T22

CENTRAL LIBRA TH ~~38~~ 43018
TEZPUR UNIVE
Accession No. T 22
Date

Tezpur University Library
Call No.
Acc. No. TH ~~38~~ 43018
Date 14.06.2004
Tezpur University

REFERENCE BOOK
NOT TO BE ISSUED
TEZPUR UNIVERSITY LI

REVISED

A SEQUENTIAL CONSTRUCTIVE SAMPLING AND RELATED APPROACHES TO COMBINATORIAL OPTIMIZATION

A Thesis submitted in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Tezpur University Library
Call No 4.30/18
Acc No TH.38
Date 14.6.2004
Tezpur University

TH-~~38~~
43018

by

ZAKIR HUSSAIN AHMED

DEPARTMENT OF MATHEMATICAL SCIENCES
SCHOOL OF SCIENCE AND TECHNOLOGY
TEZPUR UNIVERSITY
Tezpur-784 028, INDIA

AUGUST, 2000

Dedicated To

My Beloved Parents

Begum Jahanara Ahmed

&

Abdus Sahid Ahmed

CERTIFICATE

This is to certify that the thesis entitled **"A SEQUNETIAL CONSTRUCTIVE SAMPLING AND RELATED APPROACHES TO COMBINATORIAL OPTIMIZATION"** submitted for the award of Doctor of Philosophy to the Tezpur University, Tezpur, Assam, is a record of bonafide research carried out ***Mr. Zakir Hussain Ahmed*** under our supervision and guidance. No part of this thesis has been submitted for any degree or diploma of any other University.

Supervisors:

[illegible]

1. (Prof. S.N.Narahari Pandit)
Director,
Center for Quantitative Methods,
Osmania University, Hyderabad.

[illegible]

2. (Dr. Munindra Borah)
Head,
Department of Mathematical Sciences,
Tezpur University, Assam.

ACKNOWLEDGMENTS

It is my pleasure and privilege to express my very deep sense of gratitude to Prof. S. N. Narahari Pandit, Director, Center for Quantitative Methods, Osmania University, Hyderabad (and formerly visiting Professor of Department of Computer Science, Tezpur University, Tezpur), who initiated me into this interesting field of research, and whose esteemed guidance has enabled me to present this thesis.

Also I express my special appreciation to Dr. Munindra Borah, Head, Department of Mathematical Sciences, Tezpur University, firstly for agreeing to be my co-supervisor and secondly, for his valuable guidance.

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

I am thankful to Prof. A. K. Borkakoty and Prof. S. K. Laskar, Department of Mathematical Sciences; Prof. D. K. Saikia, Head, Department of Computer Science; Tezpur University and other faculty members of various departments for their constant encouragement and moral support.

My special thanks goes to Ms. Shanta Pandit for her motherly behavior and Dr. V. V. Haragopal of Osmania University for hostel accommodation in New Research Scholars' Hostel during my visit to Hyderabad.

I wish to thank the authorities of the Computer Center of Tezpur University, Navigational Electronics Research and Training Unit, and Center for Quantitative Methods, Osmania University for extending computational facilities.

I offer my thanks, for the interest and encouragement accorded by my well wishers, friends- Anjan, Deep, Jamal, Sofiur, Dhiraj and Mrs. Roshanara Begum of Tezpur University and Laxminarayan, Pavan, Hamid, Sampath and Selim of Osmania University.

Finally, I thank the North Eastern Council, Shillong and Council of Scientific and Industrial Research, New Delhi for providing financial assistance for my research work.

[illegible]

(Z. H. Ahmed)

CONTENTS

Chapter I INTRODUCTION 1

1.1 The Optimal Assignment Problem 2

1.2 The Travelling Salesman Problem 2

1.3 The Shortest Route Problem 2

1.4 The Job-Scheduling Problem 2

1.5 The Knapsack Problem 2

1.6 Chapter-wise summary 4

Chapter II: THE TRAVELLING SALESMAN PROBLEM-A SURVEY 6

2.1 Introduction 6

2.2 Methods of solution-general comments and classification 8

2.2.1 Exact methods 9

2.2.1.1 Integer programming approach 9

2.2.1.2 Dynamic programming approach 10

2.2.1.3 Branch and bound approach 11

2.2.1.4 Lexicographic search 12

2.2.2 Approximate solution methods (heuristics) 13

2.2.2.1 Greedy algorithms 13

2.2.2.2 Simulated annealing 21

2.2.2.3 Genetic algorithm approach 23

2.3 Some variations of the usual TSP 24

2.3.1 The Truncated Travelling Salesman Problem 24

2.3.2 The Precedence Constrained Travelling Salesman Problem... 24

2.3.3 The Fixed Positional Travelling Salesman Problem 24

2.3.4 Vehicle Routing Problem 24

2.3.5 The Clustered Travelling Salesman Problem 25

2.3.6 "M" Travelling Salesman Problem 26

2.3.7 The Traveller Purchaser Problem 26
2.3.8 The Time Constrained Travelling Salesman Problem 26

Chapter III: THE LEXI SEARCH APPROACH S FOR THE TSP 27
3.1 Introduction 27
3.2 Adjacency approach and illustration 31
3.3 Path approach and illustration 40
3.2.1 Bound calculation 42
3.2.2 Illustration 43
3.4 Sequential constructive sampling approach and illustration 47
3.4.1 Bound calculation 50
3.4.2 Illustration 50
3.4.3 Modified 2-Opt move 54

Chapter IV: GENTIC ALGORITHM APPROACH 56
4.1 Introduction 56
4.2 Genetic algorithm implementation 57
4.3 Control parameters 61
4.4 Genetic algorithm for the TSP- a review 62
4.5 The sequential constructive operator 70
4.6 Hybrid genetic algorithm for the TSP 73
4.7 Relative efficiency of different approaches 77

Chapter V: THE TRAVELLING SALESMAN PROBLEM WITH PRECEDENCE CONSTAINTS 80
5.1 Introduction 80
5.2 Lexisearch approach and illustration 83
5.2.1 Illustration 85
5.3 Sequential constructive sampling approach and illustration 89
5.3.1 Illustration 90
5.4 Hybrid genetic algorithms for the TSP-PC 92

5.5 Relative efficiency of different approaches 94

Chapter VI: THE TRAVELLING SALESMAN PROBLEM WITH FIXED POSITION CONSTRAINTS 98

6.1 Introduction 98

6.2 Lexisearch approach and illustration 98

6.2.1 Illustration 99

6.3 Sequential constructive sampling approach and illustration 103

6.3.1 Illustration 103

6.4 Hybrid genetic algorithm for the TSP-FPC 106

6.5 Relative efficiency of different approaches 108

Chapter VII: THE TRAVELLING SALESMAN PROBLEM WITH FIXED POSITION AND PRECEDENCE CONSTRAINTS 111

7.1 Introduction 111

7.2 Lexisearch approach and illustration 111

7.2.1 Illustration 112

7.3 Sequential constructive sampling approach and illustration 115

7.2.1 Illustration 116

7.4 Hybrid genetic algorithms for the TSP-FPPC 119

7.5 Relative efficiency of different approaches 121

Chapter VIII: THE TRAVELLING SALESMAN PROBLEM WITH BACKHAULS 125

8.1 Introduction 125

8.2 Lexisearch approach and illustration 127

8.2.1 Illustration 128

8.3 Sequential constructive sampling approach 131

8.4 Hybrid genetic algorithm for the TSPB.................................. 131

8.5 Relative efficiency of different approaches 133

Chapter IX: THE MIN-MAX TRAVELLING SALESMAN PROBLEM 136

9.1 Introduction 136

9.2 The adjacency approach and illustration 136

9.2.1 Illustration 137

9.3 Path approach and illustration 139

9.3.1 Bound calculation 139

9.3.2 Illustration 140

9.4 Sequential constructive sampling approach 141

9.5 Hybrid genetic algorithm for the MMTSP 142

9.6 Relative efficiency of different approaches 142

Chapter X: DISCUSSIONS AND COMMENTS 145

APPENDIX 152

BIBLIOGRAPHY 254

Research paper published in the Journal from the Thesis:—

- **The Travelling Salesman Problem with Precedence Constraints,** *OPSEARCH,* Vol. 38, No.3, pp 299-318, 2001. (by Ahmed, Z.H., and S.N.N. Pandit)

CHAPTER I
INTRODUCTION

The problem of optimization (or Mathematical Programming) can be construed as one of extremizing a function (viz., objective function) of some decision variables, which latter are to satisfy some constraints. Depending on the nature of the constraints and objective function, one has many types of optimization problems like Linear programming, Convex programming, Geometrical programming etc. In particular, when the decision variables are continuous, it is called Continuous Programming Problem. This type of problems is solved by some procedures which are based on concepts like continuity, convexity and neighborhood. Some of them are Simplex procedure, Lagrangian multiplier method and Steepest Ascent method.

If the decision variables are not continuous but rather discrete (e.g., confined to a set of integers), it is called Integer Programming Problem. The decision variables may also be associated with Permutations, Combinations etc. Then the optimization problem is said to be a "Combinatorial Programming Problem" and the problem of finding optimal solution to such a problem is known as "Combinatorial Optimization Problem".

The general structure of the Combinatorial Programming Problems can be described as follows (cf, Pandit 1963):

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

Following are some well-known combinatorial programming problems.

1.1 THE OPTIMAL ASSIGNMENT PROBLEM

LP

There are *n* workers and *n* jobs. Each worker must be assigned to exactly one job and vice-versa. The ~~cost~~ value of assigning worker *i* to job *j* is ~~c_{ij}~~ a_{ij}. The problem is to find the optimal assignment so that the total cost of the assignment is ~~minimum~~. max

you mean the Quadratic Assign Pr

1.2 THE TRAVELLING SALESMAN PROBLEM

A network with 'n' nodes (or cities) and a cost matrix of order *n* associated with each node pair (i, j) is given. The problem is to find a least cost tour for a salesman who must visit each of the nodes exactly once except that the salesman should end the tour at the same node from which he starts.

1.3 THE SHORTEST ROUTE PROBLEM

There are *n* nodes connected with one another by '*links*' (or arcs) of given lengths (weighted network). The problem is to find the shortest route from one specified node to another specified node, if necessary through other nodes in the network.

L P or D P

1.4 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 can not be processed on machine 'j' until it is finished on machine 'j-1'. The processing times of each job on each machine are given. The problem is to find a sequencing of jobs (defined as a schedule) that minimizes the total elapsed time (makespan) to complete all the jobs on all the machines.

Johnson solution

1.5 THE KNAPSACK PROBLEM

An instance *KNAP(A, c)* of the knapsack problem consists of a set *A* of *m* objects, and an integer knapsack capacity, denoted by *c*. Each object $a_i \varepsilon A$ has positive weight w_i and profit p_i. For each object a_i that is placed in the knapsack, a weight ~~profit~~ ~~p_i~~ is earned. The object is to fill the knapsack in such a way that the total ~~profit~~ utility is maximized.

