3)) (a(1,5)+a(5,3))\} \\ &= \text{Min}\{(0+1), (0+0), (1+0), (1+0), (1+0)\} \\ &= 0 \end{aligned}$$

Similarly, the $(3,1)^{\text{th}}$ also becomes zero.

$$\text{Second pass matrix } A(2) = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \end{bmatrix}$$

It is so happen that in this problem there is no change from A(1) to A(2). However, this need not be always the case.

APPENDIX 2

```
C   PROGRAMME FOR K- TRAVELLING SALESMAN
C   M IS THE NUMBER OF ELEMENTS USED IN BOUND CALCULATIONS
      IMPLICIT INTEGER (A-Z)
      DIMENSION A(50,50),B(50,50),IND(50),IP(50),RSTART(100)
      DIMENSION OPT(100),X(50),C(50),W(50),START(50),TS(50),E(20,10)
      DIMENSION B1(50,50),SA(1000),SB(1000),EB(20,20)
      INTEGER *4 A,X,TR
      CHARACTER*18 CTIME
      LOGICAL SETTIM
      DATA CTIME/'THE TIME TAKEN IS'/
      IF(.NOT.(SETTIM(0,0,0,0)))WRITE(*,*)'SETTIM FAILED'1
      READ(V)N,NT
      WRITE(*,*)'NO OF CITIES =',N
      WRITE(*,*)'NO OF SALESMAN =',NT
      READ(V)(TS(IL),IL=1,NT)
      WRITE(*,*)'RESTRICTION ON EACH SALESMAN'
      WRITE(*,*)(TS(IL),IL=1,NT)
      READ(*,*)((A(I,J),J=1,N),I=1,N)
      DO 51 I = 1, N
      DO 52 J = 1,N
      X(J)=A(I,J)
52   CONTINUE
      CALL ASCEND(N,X,IP)
      DO 53 JK = 1,N
      B1(I,JK)=IP(JK)
53   CONTINUE
51   CONTINUE
      DO 751 I = 1,N
      LOUNT = 0
      DO 752 J =1,N
      DO 753 K = 1,NT
      IF (B1(I,J).EQ.K) GO TO 752
753  CONTINUE
      LOUNT = LOUNT + 1
      B(I,LOUNT)=B1(I,J)
752  CONTINUE
751  CONTINUE
      N1 = N-NT
      KA = 0
      DO 155 IL = 1,N
      DO 155 IJ = 1,N
      KA = KA+1
```

```

      SA(KA)= A(IL,IJ)
155  CONTINUE
      KB = 0
      DO 156 IL = 1,N
      DO 156 IJ = 1, N-NT
      KB = KB + 1
      SB(KB) = B(IL,IJ)
156  CONTINUE
      DO 57 I=1,N
      X(I)=A(I,1)
57   CONTINUE
      CALL ASCEND(N,X,IP)
      DO 59 I9=1,N
      C(I9)=IP(I9)
59   CONTINUE
C    LEXICOSEARCH STARTS
      START(1)=1
      IND(1)=0
      DO 48 I4=2,NT
      START(I4)=1
      RSTART(I4)=(I4-1)*(N-NT)
48   IND(I4)=0
      DO 49 I5=NT+1,N
      RSTART(I5)=(I5-1)*(N-NT)
49   IND(I5)=0
C    COMPUTATION OF BOUND MATRIX
      DO 123 I=1,N
      DO 124 J=1, M
      E(I,J) = B1(I,J)
      JL = B1(I,J)
      EB(I,J) = A(I,B1(I,J))
124  CONTINUE
123  CONTINUE
      DO 259 IY=1,N
259  CONTINUE
      TR=99999
      COUNT=1
      W(COUNT)=1
      IND(1)=100
30   DO 1 JJ=1,N-NT
      SUM=0
      KING=SB(JJ)
      IND(KING)=1
      SUM=SUM+SA(KING)
      IF(SUM.GE.TR) GO TO 100

```



```

COUNT=COUNT+1
W(COUNT)=KING
101 FORMAT(15I6)
KSTART=0
START(1)=2
I=1
5 IT=I
20 DO 6 J=START(I),TS(I)
ALPHA=J
920 UT=RSTART(KING)+1
UT1=(KING-1)*(N-NT)
UT11=UT1 +5
DO 7 K=UT,UT11
QUEEN=SB(K)
IF(IND(QUEEN).EQ.0) GO TO 8
7 CONTINUE
GO TO 147
8 IND(QUEEN)=I
KPT=(KING-1)*N+QUEEN
SUM=SUM+SA((KING-1)*N+QUEEN)
IF(SUM.GE.TR) GOTO 45
BOUND1=0
DO 800 JX=1,N
IND(1)=0
JXX=E(QUEEN,JX)
IF(IND(JXX).NE.0) GO TO 800
BOUND1=BOUND1+A(QUEEN,JXX)
GO TO 810
800 CONTINUE
BOUND 1=BOUND 1+EB(QUEEN,M)
810 DO 80 IX= 2,N
IF (IND(IX) .NE. 0) GO TO 80
C NO OF ELEMENTS IN BOUND MATRIX IS M
DO 151 JX=1,M-1
JK=E(IX,JX)
IF (IND(JK) .NE. 0) GO TO 151
BOUND 1=BOUND1+A(IX,JK)
GO TO 80
151 CONTINUE
BOUND1=BOUND1 + EB(IX,M)
80 CONTINUE
IND(1)=100
IF((SUM+BOUND 1 ).GE.TR) GO TO 40
IND(QUEEN) = I
START(I)=J

```

```

RSTART(KING)=K
COUNT=COUNT+1
W(COUNT)=QUEEN
KING=QUEEN
6  CONTINUE
   KT=I+1
   SUM=SUM+SA(KING)
   IF(SUM.GE.TR) GOTO 41
   IF((I+1).GT.NT) GOTO 81
   COUNT=COUNT+1
   W(COUNT)=I+1
   IND(I+1)=I+1
   KSTART=I+1
   KING=I+1
   I=I+1
   GOTO 5
81  TR=SUM
   COUNT=COUNT+1
   W(COUNT)=1
   DO 99 KP=1,COUNT
99  OPT(KP)=W(KP)
   VAL=TR
   GOTO 46
C  *****:
C      SAME BLOCK
C  *****:
40  SUM=SUM-SA((KING-1)*N+QUEEN)
   IT=IND(QUEEN)
   IND(QUEEN)=0
   START(IT)=ALPHA
   I=IT
   RSTART(KING)=K
   KING=W(COUNT)
   GOTO 920
C  *****:
C      SUPER BLOCK
C  *****:
45  IF(ALPHA.GT.1) GOTO 71
   SUM=SUM-SA((KING-1)*N+QUEEN)
   IT=IND(QUEEN)
   START(IT)=1
   IND(QUEEN)=0
   KING=W(COUNT-1)
   QUEEN=W(COUNT)
   SUM=SUM-SA((KING-1)*N+QUEEN)