Most of the combinatorial programming problems are NP-complete or NP-hard (cf, Papadimitriou et al., 1997). define To solve such problems one has can ~~simply~~ to list all the possible solutions find out their objective function values and pick out the best. It is easy to solve any problem in this way when the size is small enough. However this 'complete enumeration approach' is not practicable even for problem of moderate size. In fact for most of the combinatorial programming problems, the computational time grows exponentially with the problem size. Special techniques are available for solving different combinatorial programming problems. However, no general approach, which is suitable for all combinatorial programming problems, seems to be available and methods of finding optimum solutions are largely search methods (cf, Pandit 1963). branch & / bound

Many real life problems, when idealized as mathematical problems-for their solution-will necessarily involve approximations, both conceptual and in numerical estimation of the parameters involved. It is therefore, obvious that good, practicable solutions to approximately well formulated problems are more useful than exact optimal solution of very approximately formulated problems. This is particularly true of combinatorial problems, where conventional criteria of approximations through continuity-based neighborhood approaches are not applicable.

Hence, currently, in the field of operations research and decision sciences, emphasis has shifted from the aim of finding exactly optimal solutions to combinatorial optimization problems, to the aim of getting, heuristically, 'good solutions' in reasonable time and 'establishing the degree of goodness'. Artificial Neural Network (ANN), Genetic Algorithm (GA), Simulated Annealing (SA) are a few such approaches.

In this present dissertation we shall develop Lexicographic search (Lexiseach, for short) approach for obtaining exact optimal solutions to various combinatorial optimization problems considered, and a quasi-exact method, which provides solution within a pre-specified neighborhood of the exact optimal solution to the problems. Also, a sequential constructive sampling approach, and a hybrid genetic algorithm have been developed for obtaining heuristically optimal solution to each of the problems. The efficiency of the hybrid genetic algorithm to each of the problems as against lexi-search approach, quasi-exact method and sequential constructive sampling approach has been examined by solving a variety of randomly generated test problems of different sizes.

1.5 CHAPTER-WISE SUMMARY

The thesis presents solution of the usual Travelling Salesman Problem (TSP) and some of its variations. Our main aim is to develop exact and heuristic algorithms to some Restricted TSPs. For that, we first developed the algorithms for the usual TSP and then modified them in the context of the required restrictions. Hence, we have reported the algorithms explicitly for the usual TSP and the modifications to the algorithms due to the constraints have been reported only for the other restricted TSPs. All the algorithms are coded in **Borland C** and the **Programs** are reported in the **Appendix**. Finally a comparative study, examining the relative efficiency of different approaches, is carried out for each problem. The computation is reported in **seconds** on a **Pentium/ 450 MHz/ 64 MB RAM.**

Chapter II discusses the usual TSP with the objective to minimize the total tour cost. Some of the available algorithms for solving the same are reported briefly.

In **Chapter III**, some of the basic concepts of the Lexisearch approach are explained. Also, this approach is explained with the usual TSP in the context of both path representation and adjacency representation of a tour. A comparative study of both these approaches is then carried out. It is also shown that the 'bias' removal of the given matrix plays an important role in case of time taken for solving the same. Also, this chapter develops a quasi-exact method based on lexisearch approach and a sequential constructive sampling approach.

Chapter IV discusses the basic concepts of the genetic algorithm with different proposed operators in context of solving the usual TSP. We have proposed here a new operator, named sequential constructive search operator, and tested the efficiency of the operator by comparing with some other crossover operators proposed by various researchers. Then a hybrid genetic algorithm is developed for the usual TSP.

Chapter V discusses the TSP with Precedence Constraints, in which every tour of the salesman has to satisfy some precedence constraints.

Chapter VI discusses the TSP with Fixed Position Constraints, in which every fixed position in the tour of the salesman is to be occupied by the prescribed node.

The TSP with Fixed Position and Precedence Constraints is discussed in **Chapter VII**, where each tour of the salesman is to satisfy both Fixed Position and Precedence Constraints as discussed in Chapter V and VI.

Chapter VIII discusses the TSP with Backhauls, in which the nodes in the network (except the starting node) are divided into two sets of nodes- linehaul and backhaul nodes, and all linehaul nodes, should be visited contiguously after the starting node and before all the backhaul nodes.

Chapter IX discusses the usual TSP but with a different objective, viz., to *minimize* the largest arc length of the tour.

The thesis concludes with some general observations on the problems considered in the thesis, which are given in **Chapter X**.

CHAPTER II

TRAVELLING SALESMAN PROBLEM-A SURVEY

2.1 INTRODUCTION

The Travelling Salesman Problem (TSP) is a relatively old problem: it was documented as early as 1759 by Euler (though not by that name), whose interest was in solving the Knights' tour problem (cf, Michalewicz 1994). A correct solution would have a knight visit each of the 64-squares of a chessboard exactly once in its tour.

[illegible] ref

The term 'travelling salesman' was first used in 1932, in a German book 'The travelling salesman, how and what he should do to get commissions and be successful in his business', written by a veteran travelling salesman (ibid.).

The TSP can be stated as follows:

A network with 'n' nodes, with node 1 as 'headquarters' and a travel cost (or distance, or travel time etc.) matrix $C=[c_{ij}]$ of order n associated with ordered node pairs (i, j) is given. The problem is to find a least cost Hamiltonian cycle. That is the problem is to obtain the tours $\{1=\alpha_0, \alpha_1, \alpha_2, \ldots, \alpha_{n-1}, \alpha_n=1\} \equiv \{1 \to \alpha_1 \to \alpha_2 \to \ldots \to \alpha_{n-1} \to 1\}$ representing irreducible permutations interpreted as simple cycles for which the total travel cost

$$C\,(1=\alpha_0, \alpha_1, \alpha_2, \ldots, \alpha_{n-1}, \alpha_n=1) \mathrel{\hat{=}} \sum_{i=0}^{n-1} c\,(\alpha_i, \alpha_{i+1})$$

is minimum.

The RAND Corporation introduced the TSP in 1948. The Corporation's reputation helped to make the TSP well-known and popular problem. The TSP also became popular at that time due to the new subject of linear programming and attempts to solve Combinatorial Optimization Problems.

On the basis of the structure of the distance (or time or cost) matrix, the TSPs are classified into two groups: symmetric and asymmetric. The TSP is symmetric if $c_{ij}=c_{ji}$, $\forall i, j$ and asymmetric otherwise. For an n-city asymmetric TSP, there are (n-1)! possible ~~solutions~~ tours, one or more of which gives the minimum cost. For an n-city symmetric TSP, there are (n-1)!/2 possible ~~solutions~~ tours along with their reverse cyclic permutations having the same total cost. In either case the number of solutions becomes extremely large for large n, so that an exhaustive search is impracticable. The TSP was proved to be NP-Complete (cf, Papadimitriou et al., 1997). NP-Complete problems are intractable in the sense that no one has found any really efficient way of solving them for large n. They are also known to be more or less equivalent to each other; if one knew how to solve one kind of NP-Complete problem one could solve the lot. Also, all the other problems considered in this present dissertation are said to be NP-hard. No

exact def of NP Complete

The TSP finds ~~the~~ application in a variety of situations. As an example, consider a scenario in which a robot arm is used to tighten the nuts on some piece of machinery on an assembly line. The arm starts from its initial position (which is over the first nut to be tightened), successively moves to each of the remaining nuts and returns to the initial position. The path of the arm is clearly a tour with cities represented by nuts. A minimum cost tour will minimize the time needed for the arm to complete its task. Only the total arm movement time is variable; the nut tightening time being independent of the tour. Other applications have also been described, as for example, automatic drilling of printed circuit boards, where the movement of the robot arm must be minimized, and threading of scan cells in a testable VLSI circuit, where cost of the scan path must be minimized (cf, Ravikumar 1992), experiments in x-ray crystallography involve sequential collection of data on a diffractometer, where a small sample of crystal under study is mounted in the apparatus, which has computer-driven motors that can accurately position the crystal and a detector. The time to repositioning the apparatus between reading is the delay time. The problem is of sequencing the readings to minimize the total delay. The readings correspond the nodes of the TSP. The length of an edge of the TSP is the delay time for repositioning the apparatus from one setting to other (cf, Bland and Shallcross 1989).

[illegible] A [illegible] the cycle

Though c_{ij}'s above are referred as 'travel cost ~~of~~ to the node j from the node i', it is obvious that they need not be 'metric', i.e., satisfy the usual distance postulates like symmetry and triangular inequality; they can be distances or times for direct travel from node i to node j, the 'energy' required for transiting a 'system' from state i to state j in one 'step' etc.; in fact, it can be any non-negative quantity which is additive.

In this chapter, we have confined our study to the 'usual' TSP in which the salesman visits each node once and only once in the entire tour.

NO The combinatorial structure of this problem is very close to that of the Assignment Problem (AP). Basic difference between the AP and the TSP solution-space is that former can be viewed as the set of all permutations of 'n' elements while the latter are restricted to 'in–decomposable' permutations only. In-decomposable permutations are the permutations that have single cycle and can't be decomposed into more than one permutation.

It may be noted in passing that the solution space of the TSP is a subset of solution space of AP and hence, steps like bias-removal (i.e., subtract each row-mini~~ma~~um from its corresponding row elements. Repeat the same column-wise on the resultant matrix. The total of the row-minima and the subsequent column-minima is called the 'bias' of the matrix), which are useful in the latter context can also be applied in the case of TSP as well (cf, Pandit 1964). proof ?

2.2 METHODS OF SOLUTION-GENERAL COMMENTS AND CLASSIFICATION

The methods for solving the TSP, which also can be applied to an~~y~~ other optimization problem, can be classified mainly in two classes based on different viewpoints or purposes. They are as follows:

2.2.1 EXACT METHODS

The methods provide the exact optimal solution to the problem, are called exact methods. An implicit way of solving the TSP is simply to list all the feasible solutions, evaluate their objective function values and pick out the best. However it is obvious that this 'complete enumeration approach' is grossly inefficient and impracticable because of vast number of possible solutions to the TSP even for problem of moderate size. Quite a few exact algorithms, which find exact optimal solution to the problem much more efficiently than complete enumeration, have been developed. Some of them are briefly discussed below.

2.2.1.1 INTEGER PROGRAMMING APPROACH

The TSP can be formulated as an integer linear programming as follows:

Define x_{ij}, a binary variable equal to 1 (one), if and only if the edge (i, j) belongs to the solution, else 0 (zero). Then the TSP can be formulated as

$$\text{Minimize } \sum_{i,j=1}^{n} c_{ij}x_{ij} \ ; i \neq j.$$

$$0 \leq x_{ij} \leq 1, \ \forall i, j$$

$$x_{ij} \text{ integer}, \ \forall i, j$$

Subject to the constraints

$$\text{(a) } \sum_{i=1}^{n} x_{ij} = 1, \ j = 1,2,\ldots,n$$

$$\text{(b) } \sum_{j=1}^{n} x_{ij} = 1, \ i = 1,2,\ldots,n$$

$\geq$ (c) $\sum_{i,j\in S} x_{ij} \nleq |S| - 1$, where S ranges over all possible subsets of size $|S|$, $S \subset \{2,3,\ldots\ldots,n\}$.