```

```

IND(QUEEN)=0
RSTART(QUEEN)=(QUEEN-1)*(N-NT)
COUNT=COUNT-1
KING=W(COUNT-1)
QUEEN=W(COUNT)
SUM=SUM-SA((KING-1)*N+QUEEN)
IT=IND(QUEEN)
RSTART(QUEEN)=(QUEEN-1)*(N-NT)
IND(QUEEN)=0
COUNT=COUNT-1
KING=W(COUNT)
START(IT)=TS(IT)
I=IT
GOTO 20
71 SUM=SUM-S A((KING-1 )*N+QUEEN)
IT=IND(QUEEN)
IND(QUEEN)=0
RSTART(QUEEN)=(QUEEN-1)*(N-NT)
START(IT)=ALPHA-1
IF((COUNT-1).EQ.1) GOTO 1
KING=W(COUNT-1)
QUEEN=W(COUNT)
IT=IND(QUEEN)
IND(QUEEN)=0
RSTART(QUEEN)=(QUEEN-1)*(N-NT)
SUM=SUM-SA((KING-1)*N+QUEEN)
COUNT=COUNT-1
KING=W(COUNT)
UT=RSTART(KING)
I=IT
GOTO 20
46 KING=W(COUNT-1)
QUEEN=W(COUNT)
SUM=SUM-SA((KING-1)*N+QUEEN)
COUNT=COUNT-1
149 KING=W(COUNT-1)
QUEEN=W(COUNT)
SUM=SUM-SA((KING-1)*N+QUEEN)
IND(QUEEN)=0
RSTART(QUEEN)=(QUEEN-1)*(N-NT)
COUNT=COUNT-1
KING=W(COUNT-1)
QUEEN=W(COUNT)
SUM=SUM-SA((KING-1)*N+QUEEN)
IT=IND(KING)

```

```

IND(QUEEN)=0
RSTART(QUEEN)=(QUEEN-1)*(N-NT)

START(IT)=START(IT)-1
COUNT=COUNT-1
KING = W(COUNT)
I=IT
GOTO 20
47 IT=IND(W(COUNT))
   KING=W(COUNT-1)
   QUEEN=W(COUNT)
   SUM=SUM-SA((KING-1)*N+QUEEN)
   RSTART(QUEEN)=(QUEEN-1)*(N-NT)
   IND(KING)=0
   COUNT=COUNT-1
   START(IT)=START(IT)-1
   IF(COUNT-1.EQ.0) GOTO 1
   I=IT
   GOTO 20
41 IF(KT.LE.2) GO TO 410
   SUM=SUM-SA(KING)
   KING=W(COUNT-1)
   QUEEN=W(COUNT)
   SUM=SUM-SA((KING-1)*N+QUEEN)
   IND(QUEEN)=0
   RSTART(QUEEN)=(QUEEN-1)*(N-NT)
   COUNT=COUNT-1
   KING=W(COUNT-1)
   QUEEN=W(COUNT)
   SUM=SUM-SA((KING-1)*N+QUEEN)
   IND(QUEEN)=0
   RSTART(QUEEN)=(QUEEN-1)*(N-NT)
   COUNT=COUNT-1
   KING=W(COUNT)
   IT=IND(KING)
   START(IT)=ALPHA-1
   I=IT
   GOTO 20
410 SUM=SUM-S A(KING)
     KING=W(COUNT-1)
     QUEEN=W(COUNT)
     SUM=SUM-SA((KING-1)*N+QUEEN)
     IND(QUEEN)=0
     RSTART(QUEEN)=(QUEEN-1)*(N-NT)
     COUNT=COUNT-1

```

```

    KING=W(COUNT)
    IT = IND(KING)
    START(IT) = TS(IT)
    I=IT
    GO TO 20
147 IF(ALPHA.GT. 1) GOTO 148
    KING=W(COUNT-1)
    QUEEN=W(COUNT)
    SUM=SUM-SA((KING-1)*N+QUEEN)
    IT=IND(QUEEN)
    START(IT)=1
    IND(QUEEN)=0
    RSTART(QUEEN)=(QUEEN-1)*(N-NT)
    IF((COUNT-1).EQ.1) GOTO 1
    COUNT=COUNT-1
    KING=W(COUNT-1)
    QUEEN=W(COUNT)
    SUM=SUM-SA((KING-1)*N+QUEEN)
    IT=IND(QUEEN)
    IND(QUEEN)=0
    RSTART(QUEEN)=(QUEEN-1)*(N-NT)
    COUNT=COUNT-1
    KING=W(COUNT)
    START(IT) = TS(IT)
    I=IT
    GOTO 20
148 KING=W(COUNT-1)
    QUEEN=W(COUNT)
    SUM=SUM-SA((KING-1)*N+QUEEN)
    IT=IND(QUEEN)
    IND(QUEEN)=0
    RSTART(QUEEN)=(QUEEN-1)*(N-NT)
    COUNT=COUNT-1
    KING= W(COUNT)
    IF(COUNT.EQ.1) GOTO 1
    I=IT
    START(IT)=ALPHA-1
    I=IT
    GOTO 20
1    CONTINUE
100 WRITE( *,*)(OPT(KP),KP=1 ,N+1)
    WRITE(*,*)'OPTIMUM VALUE=',VAL
    CALL GETTIM(IHR,IMIN,ISEC,I100TH)
    WRITE(*,*)CTIME,IHR,IMIN,ISEC,I100TH
    STOP

```

```
      END
C    SUB ROUTINE FOR ASCEND ORDERING
      SUBROUTINE ASCEND(N,X,IP)
      DIMENSION IP( 100)
      INTEGER*4 X( 100)
      DO 26 I=1,N
      IP(I)=1
26    CONTINUE
      DO 27 I=1,N-1
      DO 28 J=I+1,N
      IF(X(I).LE.X(J)) GOTO 28
      LEMP=X(I)
      X(I)=X(J)
      X(J)=LEMP
      KEMP=IP(I)
      IP(I)=IP(J)
      IP(J)=KEMP
28    CONTINUE
27    CONTINUE
      RETURN
      END
```

```

C      PROGRAMME FOR MINIMISING THE MAXIMUM STEP IN A TOUR
      IMPLICIT INTEGER (A-Z)
      PARAMETER(N1= 100)
      INTEGER A(N1 ,N1),L(N1 ),TW(N1 ),IND(N1 ,N1 ),IROW(N1 ),B(N1 ,N1 )
      INTEGER TW1(N1)
      CHARACTER* 18 CTIME
      LOGICAL SETTIM
      DATA CTIME/'THE TIME TAKEN IS'/
      IF(.NOT.(SETTIM(0.0,0,0)))WRITE(*,*)'SETTIM FAILED'
      READ(*,*)N
      READ(*,*)((A(I,J),J=1,N),I=1,N)
      WRITE(*,*)'THE GIVEN MATRIX'1
      WRITE(*,5)((A(I,J),J=1,N),I=1,N)
      DO 4 I = 1,N
      IROW(I) = 0
      L(I) = 0
      DO 4 J = 1,N
      IND(I,J)=J
      B(I,J) = A(I,J)
4      CONTINUE
      MAX = A(N,1)
      DO 46 I= 1,N-1
      SUM1 = SUM1 + A(I,I+1)
      IF(MAX.GT.A(I,I+1)) GO TO 46
      MAX = A(I,I+1)
46     CONTINUE
      SUM1 = SUM1 + A(N,1)
      DO 1 I=1,N
      DO 2 J = 1,N-1
      DO 3 K= J+1,N
      IF(A(I,J).LE.A(I,K)) GO TO 3
      LEMP = A(IJ)
      A(I ,J)=A(I,K)
      A(I,K) = LEMP
      KEMP = IND(I,J)
      IND(I,J) = IND(LK)
      IND(LK) = KEMP
3      CONTINUE
2      CONTINUE
1      CONTINUE
      WRITE(V)'THE ARRANGED MATRIX'
      WRITE(*,5)((A(I,J),J=1,N),I=1,N)
      WRITE(*,*)'THE INDEX MATRIX'
      WRITE(*,5)((IND(I,J),J=1,N),I=1,N)
      WRITE(*,5)(IROW(I),I=1,N)

```

```

5  FORMAT(10(2X,I5))
   LOUNT = 0
C   ***** LEXI SEARCH BEGINS *****
   DO 6 J = 1,N-1
     SUM = 0
     COUNT = 1
     TW(COUNT) = 1
     IROW(1) = 1
     IF(A(1,J).GT.MAX) GO TO 100
     K = IND(1,J)
C   ***** BOUND CALCULATIONS *****
     IROW(1) = 0
     DO 7 JB = 1,N-1
       KB = IND(K,JB)
       IF(IROW(KB).NE.0) GO TO 7
       IF(A(K,JB).GE.MAX) GO TO 6
       BOUND = A(K,JB)
       GO TO 8
7    CONTINUE
8    IROW(K) = 100
     DO 9 IB = 2,N
       IF(IROW(IB).NE.0) GO TO 9
       DO 10 JB = 1, N-1
         KB = IND( IB,KB)
         IF(IROW(KB).EQ.0) GO TO 11
10   CONTINUE
     GO TO 9
11   LEMP = A(IB,JB)
     IF( BOUND.GE. LEMP) GO TO 9
     BOUND = LEMP
9    CONTINUE
     IROW(1) = 100
     IF(BOUND.GE.MAX) GO TO 6
     COUNT = COUNT + 1
     SUM = SUM + A(1,J)
     TW(COUNT) = K
     IROW(K) = 100
20   DO 12 J1 = L(K)+ 1,N-1
     K1 = IND(K,J1)
     IF(IROW(K1).EQ.0) GO TO 13
12   CONTINUE
     GO TO 30
13   IF(A(K,J1).GE.MAX) GO TO 30
C   ***** BOUND CALCULATIONS *****
     IROW(1) = 0

```



```

DO 14 JB = 1, N-1
KB = IND(K,JB)
IF(IROW(KB).EQ.0) GO TO 15
14  CONTINUE
15  BOUND = A(K1,JB)
    IF(BOUND.GE.MAX) GO TO 16
    IROW(K1) = 100
    DO 17 IB = 2,N
    IF(IROW(IB).NE.0) GO TO 17
    DO 18 JB = 1,N-1
    KB = IND(IB,JB)
    IF(IROW(KB).EQ.0) GO TO 19
18  CONTINUE
19  LEMP = A(K1,JB)
    IF( BOUND .GE .LEMP ) GO TO 17
    BOUND = LEMP
    IF( BOUND .GE .MAX ) GO TO 36
17  CONTINUE
    IROW(1) = 100
    SUM = SUM+B(K.K1)
    COUNT = COUNT +1
    TW ( COUNT ) = K1
    IROW(K1) = 100
    L(K) = J1
    K = K1
    IF(COUNT .EQ.N) GO TO 31
    GO TO 20
16  IROW(1) =100
    L(K) = J1
    GO TO 20
36  IROW(K1) = 0
    IROW(1) = 100
    L(K) = J1
    GO TO 20
C   *****BEGINING OF THE SUPER BLOCK*****
30  IF(COUNT.EQ.2) GO TO 21
    IROW(TW(COUNT))= 0
    L(TW(COUNT))= 0
    SUM = SUM - B(TW(COUNT-1),TW(COUNT))
    COUNT = COUNT - 1
    K = TW(COUNT)
    GO TO 20
21  IROW(TW(2)) = 0
    L(TW(2)) = 0
    SUM = 0

```

```

      GO TO 6
C      *****END OF SUPER BLOCK *****

C      *****BEGINING OF THE SOLUTION BLOCK *****
31  LEMP = B(TW(COUNT), 1)
      IF(LEMP.GE.MAX) GO TO 22
      MAX1 =LEMP
      IOPT = LEMP
      DO 23 IS = 1,N-1
      IOPT = IOPT + B(TW(IS),TW(IS+1))
      IF(MAX1.GE.B(TW(IS),TW(IS+1))) GO TO 23
      MAX1 = B(TW(1S),TW(IS+1))
23  CONTINUE
      IF(MAX.LE.MAX 1) GO TO 22
      MAX = MAX1
      OPTSOL = IOPT
      OPTMAX=MAX
      DO 200 IO = 1,COUNT
      TW1(IO) = TW(IO)
200  CONTINUE
22  IROW(TW(COUNT)) = 0
      L(TW(COUNT))= 0
      SUM = SUM - B(TW(COUNT-1),TW(COUNT))
      COUNT =COUNT-1
      IROW(TW(COUNT))= 0
      L(TW(COUNT))= 0
      SUM = SUM - B(TW(COUNT-1),TW(COUNT))
      COUNT =COUNT-1
      K = TW(COUNT)
      GO TO 20
6    CONTINUE
111  FORMAT(15(1X,I4))
100WRITE(*,111)(TW 1 (IO),IO=1 ,N),OPTSOL,OPTMAX
      CALL GETTIM(IHR,IMIN,ISEC,1100TH)
      WRITE(*,*)CTIME,IHR,IMIN,ISEC,1100TH
      STOP
      END

```

```

C   PROGRAMME TO FIND OUT MAXIMUM NODE WEIGHTED
C   CONNECTED SUB GRAPH FROM THE GIVEN GRAPH
C   IN THIS 0 STANDS FOR DIRECT CONNECTION BETWEEN ANY
C   TWO BLOCKS
C   IN THIS 1 STANDS FOR NO DIRECT CONNECTION BETWEEN
C   ANY TWO BLOCKS
C   N IS SIZE OF THE GRAPH
C   ARRAY X CONTAINS WIGHT OF THE NODES
C   A IS CONNECTIVITY MATRIX, P IS THE NUMBER OF SUPER
    NODES
    IMPLICIT INTEGER (A-Z)
    PARAMETER(NN=100)
    DIMENSION A(NN,NN),A1(NN,NN),B(NN,NN),C(NN,NN)
    DIMENSION D(NN,NN),E(NN,NN),NS(NN,NN)
    DIMENSION FN(NN,NN),FP(NN,NN)
    DIMENSION KM(NN),LA(NN),IS(NN),T(NN),VAL(NN),VP(NN)
    DIMENSION VN(NN),TP(NN),TN(NN),IN(NN),W(NN)
    DIMENSION PNN(NN),IP(NN),OPT(NN),ND(NN),X(NN)
    LOGICAL SETTIM
    DATA CTIME/'THE TIME TAKEN IS7
    IF(.NOT.(SETTIM(0,0,0,0)))WRITE(*,*)'SETTIMFAILED'1
    READ(*,*)N
    READ(V)(X(I),I=1,N)
    READ(V)((A(I,J),J=1,N),I= 1, N)
    DO 140 I = 1,N
    IF(X(I).GT.0) GO TO 150
140  CONTINUE
150  M1 = I
    N1 =(N+1)-M1
    DO 160 K = 1, N1
    K1=K-1
    DO 160 LL = 1,N1
    L1 = LL-1
    B(K,LL)=A(M1+K1,M1+L1)
160  CONTINUE
    CALL LOMPT(NN,B,N1,C,Q)
    NAL = 1
    DO 190 I = 1,N1
190  ND(I) = 0
    I = 1
    J = 1
    ND(J) = I
194  K1 = 1
    NS(I,K1) = J + M1 - 1