The equalities in (a) express the fact that exactly one arc enters each node, (b) exactly one leaves each of the nodes 1,2,....,n and (c) confirms the exclusion of cycle formulation. The objective function with only constraints (a) and (b) leads to the Assignment Problem.

The difficulties in finding an optimal tour in solving the above integer program are the enormous number of loop constraints ($2^{n-1}-1$) and the requirement that the $(n^2-n)/2$ variables x_{ij} equal to 0 or 1. The solution of a linear program with the loop constraints and the constraints $0 \le x_{ij} \le 1$ will not necessarily satisfy $x_{ij}=0$ or 1. This is in contrast with the situation for solution of Transportation Problem, which guarantees an optimal integer solution if supplies and demands are integers.

It is Interesting to note that Dantizig et al. (1954) found an optimal solution to a 42-city problem using linear programming. They overcame the large number of loop constraints by beginning with only a few, and then adding new ones only as they were needed to block sub-tours. Combinatorial arguments were used to eliminate fractional solutions and to find an optimal tour. Finally, it was demonstrated that for the problem at hand, an ordinary linear program could be devised whose solution gave integer-valued x_{ij}'s representing the optimal tour. The constraints that rule out some fractional solutions, but no integer solutions were forerunners to Gomory's 'cutting plane' constraints for solving any integer linear programs (cf, Gomory 1963).

Miller et al. (1960) earlier formulated the problem by replacing the sub-tour eliminating constraint (c) by

$$u_i-u_j+nx_{ij} \le n-1;\ 1\le i\neq j \le n,$$

where u_i, $i=1,2,\dots,n$ are unrestricted real variables. However results were rather disappointing (cf, Srinivas 1989).

2.2.1.2 DYNAMIC PROGRAMMING APPROACH

Dynamic programming is the mathematical technique whose development is largely due to Richard E. Bellman. It is applicable to many types of problems where in a series of interrelated decisions is required. This technique of dynamic programming divides a given problem into sub problems and then solve the sub-problems sequentially (usually working backward from the natural end of the problem) until the initial problem is finally solved. This technique of solving a given problem is often termed as recursive approach. The principle behind the operation of this technique is known as principle of optimality, which as set forth by Bellman, states that "An optimal solution has the property: whatever the initial state and decision are, the remaining decisions must constitute an optimal sequence with regard to the state resulting from the first decision" (cf, ~~Kothari~~ Bellman 1992).

This technique of dynamic programming has been applied to TSP (cf, Bellman 1962, Held and Karp 1962). Owing to its enormous storage requirements this technique 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 (cf, Srinivas 1989). Explain

2.2.1.3 BRANCH AND BOUND ALGORITHMS but exact

The Branch and Bound approach was developed by Little et al. (1963) in the first instance, in the context of the TSP. The technique involves a systematic search of the space of all feasible solutions. The basic approach followed in this algorithm 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 (or length) of the tour. The bounds guide the algorithm in deciding whether a given subset needs to be further scrutinized for optimality or whether it can be rejected for containing no solution better than the one at hand.

The subsets of tours are conveniently represented as the nodes of a tree and the process of partitioning as a branching of the tree. Hence this method is known as "Branch and Bound".

In fact in 1958, Eastman (cf, Eastman 1958) suggested one technique for solving the TSP, which is similar to the branch and bound approach. The lower bound considered was the optimal solution of the corresponding Assignment Problem.)As it is known that any solution to the TSP is also a solution of the Assignment Problem, so

which has lower min since less constraint the Assignment Problem is first solved allowing the sub-tours and then systematically sub-tours are eliminated until finally a tour is obtained that is optimal.

Depending on the procedure used for getting the bounds at each branch, many branch and bound algorithms were developed for instance by Little et al. (1963), Shapiro (1966), Christofides (1970), Bellmore and Mellone (1971), Smith et al. (1977). How good ?

2.2.1.4 LEXICOGRAPHIC SEARCH

This search (Lexi-search, for short) is a systematic branch and bound approach, developed by Pandit (1962), first for Loading Problem, before branch and bound approach of Little et al. (1963) was published. The approach may be summarized as follows (cf, Pandit 1963): The set of all possible 'solutions' to a combinatorial programming problem is arranged in a hierarchy- like words in a dictionary, such that each 'incomplete word' represents the block of words with this incomplete word as the 'leader'. Bounds are computed for the values of the objective function over these blocks of words. These are compared with the 'currently best available solution value'. If no word in the block can be better in value than the trial value (i.e., the value of best solution available so far), jump over the block to the next one. However, if the bound indicates a possibility of better solutions in the block, enter into the sub block by concatenating the present leader with appropriate 'letter' and set a bound for the new (sub) block so obtained. This procedure is very much like looking for a word in a dictionary; hence the name 'Lexi(cographic) Search'. The basic difference with the branch and bound approach of Little et al. (1963) is that lexi search approach is one-pass, implicitly exhaustive, search approach, avoiding the need for book-keeping involved in storing, in active memory, the bounds, at various branching nodes at various levels and related backtracking procedures, which can be expensive in terms of memory space and computing times. incomprehensible

Details of this approach are given in Chapter III.

TH 38

As the problem size increases, solving the TSP (and also any combinatorial optimization problem) exactly becomes impractical and so heuristic must be used. Also, it is to be noted that one can truncate the search by setting a % bound margin; setting this to k means that one winds up with a tour whose value is within k% of the exact optimal value. Such a solution is easily obtained by suitably modifying the '*bound*' in the exact search procedure. These methods of truncating the search are called Quasi-Exact (QE) methods. They give solution more quickly than that of the exact methods, but do not guarantee the optimality of the solution. Of course, they will guarantee that the solution is at most k% away from the exact optimal solution.

In this study, after improving the solution by lexi-search every time, the new total travel cost is further reduced by 15% (and also for all the problems considered in the present dissertation) and the 'current trial solution value' is replaced by the reduced travel cost. This way the upper bound is modified in the lexi-search approach. This modification will guarantee that the solution is at most 15% away from the exact optimal solution.

why 15%

2.2.2 APPROXIMATE SOLUTION METHODS (HEURISTICS)

These are the techniques, which seek good solutions (i.e., near optimal solutions) at a reasonable computational cost without being able to guarantee either optimality or even in many cases to state how close to optimality a particular solution is. Examples are Greedy algorithms, Artificial Neural Network (ANN), Genetic Algorithm (GA), Simulated Annealing (SA) etc.

2.2.2.1 GREEDY ALGORITHMS

The greedy algorithms are usually very fast and intuitively appealing, but they do not always work; one can construct examples to this effect. Some popular greedy algorithms for solving the TSP are nearest neighbor algorithm (cf, Bhatia and Rocha 1987) and k-Opt moves.

ENTRAL LIBRARY
CC. NO.............

(i) NEAREST NEIGHBOR ALGORITHM

The nearest neighbor algorithm works by selecting the locally superior alternative at each step. Applying it to the TSP produces the following procedure:

Step 1: - Select node '1' as the starting node (current node 'i=1').

Step 2: - To select the next node to be visited, look at all the 'legitimate' nodes (i.e., *the nodes that are not yet visited). Select the node 'j' such that the cost to go to* the node 'j' from node 'i' is the minimum. Go to the node 'j' next and then rename this node as node 'i'.

Step 3: - Repeat **step 2** until all nodes have been visited.

Step 4: - When all nodes have been added to the path, add an edge connecting the starting node to the last one and **stop**.

This greedy algorithm is easy to program. The worst feature of the algorithm is that its greed to pick up the cheap costs at the early and middle visits can force it to choose very expensive options in the last few visits. However from the empirical evidence, it is claimed that the solutions obtained by these algorithms are within 20% nearness of the optimal solution (cf, Bhatia and Rocha 1978, Lawler et al. 1985). It is typically a tour-construction heuristic. Many variants are described in Johnson (1990), Grendeau et al. (1992).

It may be noted here that the 'word-building' in the lexi search procedure (discussed later) also essentially follows this approach.

(ii) K-OPTIMAL MOVES

Using k-Opt moves, neighboring solutions can be obtained by deleting k edges from the current tour and reconnecting the resulting paths using k new edges. The k-Opt moves are the basis of the three most famous local search heuristics (greedy algorithms) for the TSP, namely, 2-Opt (cf, Croes 1958), 3-Opt (cf, Lin 1965, Stewart 1987) and Lin-Kernighan (LK) (cf, Lin and Kernighan 1973).

(a) 2-OPT MOVES

A neighboring solution is obtained from the current solution by removing two edges, replacing them by a different set of edges, in such a way as to maintain the feasibility of the tour.

Let α_i be the node in the position i of the tour, then if the edges (α_i, α_{i+1}) and (α_j, α_{j+1}) are removed, the only way to form a new valid tour is to connect α_i to α_j and α_{i+1} to α_{j+1}. Let us have a look on **Figure 2.1**, which represents a complete TSP tour $\{1 \rightarrow 3 \rightarrow 4 \rightarrow 2 \rightarrow 7 \rightarrow 6 \rightarrow 5 \rightarrow 8 \rightarrow 1\}$ of an 8-node problem.

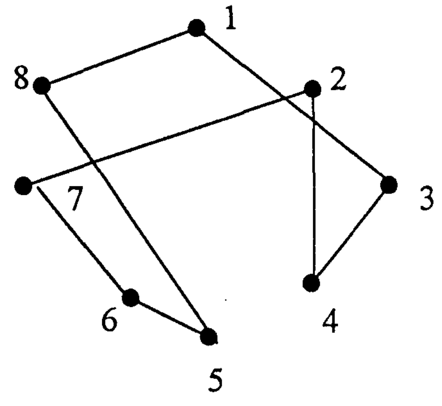

Figure 2.1:- An initial TSP tour.

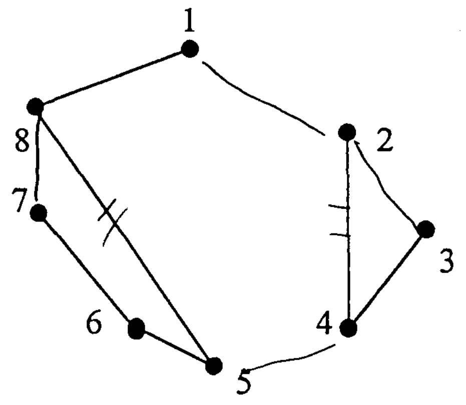

Figure 2.2:- After edge removal

NB Convex hull is optimal

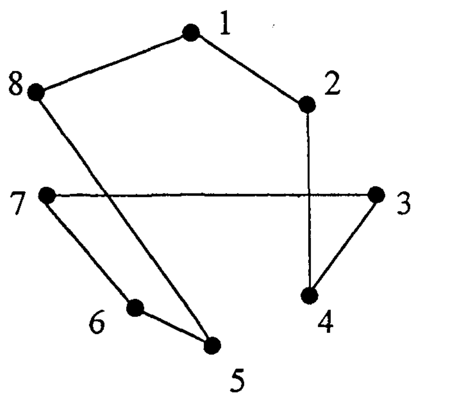

Figure 2.3: - Neighboring tour.