```

```

ND(J) = I
NA = NAL+1
DO 191 J = NA,N1
IF(NA.GT.N1)GO TO 1100
IF(C(NAL,J).EQ.1) GOTO 191
K1 = K1+1
NS(I,K1) = J + M1 -1
ND(J) = I
191  CONTINUE
    KM(I) = K1
    DO 192 J = 1,N1
    IF(ND(J).EQ.0) GO TO 193
192  CONTINUE
    GOTO 1100
193  NAL = J
    I = I + 1
    ND(J) = I
    GO TO 194
1100 P = I
    DO 161 I = 1,P
    IS(I) = 0
    L1 = KM(I)
    DO 161 J = 1, L1
    L2 = NS(I,J)
    ISO) = IS(I) + X(L2)
161  CONTINUE
    CALL LARANG(NN,IS,P,T,LA)
    DO 175 I = 1, P
    I1 =LA(I)
    MR = KM(I1)
    DO 176 J = 1,MR
    D(I,J) = NS(I1,J)
176  CONTINUE
175  CONTINUE
174  FORMAT(5X, 10(2X,I3))
    DO 151 I = 1 ,P
    DO 151 J = 1 ,P
    IF(I.EQ.J) GO TO 152
    E(I,J) = 1
    E(J,I) = 1
    GO TO 151
152  E(I,J) = 0
    E(J,I) = 0
151  CONTINUE
    DO 154 LN = 1 ,M1-1

```

```

      MJ = P + LN
      E(MJ,MJ) = 0
      DO 155 I = 1,P
      DO 156 J = 1 ,KM(LA(I))
      LJ = D(I,J)
      IF(A(LN,LJ).EQ.0) GO TO 157
156  CONTINUE
      E(I,MJ) = 1
      E(MJ,I) = 1
      GO TO 155
157  E(I,MJ) = 0
      E(MJ,I) = 0
155  CONTINUE
154  CONTINUE
      DO 1560 I = 1, M1-1
      DO 1570 J = 1 ,M1-1
      E(P+I,P+J) = A(I,J)
1570 CONTINUE
1560 CONTINUE
      N2 = P + M1 - 1
C      ***** N2 IS THE SIZE OF SUPER NODES MATRIX *****
      DO 199 I = 1, N2
      DO 199 J = 1, N2
      A1(I,J) = E(I,J)
199  CONTINUE
      IF(P.NE.1) GO TO 1999
      DO 7777 I = 2,M1-1
      IF(A1(I,1).NE.0) GO TO 7777
      OPTIMUM = IS(I)+X(I)
      GO TO 1001
7777 CONTINUE
      GO TO 1001
C      ..... END OF FIRST STAGE .....
C      K = NUMBER OF NEGATIVE NODES
C      L = NUMBER OF POSTIVE NODES
C      VAL(J) VALUE OF THE J TH NODE
1999 K = M1-1
      L = P
      N2 = K + L
      DO 215 I = 1, L
      VAL(I) = IS(I)
215  CONTINUE
      DO 216 J = L+1,N2
      VAL(J) = X(J-L)
216  CONTINUE

```

```

C      $$$$$$ END OF SECOND STAGE $$$$$$$$$$
C      FN(I,J) = SET OF NEGATIVE NODES CONNECTED TO I TH -VE
C      NODE
C      FP(I,J) = SET OF THE POSITIVE NODES CONNECTED RO I TH -VE
C      NODE
C      TP(I) = NUMBER OF POSITIVE NODES CİNNECTED TO I TH - VE
C      NODE
C      TN(I) = NUMBER OF NEGATIVE  NODES  CONNECTED TO I TH
C      -VE NODE
C      SN = 0
C      DO 3307 I = 1, K
C      I10 = I + L
C      VN(I) = VAL(I10)
C      SN = SN + VN(I)
3307  CONTINUE
C      SP = 0
C      DO 318 I = 1, L
C      VP(I) = IS(I)
C      SP = SP + VP(I)
318  CONTINUE
C      TPN = SN + SP
C      DO 320 I = 1 , K
C      I5 = I + L
C      CK = 0
C      CL = 0
C      TP(I) = 0
C      TN(I) = 0
C      DO 330 J = 1 , L
C      IF(A1(I5,J).NE.0) GO TO 330
C      CK = CK + 1
C      TP(I) = TP(I) + 1
C      FP(I,CK) = J + K
330  CONTINUE
C      DO 340 J = L+1 , N2
C      J5 = J - L
C      IN(J5) = 0
C      IF(A1(I5,J).NE.0) GO TO 340
C      IF(J.EQ.I5) GO TO 340
C      CL = CL + 1
C      TN(I) = TN(I) + 1
C      FN(I,CL) = J5
C      IN(J5) = 1
340  CONTINUE
C      DO 350 I1 = 1 , TP(I)
C      DO 355 J = L + 1, N2

```

```

I2 = FP(I,I1)
I8 = I2-K
IF(A1 (I8,J).NE.0) GO TO 355
IF(J.EQ.(I+L)) GO TO 355
IF(IN(J-L).NE.0) GO TO 355
IN(J-L) = 1
CL = CL + 1
FN(LCL) = J-L
TN(1)=TN(I) + 1
355  CONTINUE
350  CONTINUE
320  CONTINUE
C    ARRANGING THE FN(I,J) IN THE INCREASING ORDER
DO 3700 I= 1,K
LIMIT = TN(I)-1
3710  IFLAG = 0
DO 3720 J = 1 , LIMIT
IF(FN(I,J).LE.FN(I,J+1)) GO TO 3720
TEMP = FN(I,J)
FN(I,J)=FN(I,J+1)
FN(I,J+1) = TEMP
IFLAG=10
3720  CONTINUE
IF(IFLAG.GT.O) GO TO 3710
3700  CONTINUE
C    END OF SORTING OF THE NEGATIVE NODES

C          MAIN SEARCH BEGINS

C    PNN = IS POSITION OF THE -VE NODE
C    W(P) = IS THE LIST OF THE -VENODES IN THE WORD
TSS = 0
DO 3200 J = 1 , L
TSS = TSS + VP(J)
3200  CONTINUE
DO 3201 I = 1 ,L
IP(I) = 0
3201  CONTINUE
DO 3202 J = 1, K
IN(J) = 0
PNN(J) = 0
3202  CONTINUE
TR = VN(1)
DO 3203 J = 1,TP(1)
J1 = FP(1,J)

```

```

TR = TR + VP(J1-K)
3203 CONTINUE
C ***** LEXI SEARCH STARTS *****
DO 3001 I = 1,K
RV = TSS
P = 0
SUM = 0
IF((VN(I)+RV).LE.TR) GO TO 100
P = P+1
W(P) = I
SUM = SUM + VN(I)
IN(W(P)) = 500
DO 3401 J = 1, TP(W(P))
M = FP(W(P), J)
SUM = SUM + VP(M-K)
RV = RV - VP(M-K)
IP(M-K) = W(P)
3401 CONTINUE
IF(SUM.GE.TR) GO TO 302
IF((SUM+RV).LE.TR) GO TO 30011
GOTO 307
302 TR = SUM
OPTIMUM = TR
NUMPOSI = P
DO 31001 J = 1, P
OPT(J) = W(J)
31001 CONTINUE
307 DO 3432 J = PNN(W(P))+1, TN(W(P))
T2 = FN(W(P), J)
IF(IN(T2).EQ.0) GO TO 34321
3432 CONTINUE
GO TO 3007
C ***** ADDING NEW LETTER *****
34321 SUM = SUM + VN(T2)
IF((SUM+RV).LE.TR) GO TO 3007
IN(T2) = T2
PNN(W(P)) = J
P = P+1
W(P) = T2
C * * * * ADDING THE POSITIVE NODES SUM * * * *
DO 3301 JP = 1, TP(W(P))
M = FP(W(P), JP)
IF(IP(M-K).NE.0) GO TO 3301
SUM = SUM + VP(M-K)
RV = RV - VP(M-K)