If we take i=1 and j=4, edge (1,3) and (2,7) would be removed as in **Figure 2.2.**The only way of reconnecting two edges to form a valid tour is to connect node 1 to node 2 and node 3 to node 7, while reversing the travel between node 2 and node 3, as in **Figure 2.3.** The size of neighborhood is n(n-1)/2 (in the case is of the symmetric TSP). A neighboring solution can be easily generated at random by generating randomly 'i' and 'j'. In order to check the preferability of the new tour, just to calculate

$$\{c(\alpha_i, \alpha_j) + c(\alpha_{i+1}, \alpha_{j+1})\}-\{c(\alpha_i, \alpha_{i+1}) + c(\alpha_j, \alpha_{j+1})\}.$$

If this has a negative value, the new tour is preferable to the current one. The worst case complexity for searching the neighborhood defined by this method is $O(n^2)$.

(b) **3-OPT MOVES**

In this case, three edges are deleted and replacing them by a different set of three edges, in such a way as to maintain the feasibility of the tour. **Figure 2.4** shows two possible 3-Opt moves that can be performed by deleting the edges (α_i, α_{i+1}), (α_j, α_{j+1}) and (α_k, α_{k+1}) of a tour.

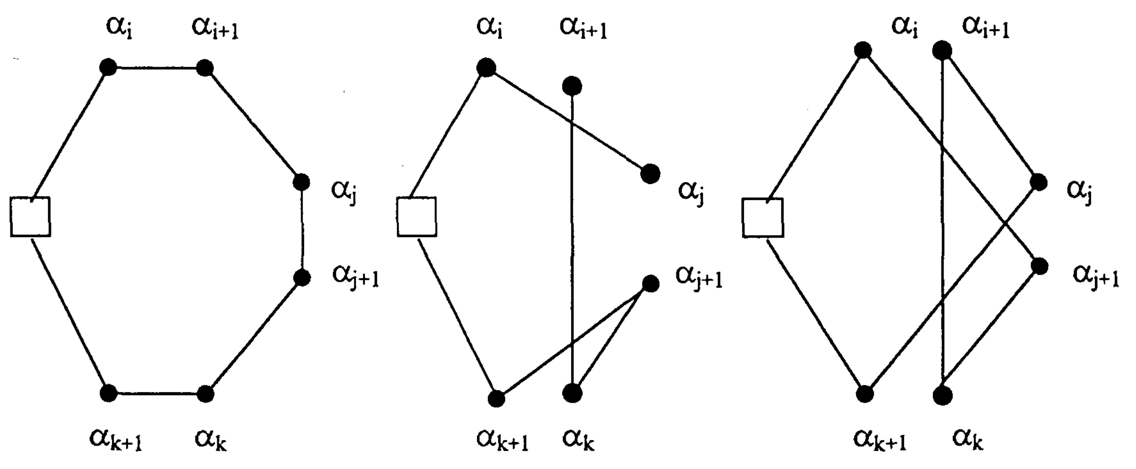

Figure 2.4. Two ways to perform 3-Opt.

There is one important difference between the two 3-Opt moves shown above: in the latter the orientation of the paths (α_{i+1},....., α_j) and (α_{j+1},....., α_k) is preserved, whereas in the former this orientation is reversed. 3-Opt is much more effective than 2-Opt, since the size of neighborhood is larger; but, for the same reason, it is more time consuming to search. The worst case complexity for searching the neighborhood defined by 3-Opt moves is $O(n^3)$.

An important modification of 3-Opt, introduced by Or (1976), can be implemented in $O(n^2)$ time. Or-Opt works by removing substrings of one, two, or three points from the tour, and reinserting them (possibly in reversed order) elsewhere such that the tour value of the new tour is better than that of initial (see **Figure 2.5**). The path (α_{i+1},......, α_j) is reinserted between α_k and α_{k+1}. No paths are reversed in this case.

[illegible]

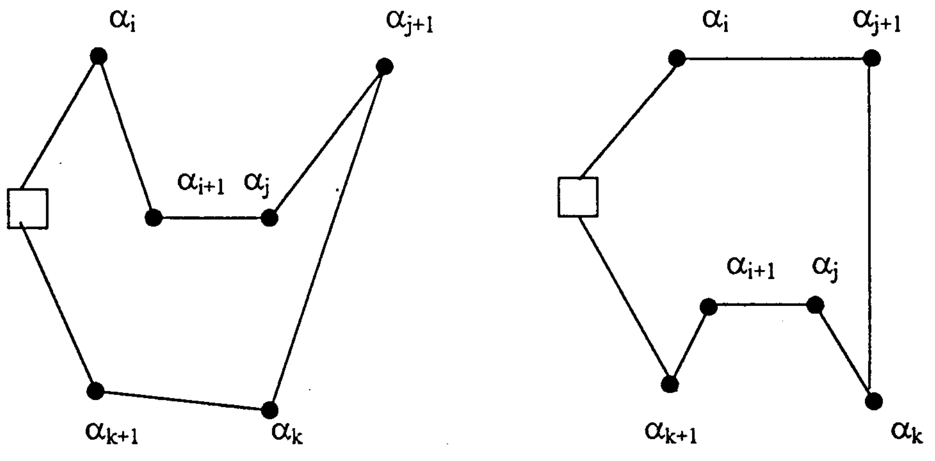

Figure 2.5. An Or-Opt move.

(c) LIN-KERNIGHAN PROCEDURE

To improve 3-Opt moves further Lin and Kernighan developed a sophisticated edge exchange procedure, where the number k of edges to be exchanged is variable (cf, Lin and Kernighan 1973). This algorithm is mentioned in the literature as Lin-Kernighan (LK) algorithm and it was considered for many years to be 'uncontested champion' of local search heuristics for the TSP. LK uses a very complex neighborhood structure, which we will briefly discuss here.

LK, instead of examining a particular 2-Opt or 3-Opt exchange, is building an exchange of variable size k by sequentially deleting and adding edges to the current tour while maintaining tour feasibility. Given node α_1 in tour T as a starting point. In step m of this sequential building of the exchange: edge (α_1, α_{2m}) is deleted, edge $(\alpha_{2m}, \alpha_{2m+1})$ is added, and then $(\alpha_{2m+1}, \alpha_{2m+2})$ is picked so that deleting edge $(\alpha_{2m+1}, \alpha_{2m+2})$ and joining edge $(\alpha_{2m+2}, \alpha_1)$ will close up the tour giving tour T_m. The edge $(\alpha_{2m+2}, \alpha_1)$ is deleted if and when step (m+1) is executed. The first three steps of this mechanism are illustrated in **Figure 2.6**.

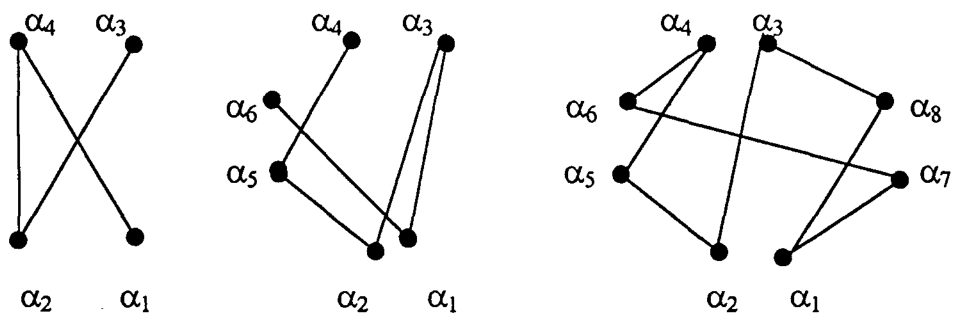

Figure 2.6 :-The first three steps of the Lin-Kernighan edge exchange mechanism.

Why we delete cross over ?

As we can see in this figure, the method is essentially executing a sequence of 2-Opt moves. The length of these sequences (i.e., depth of the search) is controlled by the LK's 'gain criterion', which limits the number of the sequences examined. In addition to that, limited backtracking is used to examine the sequences that can be generated if a number of different edges are selected for addition at steps 1 and 2 of the process.

The neighborhood structure described so far, although it provides the depth needed, is lacking breadth, and potentially missing improving 3-Opt moves. To gain breadth, LK temporarily allows tour infeasibility, examining so called 'infeasibility' moves which consider various choices for nodes α_4 to α_8 in the sequence generation process, examining all possible 3-Opt moves and more. **Figure 2.7** illustrates the infeasibility move mechanism. The worst case complexity for searching the LK neighborhood in $O(n^5)$.

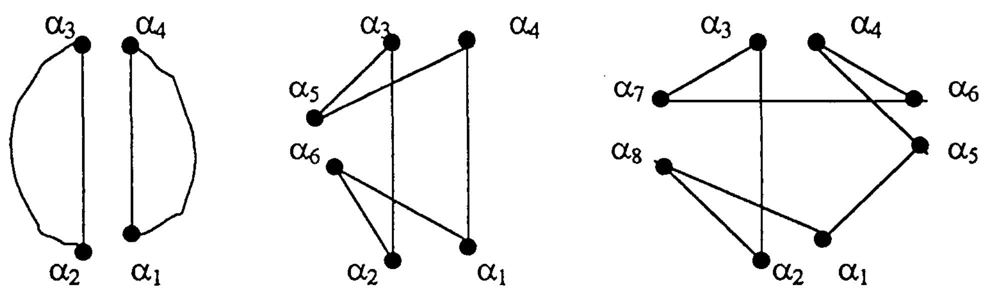

Figure 2.7:- Lin-Kernighan's infeasibility moves.

Implementations of 2-Opt, 3-Opt and LK-based local search methods may vary in performance. Martin et al. (1991, 1992) and Johnson (1990) have improved the above algorithms. These algorithms, along with divide and merge, comprise a family of similar heuristics. They work by first exchanging four old edges for four new edges, and then improving the resulting tour with local search optimizations. Johnson (1990) uses random cycle maintaining 4-exchanges, while Martin et al. (1991, 1992) use random 4-exchanges with a limit on the amount the tour's value can increase. Martin et al. have explored the use of both 3-Opt and LK algorithm for local optimization, while Johnson focuses on the LK algorithm. Zweig (1995) developed a divide and merge neighborhood technique, by dividing a cycle into two sub-cycles, reversing a sub-cycle and splicing two sub-cycles back together into a single cycle, and finally, performing modified Or-Opt and modified 2-Opt, in the neighborhood of all the newly formed edges. This form of divide and merge takes linear time per iteration. Zweig also described a restricted form of divide and merge that takes a constant time per iteration. Martin et al. (1991) reported running times averaging one hour on SPARC-1 for the 532-city instance and three hours for 783-city instance, while divide and merge of Zweig requires only about 42 and 90 SPARC-1 seconds respectively. But they did not report how far their solutions were from the exact optimals.