```



```

IP(M-K) = W(P)
3301 CONTINUE
    IF(SUM.GE.TR) GO TO 302
    GO TO 307
C      ***** SUB BLOCK *****
3007 IF((P-1).EQ.0) GO TO 30011
3009 SUM =SUM-ABS(VN(W(P)))
    IN(W(P)) = 0
    DO 304 I1 = 1,L
    IF(IP(I1).NE.W(P)) GO TO 304
    SUM = SUM-VP(I1)
    RV=RV+VP(I1)
    IP(I1) = 0
304 CONTINUE
    PNN(W(P))= 0
    P=P-1
    GO TO 307
30011 SUM =SUM-ABS(VN(W(P)))
    PNN(W(P))=0
    DO 305 I1 = 1,L
    IF(IP(I1).NE.W(P)) GO TO 305
    SUM = SUM-VP(I1)
    IP(I1)= 0
    RV = RV+VP(I1)
305 CONTINUE
3001 CONTINUE
100 DO 5055 I=1,NUMPOSI
    WRITE(*,*)OPT(I)
    WRITE(*,*)(FP(OPT(I),J),J=1,TP(OPT(I)))
5055 CONTINUE
    WRITE(*,*)'OPTIMUM VALUE=',OPTIMUM
    GO TO 1002
1001 WRITE(*,*)OPTIMUM, I
    CALL GETTIM(IHR,IMIN,ISEC,I100TH)
1002 WRITE(*,*)CTIME,IHR,IMIN,ISEC,I100TH
    STOP
    END

C      ***** SUBPROGRAMMES *****

C      SUBPROGRAMME FOR COMPACTSTORAGE ALGORITHM
    SUBROUTINE LOMPT(NN,C1,N4,D1,Q)
    IMPLICIT INTEGER (A-Z)
    DIMENSION C1 (NN,NN),D 1 (NN,NN)
    DO 12 I = 1,N4

```

```

DO 12 J = 1 ,N4
L = C1(I,1) + C1(1,J)
IF(L.EQ.0) GO TO 14
DO 15 K = 1 ,N4
L = C1(I,K)+C1(K,J)
IF(L.EQ.0) GO TO 14
15  CONTINUE
C1(I,J)=1
C1(J,I) = 1
GO TO 12
14  C1(I,J)=0
C1(J,I)=0
12  CONTINUE
DO 20 I = 1 , N4
I1 = N4 - I + 1
DO 20 J = 1 , N4
J1 = N4 - J + 1
L = C1(I1,N4) + C1(N4,J1)
IF(L.EQ.0) GO TO 16
DO 30 K = 1 , N4
K1 = N4 - K + 1
L = C1(I1,K1) + C1(K1,J1)
IF(L.EQ.0) GO TO 16
30  CONTINUE
C1(I1,J1) = 1
C1(J1,I1)=1
GO TO 20
16  C1(I1,J1)=0
C1(J1,I1)=0
20  CONTINUE
DO 111 I = 1, N4
DO 111 J=1 ,N4
111 D1(I,J) = C1(I,J)
DO 70 I = 1, N4
DO 70 J = 1, N4
IF(D1(I,J).EQ.1) GO TO 71
70  CONTINUE
Q = 0
GO TO 72
71  Q = 1
72  WRITE(*,*)'VALUE OF Q = ',Q
RETURN
END
C  SUBROUTINE FOR ARRANGING THE VALUES OF SUPER NODES
SUBROUTINE LARANG(NN,IS,P,T,LA)

```

```

        INTEGER IS(NN),LA(NN),TEMP,T(NN),P
        DO 665 I = 1 , P
665    LA(I) = I
        DO 64 I = 1 ,P-1
        DO 64 J = I ,P
        IF(IS(I).GE.IS(J)) GO TO 64
        TEMP = IS(I)
        IS(I) = IS(J)
        IS(J) = TEMP
        LEMP = LA(I)
        LA(I)=LA(J)
        LA(J)=LEMP
64    CONTINUE
        DO 65 I = 1 ,P
65    T(I) = IS(I)
        WRITE(*,*) ' THE ARRANGED SUPER NODES VALUES '
        WRITE(*,68)(T(I),I= 1 , P)
68    FORMAT(5X, 10(2X,I4))
        RETURN
        END

```

```

C      PROGRAMME FOR QUADRATIC ASSIGNMENT PROBLEM
PARAMETER(N1=20,M1=N1*(N1-1)/2)
IMPLICIT INTEGER(A-Z)
INTEGER F(N1,N1),D(N1,N1),X(N1)
INTEGER L(N1),FW(N1),DW(N1),IFX(N1),IDX(N1),VF(M1),VD(M1)
INTEGER NBOUND(N1),LBOUND(M1),PX(M1),PY(M1),PZ(M1)
INTEGER PC(M1,M1),F1(N1,N1),D1(N1,N1),PQ(N1,N1),FW1(N1)
CHARACTER*18CTIME
LOGICAL SETTIM
DATA CTIME/'THE TIME TAKEN IS'/
IF(.NOT.(SETTIM(0.0,0.0)))WRITE(*,*)'SETTIM FAILED'
C      N IS SIZE OF THE DISTANCE MATRIX D AND FLOW MATRIX F
C      FW IS THE FACILITY WORD TR IS TRIAL SOLUTION VALUE
READ(V)N
DO 1 I=1,N
  READ(*,*)(D(I,J),J=1,N)
1      CONTINUE
DO 11884 IP=1,N
DO 11884 JP=1,N
  D1(IP,JP)=D(IP,JP)
11884  CONTINUE
DO 2 I=1,N
  READ(*,*)(F(I,J),J=1,N)
2      CONTINUE
DO 11886 IP=1,N
DO 11886 JP=1,N
  F1(IP,JP)=F(IP,JP)
11886  CONTINUE
  TR=0
DO 55 I=1,N-1
DO 56 J=I+1,N
  TR=TR+F(I,J)*D(I,J)
56      CONTINUE
55      CONTINUE
  KOUNT=0
DO 3 I=1,N-1
DO 4 J=I+1,N
  KOUNT=KOUNT+1
  VF(KOUNT)=F(I,J)
  VD(KOUNT)=D(I,J)
4      CONTINUE
3      CONTINUE
DO 7 I=1,KOUNT-1
DO 77 J=I+1,KOUNT

```

```

IF(VF(I).LT.VF(J)) GO TO 77
LEMP = VF(I)
VF(I)= VF(J)
VF(J) = LEMP
77  CONTINUE
7   CONTINUE
DO 8 I = 1,KOUNT-1
DO 88 J= I+1,KOUNT
IF(VD(I).GT.VD(J)) GO TO 88
LEMP = VD(I)
VD(I)= VD(J)
VD(J) = LEMP
88  CONTINUE
8   CONTINUE
DO 9 I = 1,N

IFX(I) = 0
IDX(I) = 0
9   CONTINUE
C   BOUND MATRIX CALCULATIONS
DO 750 IB = 1,N-1
NBOUND(IB) = 0
LE = NOEL(IB,2)
DO 751 J = 1,KOUNT-LE
J1=LE+J
NBOUND(IB) =NBOUND(IB)+VF(J)*VD(J1)
751 CONTINUE
750 CONTINUE
DO 752 IB = 1,N-1
752 CONTINUE
DO 1010 PJ=3,N
PCOUNT = 0
DO 1020 PI = 1,PJ-1
PCOUNT = PCOUNT+1
PX(PCOUNT)=D1(PI,PJ)
1020 CONTINUE
DO 1040 PI = 1 ,PCOUNT-1
DO 1050 PJ1 = PI+1,PCOUNT
IF(PX(PI).GT.PX(PJ1)) GO TO 1050
LTEMP = PX(PI)
PX(PI) = PX(PJ1)
PX(PJ1) = LTEMP
1050 CONTINUE
1040 CONTINUE
DO 1060 PI = 1,PJ-1

```

```

        D1(PI,PJ) = PX(PI)
1060    CONTINUE
1010    CONTINUE
        DO 1080 PJ=1,N
        PKOUNT = 0
        DO 1090 PI = UN
        PKOUNT = PKOUNT+1
        PY(PKOUNT) = F1(PI,PJ)
1090    CONTINUE
        DO 10100 PI = 1,PKOUNT-1
        DO 10110 PJ1=PI+1, PKOUNT
        IF(PY(PI).LT.PY(PJ1)) GO TO 10110
        LTEMP = PY(PI)
        PY(PI) = PY(PJ1)
        PY(PJ1) = LTEMP
10110    CONTINUE
10100    CONTINUE
        DO 10120 PI = UN
        F1(PI,PJ) = PY(PI)
10120    CONTINUE
1080    CONTINUE
        DO 10150 PJ = 2,N
        DO 10155 PI= UN
        PC(PI,PJ-1)=0
        DO 10160 K= 1,PJ-1
        PC(PI,PJ-1) = PC(PI,PJ-1) +D1(K,PJ)*F1(K,PI)
10160    CONTINUE
10155    CONTINUE
10150    CONTINUE
        DO 10277 PL = UN
        DO 10278 PJ= 1, N-1
        PQ(PL,PJ) = PC(PL,PJ)
10278    CONTINUE
10277    CONTINUE
C      BOUND MATRIX CALCULATIONS BOUND MATRIX IS PQ
        DO 901 PJ=N-2,1,-1
        KOUNT = 0
        PJ1 = PJ+1
        DO 902 PI = UN
        KOUNT = KOUNT+1
        PZ(KOUNT) = PQ(PI,PJ1)
        X(KOUNT) = KOUNT
902    CONTINUE
        DO 903 PPI = 1,N-1
        DO 904 PPJ =PPI+1,N

```

```

IF(PZ(PPI).LT.PZ(PPJ)) GO TO 904
LEMP = PZ(PPI)
PZ(PPI)=PZ(PPJ)
PZ(PPJ) = LEMP
LEMP = X(PPI)
X(PPI) = X(PPJ)
X(PPJ) = LEMP
904  CONTINUE
903  CONTINUE
IF(NBOUND(PJ1).GE.PZ(1)) GO TO 905
NBOUND(PJ1) = PZ(1)
DO 906 PPI = 1,X(1)-1
PQ(PPI,PJ) = PQ(PPI,PJ)+PZ(1)
906  CONTINUE
PQ(X(1),PJ) = PQ(X(1),PJ) + PZ(2)
DO 907 PPI = X(1)+1,N
PQ(PPI,PJ) = PQ(PPI,PJ)+PZ(1)
907  CONTINUE
GO TO 901
905  DO 908 PPI = 1,N
PQ(PPI,PJ) = PQ(PPI,PJ) + NBOUND(PJ1)
908  CONTINUE
901  CONTINUE
C ***** MAIN SEARCH *****
COUNT = 0
DO 11 I = L(1) +1,N
COUNT = 1
IF(NBOUND(COUNT).GE.TR) GO TO 11
SUM = 0
FW(COUNT) = 1
DW(COUNT) = COUNT
IFX(FW(COUNT)) = 1
IDX(DW(COUNT)) = 1
L(2) = 0
15  DO 12 J = L(2)+1,N
SUM=0
IF(IFX(J).EQ.0) GO TO 13
12  CONTINUE
GO TO 17
13  IT = J
IF((SUM + PQ(IT,COUNT)).GE.TR) GO TO 23
DT = COUNT +1
SUM = SUM + F(FW(COUNT),IT)*D(DW(COUNT),DT)
IF (SUM.LT.TR) GO TO 14
23  L(2) = J

```

```

GO TO 15
17  IFX(FW(COUNT))= 0
    IDX(DW(COUNT))= 0
    L(2)=0
11  CONTINUE
    GO TO 100
14  IFX(IT) = 1
    IDX(DT) = 1
    COUNT = COUNT +1
    FW(COUNT) = IT
    DW(COUNT) = COUNT
    L(COUNT)= J
2303 IF((SUM + NBOUND(COUNT)).GE.TR) GO TO 260
26  DO 24 J1 = L(COUNT+1)+1,N
    LP = L(COUNT+1)
    IF(IFX(J1).EQ.0) GO TO 250
24  CONTINUE
260  IF(COUNT.EQ.2) GO TO 151
    DO 25 J2 = 1,COUNT-1
    SUM = SUM - F(FW(J2),FW(COUNT))*D(DW(J2),DW(COUNT))
25  CONTINUE
    IFX(FW(COUNT))= 0
    IDX(DW(COUNT))= 0
    L(COUNT+1)= 0
    COUNT = COUNT-1
    GO TO 26
151  L(3) = 0
    IFX(FW(COUNT))=0
    IDX(DW(COUNT))=0
    COUNT = COUNT-1
    GO TO 15
250  IT = J1
    IF((SUM+PQ(IT,COUNT)).GE.TR) GO TO 29
    DT = COUNT +1
    TEMP = 0
    DO 27 J3 = 1, COUNT
    TEMP = TEMP + F(FW(J3),IT)*D(DW(J3),DT)
27  CONTINUE
    KEMP = SUM+TEMP
    IF((SUM+TEMP).LT.TR) GO TO 28
    GO TO 50
28  IFX(IT)= 1
    IDX(DT) = 1
    SUM1 =SUM + TEMP
112  SUM = SUM1

```



```

COUNT = COUNT+1
FW(COUNT) = IT
DW(COUNT) = DT
IF(COUNT.EQ.N) GO TO 30
L(COUNT)= J1
GO TO 2303
29  L(COUNT+1)= J1
GO TO 26
C    SOLUTION BLOCK
30  TR = SUM
OPTVAL = TR
DO 155 IO = 1,COUNT
155  FW1(IO) = FW(IO)
DO 31 J4 = 1,COUNT-1
SUM = SUM - F(FW(J4),FW(COUNT))*D(DW(J4),DW(COUNT))
31  CONTINUE
IFX(FW(COUNT)) = 0
IDX(DW(COUNT)) = 0
L(COUNT) = 0
COUNT = COUNT-1
DO 32 J5 = 1,COUNT-1
SUM = SUM - F(FW(J5),FW(COUNT))*D(DW(J5),DW(COUNT))
32  CONTINUE
IFX(FW(COUNT)) = 0
IDX(DW(COUNT)) = 0
COUNT = COUNT-1
GO TO 26
C    SUPER BLOCK
50  IF(COUNT.EQ.2) GO TO 151
DO 33 J6 = 1,COUNT-1
SUM = SUM - F(FW(J6),FW(COUNT))*D(DW(J6),DW(COUNT))
33  CONTINUE
IFX(FW(COUNT)) = 0
IDX(DW(COUNT)) = 0
COUNT = COUNT-1
GO TO 26
100 WRITE(*, 125)(F W1 (IL) ,IL= 1 .N),OPTV AL
125  FORMAT(12I7)
CALL GETTIM(IHR,IMIN,ISEC,I100TH)
WRITE(*,*)CTIME,IHR,IMIN,ISEC,I100TH
STOP
END
FUNCTION NOEL(N,K)
IF(N.LT.K) GO TO 72
L = FACT(N)/(FACT(K)*FACT(N-K))

```

```
NOEL = L
RETURN
72 NOEL = 0
RETURN
END
FUNCTION FACT(N1)
IF(N1)83,86,89
83 FACT = 0
RETURN
86 FACT = 1
RETURN
89 PRODUCT = 1
DO 2 I = 2 , N1
PRODUCT = PRODUCT*I
2 CONTINUE
FACT = PRODUCT
RETURN
END
```

APENDIX 3

The following table gives optimum values,time taken, sum of squares of timings for 10 problems..