2.2.2.2 SIMULATED ANNEALING

Simulated Annealing (SA) was first proposed by Metropolis et al. (1953) for the efficient simulation of the evolution of a solid to a thermal equilibrium. After almost thirty years, Kirkpatrick et al (1983) realized that there exists a strong similarity between minimizing the cost function of a combinatorial optimization problem and slow cooling of a solid until it reaches its low energy ground state and that the optimization process can be realized by applying Metropolis criterion. By substituting cost function for energy of the solid and by executing the Metropolis algorithm at a sequence of slowly decreasing 'temperature' values, Kirkpatrick et al (1983) obtained a combinatorial optimization algorithm, which they called "Simulated Annealing". The algorithm is based on Monte-Carlo methods and can be considered as a special form of iterative improvement.

Annealing is a process of heating a solid and cooling it slowly so as to remove the strain and crystal imperfections. The SA procedure simulates this process of slow cooling of molten metal to achieve minimum function value in a minimization problem. Controlling a temperature- like parameter introduced with the concept of the Boltzmann probability distribution simulates the cooling phenomenon. According to the Boltzmann probability distribution, a system in thermal equilibrium at a temperature T has its energy distributed probabilistically according to

$$p(f) = \exp\left(-\frac{f(x)}{kT}\right),$$

where *f*(x) is a dynamic function and k is the Boltzman constant.

This expression suggests that a system at high temperature has almost uniform probability of being at any energy state, but at a low temperature, it has a small probability of being at high-energy state. So, by controlling the temperature T and assuming that the search process follows the Boltzmann probability distribution, the convergence of an algorithm is controlled.

Metropolis et al. set up a method, which simulates, for a fixed temperature T, the random evolution of a physical system in contact with a heat bath. The procedure is as follows (see also Deb 1995):

Step 1: - Choose an initial point x_i and a termination criterion ε. Set T a sufficiently high value, number of iterations to be performed at a particular temperature m, and i=0.

Step 2: - Calculate a neighboring point $x_{i+1}=N(x_i)$. A random point in the neighborhood is usually created.

Step 3: - If $f=f(x_{i+1})-f(x_i) < 0$, set i=i+1;

else create a random number (r) in the range (0,1).

If $r \leq \exp\left(-\frac{f}{T}\right)$, set i=i+1;

else go to **step 2**.

Step 4: - If $|x_{i+1}-x_i| < \varepsilon$ and T is small, then terminate;

else, if (i mod m) = 0, then lower T according to a cooling schedule and

go to **step 2**.

Else go to **step 2**.

The initial temperature (T) and the number of iterations (m) performed at a particular temperature are two important parameters, which govern the successful working of the SA procedure. A suitable value of m can be chosen (usually between 20 to 100, see Deb 1995) depending on the available computing resource and the solution time. The choice of the initial temperature and the subsequent cooling schedule still remain an art and usually require some trial and error efforts.

Lowering T ~ reducing tour length ?

Several Simulated Annealing for solving the TSP have been reported (cf, Johnson 1990, Sumana 1995). Sumana examined 24 different SA algorithms based on different cooling schedules. But the results showed that lexi search was better than SA.

2.2.2.3 GENETIC ALGORITHM APPROACH

Genetic Algorithms (GAs) first developed by John Holland (Holland 1975) are based essentially on 'mimicking' the 'survival of the fittest among the species' generated by random changes in the gene-structure of the 'chromosomes' in the evolutionary biology (cf, Goldberg, 1989).

In order to solve any real life problem by GA, two main requirements are to be satisfied:

(a) a (possibly binary) string can represent a solution of the solution space,

(b) an objective function and hence a fitness function which measures the goodness of a solution can be constructed / defined.

A simple GA works by randomly generating an initial population of strings which is referred as gene pool and then applying (possibly three) operators to create new, and hopefully, better populations as successive 'generations'. The first operator is 'reproduction' where strings are copied to the next generation with some probability based on their objective function value. The second operator is 'crossover' where randomly selected pairs of strings are 'mated', creating new strings. The third operator, 'mutation', is the occasional random alteration of the value at a string position. The 'crossover' operator together with 'reproduction' is the most powerful process in the GA search. Mutation diversifies the search space and protects from loss of genetic material that can be caused by reproduction and crossover.

Several genetic-based algorithms for solving the TSP were reported (cf, Goldberg 1989, Michalewicz 1994, Reeves 1993, Deb 1995). Details of this approach are given in Chapter IV.

2.3 SOME VARIATIONS OF THE USUAL TSP

Some of the many interesting variations of the TSP are given below:

2.3.1 THE TRUNCATED TRAVELLING SALESMAN PROBLEM

There are 'n' cities i=1,2,....,n and N={1,2,.....,n}. The distance d_{ij} between any pair of cities (i, j) is known. A subset 'O' of size less than n, of the cities constitutes the potential places for setting up a 'headquarters'. A salesman has to visit only 'm' (m<n) cities out of 'n' cities with the restriction that this tour should include at least one city from the subset O. The problem is to find a feasible tour of 'm' cities with a minimum length (cf, Sundara Murthy 1979).

2.3.2 THE PRECEDENCE CONSTRAINED TRAVELLING SALESMAN PROBLEM

Some ordered pairs of the cities are given. The salesman should visit the first city in each pair before the second one, not necessarily immediately (cf, Scroggs and Therp 1972, Das 1976, Bianco et al. 1994).

2.3.3 THE FIXED POSITIONAL TRAVELLING SALESMAN PROBLEM

Some cities are specified to be visited at specific steps from the 'headquarters' (cf, Scroggs and Therp 1972, Das 1976).

2.3.4 VEHICLE ROUTING PROBLEM

The 'n' nodes are indexed i=1,2,...,n and i=1 refers to the origin. There are 'm' vehicles, indexed k=1,2,....,m. Node 'i' has a demand q_i, the distance between 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 contain the origin. An assignment is a 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 minimum. The basic Vehicle Routing Problem is that of identifying the partition of nodes, which has the shortest distance covered by the vehicles (cf, Clerke and Wright 1964, Laporte et al. 1985).

2.3.5 THE CLUSTERED TRAVELLING SALESMAN PROBLEM

The set of nodes in the usual TSP is partitioned into k clusters. The problem is to find a least cost Hamiltonian tour such that the clusters are visited contiguously (cf, Chisman 1975, Lokin 1978). A particular type of this problem is called TSP with Backhauls (TSPB) in which the nodes of the network (except the starting node) are divided into linehaul and backhaul nodes. All the linehaul nodes should be visited contiguously after the starting node, followed by all the backhaul nodes.

One generalization of the TSPB is the Vehicle Routing Problem with Backhauls (VRPB) studied by Deif and Bodin (1984). In the VRPB, there is a non-negative demand q_i associated with each node i, and a fleet of k vehicles of capacity Q. The problem is to find a set of k vehicle routes of least total travel cost, in such a way that (1) each route starts and ends at the headquarters, (2) each node (except the headquarters), is visited exactly once by exactly one vehicle, (3) the total demand of any route does not exceed Q, and (4) on any route, all backhaul nodes are visited contiguously after all linehaul nodes. So, the TSPB may also be regarded as a subproblem of the VRPB, as each individual vehicle route is the solution of a TSPB.

2.3.6. "M" TRAVELLING SALESMAN PROBLEM

Let there are 'm' travelling salesman and we are given 'n' nodes with the specified direct travel costs c_{ij}, i,j=1,2,...,n, where node '1' is the 'headquarters'. Each of the salesmen is to start from node '1' and after touring his set of nodes should return to node '1'. The tours should have no common nodes (except node '1'). The problem is to determine the optimal tour plan, i.e., the sequence of nodes for each salesman, so that the total cost of the tour is minimum (cf, Frederickson et al. 1978, Ramesh 1997). It may be noted that this problem can be reduced to a single salesman problem by introducing a suitable number of dummy-origins (cf, Ramesh 1997).

2.3.7. THE TRAVELLER PURCHASER PROBLEM

There is a set I={1,2,...,m} of 'm' markets and a set K={1,2,.....,n} of 'n' commodities. Cost of a commodity $k \in K$ at market $i \in I$ is d_{ik} and the cost of travel from market 'i' to market 'j' is c_{ij}. The purchaser starts from home place (one market in I) and returns to it after purchasing all the commodities. The purchaser in his tour can visit a market, which already visited, and also he need not visit all the markets. The problem is to find a tour for the purchaser such that the total travel cost and purchase cost of all the commodities is minimum (cf, Ramesh 1980).

2.3.8 THE TIME CONSTRAINED TRAVELLING SALESMAN PROBLEM

The salesman of the usual TSP should start from node '1' at time '0' and must visit each node 'i' at time t_i, such that $e_i \leq t_i \leq l_i$, where $[e_i, l_i]$ is called time window of the node 'i'. The salesman is required to return to node '1' before time l_1. (cf, Baker 1983, Mingozzi et al. 1997).

CHAPTER III

THE LEXISEARCH APPROACHES FOR THE TSP

3.1 INTRODUCTION

The Lexicographic Search (Lexisearch, for short) approach to the Combinatorial Optimization Problem, as already mentioned in section 2.2.1.4, was first developed in the context of 'The Loading Problem" (cf, Gali 1960), popularly known as knapsack problem, and has since been applied to many combinatorial programming problems efficiently (cf, Pandit 1962), e.g., The Assignment Problem (cf, Pandit et al. 1964), The Travelling Salesman Problem (cf, Pandit 1964, Sundara Murthy 1979, Srinivas 1989), The facility Location Problem (cf, Das 1976, Ramesh 1997), Job Scheduling Problem (cf, Gupta 1969).

The Lexisearch derives its name from lexicography, the science of effective storage and retrieval of information. As already mentioned, the set of all possible solutions to a combinatorial programming problem is arranged in a hierarchy - like words in a dictionary, such that each 'incomplete word' represents the block of words with this incomplete word as the 'leader'. Each node is considered as a letter in an alphabet and each tour can be represented as a word with this alphabet. Thus the entire set of 'words' in this 'dictionary' (viz., the set of solution) is partitioned into 'blocks'. A block 'B' with a 'leader (α_1, α_2, α_3) of length three', consists of all the words beginning with (α_1, α_2, α_3) as the string of 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 sub-blocks. The block 'C' with leader (α_1, α_2, α_3,β) is a sub-block of 'B'. The block 'B' consists of many sub-blocks (α_1,α_2, α_3,β_i), one for each β_i. The block 'B' is the immediate super-block of block 'C'.

By the structure of the problem it is often possible to get bounds to the values of all words in a block (i.e., the 'bound for the block') by an examination of its leader. Hence, by comparing that bound with the value of a trial solution, one can

(i) go into the 'sub-blocks', if the block-bound is less than the trial solution value and hence, the current block may contain a solution better than the trial solution value,

(ii) jump over to the 'next' block; if the block-bound is greater than the trial solution value, or

(iii) jump out to the next 'super-block', if the current block, which is to be jumped over is the last block of the present superblock.