Size = 75 Connectivity = 10% % of positive nodes == 10										
O.V	55	80	51	20	15	30	49	13	14	42
time	0.27	0.27	0.27	0.27	0.27	0.32	0.32	0.27	0.32	0.32
N.S.N	6	3	7		5	4	4	4	6	7
Mean time= 0.29 S.D.(time)=0.0245 C.V = 8.45 SS = 0.847										
Size = 75 Connectivity = 10% % of positive nodes = 20										
O.V	147	156	208	120	222	160	128	303	148	93
time	0.27	0.27	0.27	0.32	0.32	0.32	0.32	0.27	0.27	0.27
N.S.N	5	5	5	5	4	5	5	2	4	4
Mean time= 0.29 S.D.(time)=0.0245 C.V = 8.45 SS = 0.847										
Size = 75 Connectivity = 10% % of positive nodes = 30										
O.V	377	670	682	515	301	823	334	434	584	538
Time	0.27	0.27	0.32	0.27	0.27	0.27	0.32	0.27	0.32	0.32
N.S.N	10	4	3	2	6	3	4	2	4	5
Mean time= 0.29 S.D.(time)=0.024 C.V = 8.45 SS = 0.847										
Size = 75 Connectivity = 20% % of positive nodes = 10										
O.V	30	25	60	48	11	76	39	16	33	111
time	0.32	0.32	0.32	0.32	0.27	0.32	0.21	0.32	0.32	0.32
N.S.N	4	4	4	3	3	2	1	5	4	7
Mean time= 0.304 S.D.(time)=0.0035 C.V = 11.41 SS ≈ 0.9362										
Size = 75 Connectivity = 20% % of positive nodes = 20										
O.V	227	183	306	180	330	181	314	185	256	279
time	0.27	0.27	0.27	0.21	0.27	0.27	0.32	0.32	0.21	0.32
N.S.N	3	2	1	1	1	1	2	2	1	4
Mean time= 0.273 S.D.(time)=0.031 C.V = 14 SS = 0.7599										

Size = 100 Connectivity = 30% % of positive nodes = 10										
O.V	34	133	77	122	58	62	17	84	80	50
time	0.54	0.27	0.32	0.54	0.54	0.32	0.27	0.54	0.32	0.60
N.S.N	2	1	1	2	2	1	1	2	2	2
Mean time= 0.426 S.D.(time)=0.128 C.V = 30.12 S.S =1.9794										
Size = 100 Connectivity = 30% % of positive nodes = 20										
O.V	206	365	581	213	387	299	336	357	391	352
time	0.28	0.32	0.32	0.32	0.32	0.32	0.32	0.32	0.27	0.32
N.S.N	1	1	1	1	1	1	1	1	1	1
Mean time= 0.31 S.D.(time)=0.02 C.V = 6.45 S.S = 0.9705										
Size = 100 Connectivity = 30% % of positive nodes = 30										
O.V	817	112	851	974	849	819	699	671	687	956
time	0.32	0.27	0.32	0.27	0.27	0.27	0.27	0.27	0.27	0.32
N.S.N	1	1	1	1	1	1	1	1	1	1
Mean time= 0.285 S.D.(time)=0.023 C.V = 8.07 S.S = 0.8175										

Note: O.V - Optimal Value; N.S.N - Number of Super nodes; S.S- sum of squares of timings,time is given in seconds

REFERENCES

REFERENCES

Agin, N. (1966) : 'Optimum seeking with Branch and Bound', Management Science, Vol.No.. 13, 4B,176-185

Armour,G.C., and Buffa, E.S. (1963): 'A heuristic algorithm and Simulation approach to relative location of facilities', Management Science, 9, pp.294-309.

Bazaraa,M.S., and Elshafei, A.N. (1979): 'An exact Branch and Bound Procedure for Quadratic Assignment Problems', Naval Research Logistic Quarterly, Vol.No..26, pp. 108-121.

Bazaraa,M.S., and Kirca, O. (1983) : ' A Branch and Bound heuristic for solving the quadratic assignment problem', Naval Research Logistic Quarterly, Vol.No.. 30, pp.287-304.

Bazaraa,M.S.,and Sherali,H.D.(1980): 'Benders Partitioning scheme applied to new formulation of Quadratic Assignment Problem', Naval Research Logistic Quarterly, Vol.No..27, pp. 29-41.

Bhanumurthy,A.(1986): 'Some Frequencing and Scheduling Problems',. Ph.D. Thesis, Osmania University, Hyderabad, India.

Bellman,R.E. (1962) : 'Dynamic Programming treatment of the travelling salesman problem', J. ACM. 9, pp.61-63

Bellmore,M., and Malone, J.C.(1971) : 'Pathology of travelling salesman subtour elimination algorithms', Oper. Res., 19,pp.278-307.

Burkard,E.R., and Stratmann. K.H. (1978) : 'Numerical Investigations on Quadratic Assignment Problems', Naval Research Logistic Quarterly, Vol.No.. 25, pp. 129-148.

Caccetta, L., and Giannini,L.M. (1988) : 'An application of discrete mathematics in the design of an open pit mine', Discrete Applied Mathematics, 21, pp. 1 -19.

Chadrasekar Reddy, T.(1987): 'On Routing & related Problems', M.Phil. Dissertation, University of Hyderabad, Hyderabad.

Christofides,N.(1970): 'The Shortest Hamiltonian chain of a graph', SIAM J. Appl. Mathematics, 19, pp.689-696.

Clarke G.,and Wright, J.W. (1964) : 'Scheduling of Vehicles from a central depot to number of delivering points', Operations Research, 12, pp.568-581.

Croes,G.A.(1958): 'A method for solving travelling salesman problem', Operations Research, 6, pp.791-812.

Dantzig,G.B., Fulkerson,D.R., and Jhonson,S.M.(1954): 'Solutions of a large scale Travelling salesman Problem', Operations Research, 2, pp. 393-410.

Das,S.(1976): ' Routing and Allied Combinatorial Programming Problems', Ph.D. Thesis, Dibrugarh University Assam, India.

Dakin,R.J. (1965) : 'A tree search algorithm for mixed integer programming problems', Comput. J. Vol.No.. 8, pp.250-255

Eastman, W.L. (1958): 'Linear Programming with pattern constraints'. Pl. Dissertation, Herward.

Edward, C.S. (1980): 'A Branch and Bound algorithm for the Koopmans-Beckman Quadratic Assignment Problem', Mathematical Programming Study, 13,pp.35-52.

Frederickson,G.N., Hecht, M.S., and Kin, C.E. (1978) : 'Approximation algorithms for some routing problems', SIAM J. Comput 7,pp.178-193.

Flood, M.M.(1956): 'The travelling salesman problem', Oper.Res., 4,pp.61-75.

Gilmore, P.C. (1962) : 'Optimal & suboptimal algorithms for the Quadratic Assignment Problem' J. Soc. Indst and Appl. Math. 10,2, pp.305-313.