Further, if the value of the current leader is already greater than or equal to the 'current trial value'; no need for checking the subsequent blocks within this super-block. These concepts are illustrated below; in case of the TSP.

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

```
                ┌--abc----abcd
      ┌- ab-----┤
      │         └--abd-----abdc
      │         ┌--acb-----acbd
a-----┼--ac-----┤
      │         └--acd-----acdb
      │         ┌--adb-----adbc
      └--ad-----┤
                └--adc-----adcb
```

```
                 |--bac----bacd
       |--ba-----|
       |         |--bad-----badc
       |         |--bca-----bcad
b------|--bc-----|
       |         |--bcd-----bcda
       |         |--bda-----bdac
       |--bd-----|
                 |--bdc-----bdca

                 |--cab----cabd
       |--ca-----|
       |         |--cad-----cadb
       |         |--cba-----cbad
c------|-cb------|
       |         |--cbd-----cbda
       |         |--cda-----cdab
       |--cd-----|
                 |--cdb-----cdba

                 |--dab----dabc
       |--da-----|
       |         |--dac-----dacb
       |         |--dba-----dbac
d------|--db-----|
       |         |--dbc-----dbca
       |         |--dca-----dcab
       |--dc-----|
                 |--dcb-----dcba
```

The words starting with ‘a’ constitute a ‘block’ with ‘a’ as its leader. In a block, there can be many sub-blocks; for instance ‘ba’, ‘bc’ and ‘bd’ are leaders of the sub-blocks of block ‘b’. There could be blocks with only one word; for instance, the block with leader ‘abd’ has only one word ‘abdc’. All the incomplete words can be used as leaders to define blocks. For each of blocks with leader ‘ab’, ‘ac’ and ‘ad’, the block with leader ‘a’ is the immediate super-block.

There are mainly two ways of representing salesman's path in the context of Lexisearch approach, viz., path representation and adjacency representation. For example, let {1,2,3,4,5} be the labels of nodes in a 5 node TSP and let path to be represented be $\{1 \to 3 \to 4 \to 2 \to 5 \to 1\}$. Adjacency representation of this path is usual representation of corresponding permutation, viz., $\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 5 & 4 & 2 & 1 \end{pmatrix}$, indicating that the edges $1 \to 3$, $2 \to 5$, ..., constitute the tour. The path representation just lists the sequence of the tour as (1,3,4,2,5). The adjacency approach (where adjacency representation is considered for representing a tour) generates permutation in a systematic lexical order. Since all permutations $\begin{pmatrix} 1 & 2 & 3 \ldots\ldots n \\ \alpha_1 & \alpha_2 & \alpha_3 \ldots\ldots \alpha_n \end{pmatrix}$ do not lead to acceptable solution (i.e., path in a single cycle), a permutation is to be tested for acceptability- either at the formation stage itself- as each 'letter α_i' is added to the incomplete word, or it is to be tested at the end – once a full permutation is generated. Whichever is the better of these two is not easy to judge – it depends on the efficiency of the algorithm used to check premature cycle formation, but in this approach, the bound setting is computationally more efficient.

In the path approach (where path representation is considered for representing a tour), explicit testing for cycle formation is avoided, but bound setting is not efficient. In particular, a schema of path may occur at different relative positions in the path and thus, can lead to highly wasteful bound computation.

The relative efficiency of these two approaches is studied for the case of usual TSP in the following sections. However, for the major problems of the present thesis – i.e., TSP with Constraints, the path approach is a better approach, since the constraints can be checked during solution formation.

[illegible] when ?

3.2 ADJACENCY APPROACH AND ILLUSTRATION

Let us now illustrate the Lexisearch algorithm by outlines of an algorithm and an associated numerical problem's solution that follows, where adjacency representation is considered.

Pandit (1964) developed a Lexisearch algorithm where adjacency representation was considered for representing the salesman's tour. The algorithm (ALS) is as follows: -

Let C=[c_{ij}] be the given n x n cost matrix and c_{ij} be the cost of travelling ~~of~~ node j from node i, and let node '1' be the starting node (cf, **Table-3.1**).

Step 0: - - Remove the 'bias' of the given matrix. This bias removal 'reduces' the cost matrix to a non-negative matrix with at least one zero in each row and in each column (cf, **Table-3.~~3~~2**). Obviously, it is enough to solve the problem with respect to this cost matrix.//Sort in ascending order, this matrix row-wise. Store the corresponding column indices of each row of the matrix in a similar manner. Initialize the 'current trial solution value' to a large number.

Step 1: - With the partial word of length (l-1), take as leader (for the partial word to be filled) the first unchecked or free letter. Compute the bound by adding the costs of the remaining (n-l) successive letters after ensuring that the column repetition with each of the (l-1) letters of the partial word is avoided.

Step 2: - If the bound is less than the 'current trial solution value', go to **step 3**, *else* go to **step 5.**

Step 3: - If the column index of the leader tallies with the corresponding column index of any of the (l-1) letters of the partial word or if there is a sub-tour, go to **step 1**, *else* go to **step 4**.

Step 4: - Go to sub-block, i.e., augment the current leader, concatenate the first free letter to it, lengthening the leader by one letter and go to **step 1**.

Step 5: - Jump this block, i.e., decrement *l* by 1 (one), rejecting all the subsequent words from this block as the solution worse than the 'current trial solution value'. If *l*=1 and letter=n, go to **step 6**, *else* go to **step 1**.

Step 6: - Current word gives the optimum tour sequence, with 'current trial solution value' as the optimum cost, and go to **step 7**.

Step 7: - Add the 'bias' to the optimum solution and **stop**.

Working of this algorithm is explained through a seven-city example; with inter city travel costs as given in **Table-3.1** with bias of the matrix. **Table-3.2, Table-3.3** and **Table-3.4** give the reduced cost matrix, the 'alphabet table' and the 'search table' respectively. The symbols used therein (and also for all the problems in this present dissertation) are listed below:

GS: Go to sub-block, i.e., attach the first 'free' letter to the current leader. GS for 'db' leads to 'dba' as augmented leader.

JB: Jump the block, i.e., go to the next block of the same order i.e., replace the last letter of the current block by the letter next to it in the alphabet table. JB for 'abc' is 'abd'.

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

TRVL: Currently trial solution value.

CR: Column repetition.

TABLE-3.1: - The Cost Matrix

	1	2	3	4	5	6	7	Row Min
1	999	75	99	9	35	63	8	8
2	51	999	86	46	88	29	20	20
3	100	5	999	16	28	35	28	5
4	20	45	11	999	59	53	49	11
5	86	63	33	65	999	76	72	33
6	36	53	89	31	21	999	52	21
7	58	31	43	67	52	60	999	31
Col. Min	9	0	0	1	0	9	0	

Total Bias = row minima + column minima = 129 + 19 = 148.

TABLE-3.2: - The Reduced Cost Matrix

	1	2	3	4	5	6	7
1	982*	67	91	0	27	46	0
2	22	979*	66	25	68	0	0
3	86	0	994*	10	23	21	23
4	0	34	0	987*	48	33	38
5	44	30	0	31	966*	34	39
6	6	32	68	9	0	969*	31
7	18	0	12	35	21	29	968*

*The diagonal terms do not have to be computed. They could as well be left as 999.

TABLE-3.3: - The Alphabet Table

	*N-V	N-V	N-V	N-V	N-V	N-V	N-V
1	4-0	7-0	5-27	6-46	2-67	3-91	1-982
2	6-0	7-0	1-22	4-25	3-66	5-68	2-979
3	2-0	4-10	6-21	5-23	7-23	1-86	3-994
4	1-0	3-0	6-33	2-34	7-38	5-48	4-987
5	3-0	2-30	4-31	6-34	7-39	1-44	5-966
6	5-0	1-6	4-9	7-31	2-32	3-68	6-969
7	2-0	3-12	1-18	5-21	6-29	4-35	7-968

*Note:- N=Node number, V=Value of the node.

Table-3.4: - Search Table

Leaders							*Bounds L-1 + N-L Value	Trial Value	Remarks
1	2	3	4	5	6	7			
4-0							0+0	9999	GS
	6-0						0+0	9999	GS
		2-0					0+12	9999	GS
			1-0				0+12	9999	CR
			3-0				0+57	9999	GS
				7-39			0+18	9999	GS
					5-0		39+18	9999	GS
						1-18	39+0	9999	GS
							TRVL=	57	JB,JO
					1-6		39+21	57	JB,JO
				1-44			0+21	57	JB,JO
			7-38				0+12	57	GS
				3-0			38+18	57	GS
					5-0		38+18	57	CR
					1-6		38+21	57	JB,JO
				1-44			38+12	57	JB,JO
			5-48				0+18	57	JB,JO
		5-23					0+6	57	GS
			1-0				23+31	57	CR
			3-0				23+36	57	JB
			2-34				23+6	57	JO
		7-23					0+0	57	GS
			1-0				23+0	57	CR
			3-0				23+30	57	GS
				2-30			23+18	57	JB
				1-44			23+0	57	JO
			2-34				23+12	57	JO
		1-86					0+0	57	JO
	7-0						0+0	57	GS
		2-0					0+12	57	GS
			1-0				0+12	57	CR
			3-0				0+62	57	JB
			6-33				0+12	57	GS
				3-0			33+18	57	GS
					5-0		33+18	57	GS
						1-18	33+0	57	GS
							TRVL=	51	JB,JO

*Note: -Here *l* is the length of the present (incomplete) tour. Total bound value is the sum of leader value + cumulative value of (*l*-1) letters + value of remaining possible (n-*l*) letters.

Table-3.4: - Search Table (Continued)

Leaders							Bounds L-1 + N-L value	Trial Value	Remarks
1	2	3	4	5	6	7			
					1-6		33+21	51	CR,JO
				1-44			33+12	51	JO
			5-48				0+18	51	JB,JO
		6-21					0+0	51	GS
			1-0				21+0	51	CR
			3-0				21+30	51	JB
			2-34				21+12	51	JO
		5-23					0+6	51	GS
			1-0				23+32	51	CR
			3-0				23+36	51	JB
			6-33				23+6	51	JO
		1-86					0+0	51	JO
	1-22						0+0	51	GS
		2-0					22+12	51	GS
			3-0				22+55	51	CR
			6-33				22+12	51	JO
		6-21					22+0	51	GS
			3-0				43+30	51	JB
			2-34				43+12	51	CR,JO
		5-23					22+31	51	JB
		7-23					22+0	51	GS
			3-0				45+30	51	JB
			6-33				45+0	51	JO
		1-86					22+0	51	CR,JO
	3-66						0+30	51	JO
7-0							0+0	51	GS
	6-0						0+0	51	GS
		2-0					0+12	51	GS
			1-0				0+12	51	GS
				3-0			0+21	51	GS
					5-0		0+35	51	CR,JO
				4-31			0+12	51	GS
					5-0		31+12	51	GS
						1-18	31+0	51	CR,JO
					1-6		31+21	51	JB,JO
			3-0				0+49	51	GS
				4-31			0+18	51	GS
					5-0		31+18	51	CR
					1-6		31+21	51	JB,JO

Table-3.4: - Search Table (Continued)

Leaders							Bounds	Trial	
1	2	3	4	5	6	7	L-1+ N-L value	Value	Remarks
				1-44			0+21	51	JB,JO
			5-48				0+18	51	JB,JO
		4-10					0+0	51	GS
			1-0				10+0	51	GS
				3-0			10+0	51	GS
					5-0		10+0	51	GS
						2-0	10+0	51	GS
							TRVL=	10	JB,JO
					2-32		10+21	10	CR,JO
				2-30			10+12	10	JO
			3-0				10+30	10	JO
		6-21					0+0	10	JO
	1-22						0+0	10	JO
5-27							0+0	10	STOP

As seen from the above search table, the optimal solution is given by the permutation $\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 7 & 6 & 4 & 1 & 3 & 5 & 2 \end{pmatrix}$ or equivalently the tour is $\{1\to7\to2\to6\to5\to3\to4\to1\}$, and optimal solution = bias + trial value = 148 + 10 = 158.