Graves, G.W., and Whinston, A.B.(1970): 'An algorithm for the Quadratic Assignment Problem', *Management Science* 17,7,pp.453-471.

Gavett, J.W., and Plyter, N.V. (1966): 'The Optimal Assignment of facilities to locations by Branch & Bound'. *Operations Research*, 14, pp. 210-232

Gupta, J.N.D.(1967): 'Combinatorial search algorithms for Scheduling Problem', M.Tech Thesis, IIT, Kharagpur.

Hadely, G.(1994) : 'Linear Programming', fifth reprint, Indian Student Edition, Narosa Publishing House, New Delhi.

Hardgrave, W.W., and Nemhauser G.L.(1962): 'On the relation between the travelling salesman problem and the longest path problems', *Opr.Res.* 10, pp. 647-657.

Hillier, F.S (1963): 'Quantitative Tools for Plant layout Analysis', *J.Ind.Eng* 14, pp. 33-40

Hillier, F.S., and Connors, M.M. (1966): 'Quadratic Assignment algorithms and the location of Indivisible Facilities', *Management Science*, 13, pp. 42-57.

Heider, C.H.(1973): 'An N- Step, 2 variable search algorithm for the component placement problem'. *Naval Research Logistic Quarterly*, Vol.No..20, pp.699-724.

Held, M., and Karp, R.M. (1962) : 'A dynamic programming approach to sequencing problems', *SIAM J. Appl. Mathematics* ,10, pp. 196-210

Jain, S.C, Misra, R., and Pandit, S.N.N.(1964): 'Optimal Machine Allocation'. *J.Inst. Engs., India*, 44, pp.226-240

Kambo, N.S.(1984): 'Mathematical Programming Techniques', Affiliated East-West Press Pvt. Ltd.

Karp, R.M. (1979): 'A Patching algorithm for the non symmetric travelling salesman problem', *Siam J. Comput*, 8, pp.561-573.

Kaufman,L., and Broeckx,F.(1978): 'an algorithm for the Quadratic Assignment Problem using Benders decomposition', European Journal of Operational Research, 2, pp.204-211.

Koopman,T.C. & Beckmann, M. (1957) : 'Assignment problems and Locations of Economic activities', Econometric Vol.No.. 25 ,No 1, pp. 53-76

Land, A.H.,(1963) : 'A problem of Assignment with interrelated costs', Operational Research Quarterly Vol.No.14 ,pp.185-198.

Lawler, E.L.(1963) : 'The Quadratic Assignment Problem', Management Science, 9, pp.586-599.

Lawler E.L., Lenstra J.K.,Rinnooy Kan ,A.H.G., and Shnoys, D.B. (1985): The travelling salesman problem , John Wiely & Sons Ltd.

Lawler, E.L., and Wood, D.E. (1966): 'Branch and Bound Methods- A Survey', Operations Research, 14, pp.699-719

Lenstra, J.K., Rinnooy Kan, A.H.G. (1975): 'Some Simple applications of the travelling salesman problem', Operations Research quarterly, 26,pp.717-733.

Lin, S., and Kernighan, B.W.(1973): 'An effective heuristic algorithm for the travelling salesman problem', Oper. Res 21, pp.498-516

Little,J.D.C, Murthy, K.G., Sweeny, D.W., and Karel, C. (1963): ' An algorithm for the travelling salesman problem', Operations Research ,11 pp.979-989.

Love, R.F., and Wong, J.Y. (1976) : 'Solving Quadratic Assignment Problems with rectangular distances and integer programming', Naval Research Logistic Quarterly Vol.No 23,pp. 623-627.

Miller, C.E., Tucker, A.W., and Zemlin, R.A.(1960): 'Integer Programming Formulations and travelling salesman problem'. J. ACM., 7 ,pp.826-832.

Nugent,C.E., Vollman,T.E.,and Ruml,J.(1968): 'an Experimental comparison of techniques for the assignment of facilities to locations', Operations Research 16,pp.150-173

Pandit, S.N.N. (1961): 'A new matrix calculus', Journal of the Society for Industrial and applied Mathematics, 9, pp.632-639

Pandit, S.N.N. (1962): 'The Loading Problem', Operational Research Vol.No.. 10, pp.639-646.

Pandit,S.N.N.(1963): 'Some Quantitative Combinatorial Search Problems'. Ph.D Thesis, I I T, Kharagapur.

Pandit, S.N.N (1964): 'A bracketing bound technique for the assignment problem', Journal of national productivity Council, India.

Pandit, S.N.N. (1965): 'An intelligent Search approach to the travelling salesman problem', Symposium in O.R. IFF, Khargpur.

Pandit, S.N.N., and Rajabongshi,M.(1976): 'Restricted travelling salesman problem through N sets of nodes'. Paper presented at the annual convention of O R S I December 3-5, Calcutta.

Pierce,J.F., and Crowston, W.B.(1971): 'Tree Search Algorithms for Quadratic Assignment Problems', Naval Research Logistic Quarterly, Vol. No 18, pp. 1-35.

Rajbongshi, M.(1982): 'Sequencing and allied Combinatorial Programming Problems', Ph.D. Thesis, Osmania University, Hyderabad.

Ramesh,T.(1980): 'Some Problems in Min-Max. and Combinatorial Programming', Ph.D Thesis, Kakatiya University, Warangal.

Ravi Kumar,M.(1989): 'A Comparative Evaluation of some 0-1 programming Algorithms', M.Phil. Dissertation ,University of Hyderabad, Hyderabad.

Ravi Kumar, M.(1995): ' Data guided algorithms in combinatorial optimisation', Ph.D. Thesis, Osmania University, Hydrabad.

Rao, S.S.(1984) : 'Optimisation- Theory and applications', 2nd edition Wiley Eastern Limited.

Roucariol C (1978) : ' A reduction method for quadratic assignment problems', 1, pp.78-87, Universite pierre et Marie Curie, Paris.

Sambasiva Rao T.V. (1990): 'Minimum value estimation in bounded populations - An Empirical Study', M.Phil. Thesis, University of Hyderabad.

Scroggs, R.E.,and Tharp A.L.(1972):'An algorithm for solving travelling salesman problem in restricted context', privately circulated

Shapiro, D.M (1966): 'Algorithms for the solution of the optimal cost and Bottleneck travelling salesman problems', Sc.D thesis, Washington University, St Louis M O.

Sherali H.D.(1979) : 'The quadratic assignment problem: Exact and Heuristic Methods', unpublished doctoral dissertation, Georgia Institute of Technology.

Smith,T.H.C.,Srinivasan,V., and Thompson,G.L.(1977): 'Computational performance of three subtour elimination algorithms for solving asymmetric travelling salesman problems', Ann. Discrete Mathematics, 1, pp.495-506

Srinivas, K .(1990): 'Data Guided algorithms in Optimisation and pattern recognition'. Ph.D. Thesis, University of Hyderabad, Hyderabad.

Sriwastav,G.L. (1976) : ' some mathematical models in theory of Reliability', Ph.D. Thesis, Dibrugarh University, Assam, India.

Sundar Murthy, M.(1979): 'Some Combinatorial Search Problems (A pattern recognition approach)', Ph.D. Thesis, Kakatiya University , Warangal.

Subramanyam, Y.V. (1980): 'Scheduling Transportation and Allied Combinatorial Programming Problems', Ph.D. Thesis, Kakatiya University, Warangal.

Wagner Hervey, M. (1973) : 'Principles of Operations Research with applications of managerial decisions', Prentice-Hall of India Pvt. Ltd New Delhi.