As it is already mentioned that in this adjacency approach a permutation is to be tested for acceptability – either at the tour formation stage or at the end – once a full permutation is generated. In the tour formation stage, if the *incomplete* permutation is $\begin{pmatrix} 1 & 2 & 3......i \\ \alpha_1 & \alpha_2 & \alpha_3....\alpha_i \end{pmatrix}$, i.e., if letter '$\alpha_i$' is added to the incomplete word, and if $\alpha_i \prec i$, then there is a possibility of a sub tour. So, check it, else need not have to check. In the other approach, the permutation is checked only after a full permutation is obtained. We have employed both the approaches of checking the cycle. So, let us have a comparative study of these two sub-tour checking Lexi search program. For simplicity, the approach involving cycle checking while building the tour is named as STI, and the other is named as STA. The comparative study is carried out for two sets of problems of sizes 25 and 30 (see in **Table-3.5**).

Table-3.5: - Comparative study of STI, STA and STC.

N	Seed	Sol	Time			N	Seed	Sol.	Time		
			STI	STA	STC				STI	STA	STC
	930	152	0.27	1.32	1.32		**930**	156	3.85	>60	>60
	160	192	0.66	0.82	0.99		**160**	184	0.55	>60	>60
25	**681**	154	0.17	1.21	0.22	**30**	**681**	166	0.82	10.60	2.25
	418	167	0.22	0.28	0.27		**418**	187	5.11	>60	>60
	522	164	0.11	13.01	11.70		**522**	199	2.97	>60	>60
	Mean		**0.29**	**3.33**	**2.90**		**Mean**		**2.66**	-----	----
	S.D.		**0.19**	**4.85**	**4.42**		**S.D.**		**1.75**		

From **Table-3.5**, it is seen that STI approach is better than the STA approach. Through theoretically, the sub-tour checking after completing a full permutation is less costly than the other, but it is seen that the cost of throwing a complete permutation, as it is not a feasible solution, is much more, which leads STA to an inefficient approach. To reduce the throwing infeasible complete permutation, we allow some check post in the entire tour, and between two check posts we concatenate the 'first free' letter to the current leader without calculating bound and checking the occurrence of a sub-tour. Only when we reach the check post (i.e., length of the present tour equals the value of the check post), the bound of the present tour is computed and check for cycle formation. In this present study we consider the check posts at the length of $\left\lfloor \frac{n}{5} \right\rfloor, \left\lfloor \frac{2n}{5} \right\rfloor, \left\lfloor \frac{3n}{5} \right\rfloor, \left\lfloor \frac{4n}{5} \right\rfloor$ *and* n. But, still this approach (which is named as STC for simplicity) is worse than the STI (as shown in**Table-3.5**). Hence, we accept the STI approach (see **Program 1.2 in Appendix**) for sub-tour checking.

Sundara Murthy (1979) proposed another scheme for the 'search' sequence of Pandit's Lexisearch algorithm. He arranged 'n(n-1)' direct distances in ascending order and these corresponding 'ordered pairs- i,j' are the words in a dictionary. When a leader is specified, he checked for

(i) non-repetition and

(ii) non-cycling in the leader **after considering** the bounds for block values.

Thus, once a word is accepted as possibly containing a better solution within the corresponding block, he checked for 'row' and 'column' repetitions of the letter and also checked for sub-tour formations. This was expected to increase the computational efficiency of the algorithm to considerable extent.

But, the proposed algorithm is likely to be efficient in case of highly skewed distance distributions and does not seem to be any better than the conventional branch and bound algorithms (cf, Srinivas 1989). So, Srinivas (1989) again modified Pandit's Lexisearch algorithm, which is named as "Data-Guided Lexisearch Algorithm". Here the nodes of the network are renamed on the basis of a scrutiny of the cost matrix and an 'alphabet' table is defined after the above pre-processing.

In the data-guided lexisearch (DGLS) algorithm the following steps of ALS algorithms are modified.

Step 0: - Remove the 'bias' of the given cost matrix. Interchange the rows so that the rows with maximum zeros are shifted to the bottom while rows with minimum zeros are shifted to the top. In the even of a tie, the tie is broken by comparing the values of the smallest non-zero elements in the two rows: the row with the larger value precedes the other. Sort in ascending order, this matrix row-wise and store the corresponding column indices of each row of the matrix in a similar manner. Initialize the 'current trial solution value' to a large number.

Table-3.7: - Search Table

Leaders							Bounds L-1 + N-L value	Trial Value	Remarks
1(5)	2(7)	3(3)	4(6)	5(4)	6(1)	7(2)			
3-0							0 + 0	9999	GS
	2-0						0 + 10	9999	GS
		2-0					0 + 0	9999	CR
		4-10					0 + 0	9999	GS
			5-0				10 + 0	9999	GS
				1-0			10 + 0	9999	GS
					4-0		10 + 0	9999	CR
					7-0		10 + 0	9999	GS
						6-0	10 + 0	9999	GS
							TRVL=	10	JB,JO
					5-27		10 + 0	10	JO
				3-0			10 + 0	10	CR
				6-33			10 + 0	10	JO
			1-6				10 + 0	10	JO
		6-21					0 + 0	10	JO
	3-12						0 + 0	10	JO
2-30							0 + 0	10	STOP

As seen from the above search table, the optimal solution is given by the permutation $\begin{pmatrix}1,2,3,4,5,6,7\\7,6,4,1,3,5,2\end{pmatrix}$ or equivalently the tour is $\{1\to7\to2\to6\to5\to3\to4\to1\}$, and optimal solution = bias + trial value = 148 + 10 = 158.

Apart from the advantage of minimal memory requirement, the Lexisearch, in general and the Data-Guided Lexisearch, in particular, is far superior than that of Branch and Bound approach of Little et al. (1963) (cf, Srinivas 1989).

As it is mentioned that for the TSP with constraints, the path approach is better than the adjacency approach. So, let us describe the path approach.

3.3 PATH APPROACH AND ILLUSTRATION

Let us first illustrate the Lexisearch algorithm by outlines of an algorithm and an associated numerical problem's solution that follows, where path representation is considered (also see Ramesh 1997).

Now for the algorithm (PLS) and illustration:

Let $C=[c_{ij}]$ be the given n x n cost matrix and c_{ij} be the cost of travelling of node j from node i, and let node '1' be the starting node (cf, **Table-3.1**). Then for all nodes in the network, generate a zero-one vector V of order n as follows:

$v_i=1$; if node 'i' is in the tour.

$=0$; otherwise.

Though this is not a part of the program (or algorithm), but for checking the feasibility of the tour, we follow this convention.

Step 0: - In addition to the **step 0** of ALS, in section 3.2, we put r=1.

Step 1: - Go to the rth element of the row (say, node α) and compute the cost of travelling. If the travel cost is greater than or equal to the 'current trial solution value', go to **step 8**, *else,* go to **step 2**.

Step 2: - If the (incomplete) word forms a sub-tour, drop the city added in **step 1** and increment r by 1, and then go to **step 6**; *else,* go to **step 3**.

Step 3: - If all the nodes of the network is visited, add an edge connecting the starting node to the last one and compute the travel cost and go to **step 4**, *else* go to **step 5.**

Step 4: - If the travel cost is greater than or equal to the 'current trial solution value', go to **step 9**, *else,* replace the travel cost as the 'current trial solution value' and go to **step 9**.

Step 5: - Calculate the Bound.

Step 6: - If the (Bound + Travel cost) is greater than or equal to the 'current trial solution value', drop the city added in **step 1** and increment r by 1, and then go to **step** 7; *else,* go to **step 8**.

Step 7: - If r is less than n (total number of nodes), go to **step 1,** *else,* go to **step 8**.

Step 8: - Go to sub-block, i.e., go to α^{th} row and then put r=1; go to **step 1**.

Step 9: - Jump this block, i.e., go to the previous row (i.e., node) and increment r by 1, where r was the column number of that row. This will automatically [illegible]bsequent words from this block as solutions worse (at least, not better) than the current trial value. If the present node is the starting node and r =n, go to **step 10,** *else,* go to **step 1**.

Step 10: - Current word gives the optimal tour sequence, with 'current trial solution value' as the optimum cost, with respect to the 'reduced' cost matrix.

Step 11: - Add the 'bias' to the optimal solution value obtained above and **stop**.

3.3.1 BOUND CALCULATION

It is already mentioned that the bound setting in this path approach is not efficient and getting very efficient lower bound is computationally very costly. However, following bound setting technique is used for the usual TSP.

Bound is the sum of travel costs of the rows (cities) (which ~~is~~ are not in the word, excluding latest city) to the first reachable city (excluding latest city) within the first (β-1) cities if any; otherwise take the cost of travelling the β^{th} city in that row. The larger value of β, the better (more 'efficient') is the bound obtained but the computations also will be higher, while smaller β requires less computations for bound setting 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. The value of β(≤n) we have considered in this present study as $\left\lfloor \frac{n}{5} \right\rfloor$ (and also for all of the problems considered in this present dissertation).

For the problems with constraints also, this bound calculation, without considering the constraints, is considered in developing path approach in the context of lexisearch approach.

3.3.2. ILLUSTRATION

Working of this algorithm is explained through the same problem as given in **Table-3.1.** The logic-flow of the algorithm at various stages is indicated in **Table-3.8,** which sequentially records the intermediate results, with decision taken (i.e., remarks) at these steps in every column.

As the illustration of the example, we first set 'trial solution value' as 9999. Since our starting node (or city) is '1', so we start from 1st row of the 'alphabet table' and select the 1st node, which is not present in the partial tour (i.e., the tour, which is not yet completed). In this table, node 4 is the acceptable node with value 0. Since the total partial tour value, if the node 4 is accepted, is less than the 'trial solution value', so we can go for bound calculation. The bound will guide us whether this node 4 will be accepted or not. If it is accepted, then go to the next step of searching, otherwise, leave this node and jump out to the next block of same size. Since node 1 is only present in the actual partial tour and node 4 is the latest node, so sequentially search all the first nodes, except node 1 and node 4, of the rows expect 1st row, and add the corresponding node values. So, in 2nd and 3rd rows we can visit node 6 with value 0 and node 2 with value 0 respectively. In 4th row we can visit node 1 with value 0, which is allowed in bound calculating as at last we have to come back to the starting node 1, though it is already present in the partial tour.

TABLE-3.8: SEARCH TABLE

$1\rightarrow\alpha_1$	$\alpha_1\rightarrow\alpha_2$	$\alpha_2\rightarrow\alpha_3$	$\alpha_3\rightarrow\alpha_4$	$\alpha_4\rightarrow\alpha_5$	$\alpha_5\rightarrow\alpha_6$	$\alpha_6\rightarrow 1$
$1\rightarrow 4_{(0)}$ (0)+0,GS	$4\rightarrow 3_{(0)}$ (0)+30,GS	$3\rightarrow 2_{(0)}$ (0)+52,GS	$2\rightarrow 6_{(0)}$ (0)+57,GS	$6\rightarrow 5_{(0)}$ (0)+57,GS	$5\rightarrow 7_{(39)}$ (39)+18,GS	$7\rightarrow 1_{(18)}$ TRVL=57 JO
				$6\rightarrow 7_{(31)}$ (31)+62,JO		
			$2\rightarrow 7_{(0)}$ (0)+52,GS	$7\rightarrow 5_{(21)}$ (5)+40,JB		
				$7\rightarrow 6_{(29)}$ (29)+44,JO		
			$2\rightarrow 5_{(68)}$, JO			
		$3\rightarrow 6_{(21)}$ (21)+30,GS	$6\rightarrow 5_{(0)}$ (21)+39,JB			
			$6\rightarrow 7_{(31)}$ (52)+61,JO			
		$3\rightarrow 5_{(23)}$ (23)+40,JB				
		$3\rightarrow 7_{(23)}$ (23)+34,JO				
	$4\rightarrow 6_{(33)}$ (33)+0,GS	$6\rightarrow 5_{(0)}$ (33)+0,GS	$5\rightarrow 3_{(0)}$ (33)+0,GS	$3\rightarrow 2_{(0)}$ (33)+18,GS	$2\rightarrow 7_{(0)}$ (33)+18,GS	$7\rightarrow 1_{(18)}$ TRVL=51 JO
				$3\rightarrow 7_{(23)}$,JO		
			$5\rightarrow 2_{(30)}$,JO			
		$6\rightarrow 7_{(31)}$,JO				
	$4\rightarrow 2_{(34)}$ (34)+33,JB	$7\rightarrow 2_{(0)}$ (38)+21,JB				
	$4\rightarrow 7_{(38)}$ (38)+0,GS	$7\rightarrow 3_{(12)}$ (50)+30,JB				
		$7\rightarrow 5_{(21)}$,JO				
	$4\rightarrow 5_{(48)}$ (48)+6,JO					
$1\rightarrow 7_{(0)}$ (0)+0,GS	$7\rightarrow 2_{(0)}$ (0)+10,GS	$2\rightarrow 6_{(0)}$ (0)+10,GS	$6\rightarrow 5_{(0)}$ (0)+10,GS	$5\rightarrow 3_{(0)}$ (0)+10,GS	$3\rightarrow 4_{(10)}$ (10)+0,GS	$4\rightarrow 1_{(0)}$ TRVL=10 JO
				$5\rightarrow 4_{(31)}$,JO		
			$6\rightarrow 4_{(9)}$ (9)+23,JB			
			$6\rightarrow 3_{(38)}$,JO			
		$2\rightarrow 4_{(25)}$,JO				
	$7\rightarrow 3_{(12)}$,JO					
$1\rightarrow 5_{(27)}$ STOP						

Next, in 5th, 6th and 7th rows the nodes 3 with value 0, 5 with value 0 and 2 with value 0 can be visited respectively. So, the bound is the 0+0+0+0+0+0=0 and the total bound is the addition of this with the present tour value, hence 0+0=0. Since this total bound is less than the 'trial solution value', so we can accept the node 4, which leads to the partial tour $\{1 \rightarrow 4\}$ with value 0. Now, go to the 4th row, as the present node is 4, and visit the 1st untouched node 3 with value 0, as node 1 is present in the partial tour and tour is not complete. This node value is added to the present tour value, and as this total is less than the 'trial solution value', hence we can go for bound calculation. For bound calculation, in 2nd, 3rd, 5th, 6th and 7th rows, the nodes 6 with value 0, 2 with value 0, 2 with value 30, 5 with value 0 and node 2 with value 0 can be visited respectively. It is to be noted that since this problem is of small size, so we take $\beta = n-1$, i.e., we search for the legitimate node in a row upto (n-1)th element for the bound calculation. In 5th row, the 1st node was 3, but as it is the latest node, we can't visit it again. So, the bound will be 30. Since the addition of this bound and the present tour value is less than the 'trial solution value', hence we can accept the node 3. Thus, the partial tour will be $\{1 \rightarrow 4 \rightarrow 3\}$ with value 0. Proceeding in this way we obtain the 1st complete tour as $\{1 \rightarrow 4 \rightarrow 3 \rightarrow 2 \rightarrow 6 \rightarrow 5 \rightarrow 7 \rightarrow 1\}$ with value 57, and as this value is less than the 'trial solution value', replace the 'trial solution value' by this value. Now, we jump out to the next higher order block, i.e., $\{1 \rightarrow 4 \rightarrow 3 \rightarrow 2 \rightarrow 6 \rightarrow 5 \rightarrow 7\}$ and go to complete the tour. Since to complete the tour no option except node 1 is left and we have already searched it, hence we jump out to the next higher order block, i.e., $\{1 \rightarrow 4 \rightarrow 3 \rightarrow 2 \rightarrow 6 \rightarrow 5\}$. Proceeding in this way, we obtain the final tour as $\{1 \rightarrow 7 \rightarrow 2 \rightarrow 6 \rightarrow 5 \rightarrow 3 \rightarrow 4 \rightarrow 1\}$ with value 10. And hence the optimal tour value with respect to the given cost matrix = bias + trial value = 148 + 10 = 158.

It is interesting to see that the problem with 'bias' (PLSB) takes more time than that of the same problem without 'bias' (PLS). Let us have a look at the comparative study of the programs in **Table-3.9**. Relative efficiency analysis is carried out for two sets of randomly generated problems of sizes 20 and 25. Each set contains 5 randomly generated problems. Later on we will apply the genetic algorithms also to the reduced cost matrix after removal of 'bias'.

Table-3.9: - A Comparative study of PLSB and PLS programs.

Seed	N=20			N=25		
	Soln.	Time		Soln.	Time	
		PLS	PLSB		PLS	PLSB
930	137	0.00	0.11	152	0.22	4.83
160	163	0.00	0.27	192	0.72	217.78
681	162	0.05	0.38	154	0.16	12.86
418	180	0.06	0.28	167	0.22	15.48
522	178	0.00	0.27	164	0.11	4.56
Mean		**0.02**	**0.26**		**0.29**	**51.10**
S.D.		**0.03**	**0.09**		**0.22**	**83.45**

Now, as it is already mentioned that in path approach bound setting is very costly, whereas in adjacency approach the sub-tour checking is costly. So, let us have a comparative study of both the approaches. Comparative study of PLS, ALS and DGLS algorithms is carried out for four sets of randomly generated test problems of sizes 34, 36, 39 and 40 (see **Table-3.10**).

Table-3.10: - A comparative study of PLS, ALS and DGLS for the Usual TSP.

N	Seed	Sol	Time			N	Seed	Sol.	Time		
			PLS	ALS	DGLS				PLS	ALS	DGLS
	930	174	7.36	1.31	0.60		**930**	168	30.65	5.87	0.27
	160	151	13.73	2.26	9.83	**36**	**160**	186	1233.96	50.75	31.53
34	**681**	209	46.85	167.35	1.38		**681**	168	4.34	1.21	1.43
	418	153	8.68	6.82	1.48		**418**	156	97.65	57.45	4.67
	522	179	9.56	6.48	16.09		**522**	205	65.64	274.52	190.42
	Mean		**17.24**	**36.84**	**5.88**		**Mean**		**286.45**	**77.96**	**45.66**
	S.D.		**14.96**	**65.29**	**6.12**		**S.D.**		**474.81**	**100.88**	**73.28**
	930	188	495.93	60.91	71.79		**930**	154	119.52	215.80	25.60
	160	150	39.49	4.07	17.41	**40**	**160**	168	796.03	94.86	159.34
39	**681**	158	121.16	0.82	14.06		**681**	166	539.76	81.95	67.83
	418	157	161.76	17.52	152.75		**418**	180	1310.14	1245.10	7.25
	522	185	20.10	45.10	4.12		**522**	192	121.06	126.94	32.24
	Mean		**167.69**	**25.68**	**52.03**		**Mean**		**577.30**	**352.93**	**58.45**
	S.D.		**172.14**	**23.54**	**55.62**		**S.D.**		**448.11**	**448.53**	**54.14**

It is seen from the **Table-3.10** that as the size of the problem increases PLS algorithm takes more time than that of ALS algorithm. Of course, for few of the problems, ALS takes more time than that of PLS. As example, for the 3rd problem of size 34, PLS takes only 46.85 seconds, whereas ALS takes 167.35 seconds. Now, let us compare the ALS algorithm with DGLS algorithm. It is seen for most of the problems that DGLS is better than ALS. Of course here also, for few problems ALS takes less time than DGLS. For example, for the 4th problem of size 39 ALS takes only 17.52 seconds, whereas DGLS takes 152.75 seconds. The DGLS is the modification of ALS, and it comprises of two stages – preliminary scrutiny and preprocessing, in addition to the stages in ALS. So, the actual building of full-fledged data-guided algorithm is not presented. Of course, if one considers the mean and standard deviation of the times, then one can conclude that DGLS algorithm is the best among these exact algorithms. So, the DGLS approach is used for getting exact optimal solution and then quasi-exact optimal solution for the usual TSP.

The DGLS approach is one of the best exact methods for the usual TSP. However, when the size of the problem increases, the computational time increases exponentially; very fast. Hence our aim is to find heuristically optimal solution to the TSP by Quasi-Exact method, Sequential Constructive Sampling approach, and Genetic Algorithms (GAs) and then establishing the 'degree of goodness' of these heuristic methods by comparing them with the solutions obtained by the DGLS.

3.4. SEQUENTIAL CONSTRUCTIVE SAMPLING APPROACH AND ILLUSTRATION

This is a sampling approach to 'guess' or 'estimate' the best tour. It can be described as a statistical version of the lexisearch approach; it combines the lexisearch approach with the sequential sampling approach for estimating the minimum of a statistical population through adaptive sampling, with reasonable stopping rule.

It is not a simple random generation of tours. The 'alphabet table' of the lexisearch approach for the TSP is considered, and instead of visiting the first 'legitimate' node of a row from its corresponding row (node), one of the 'legitimate' nodes is visited probabilistically. For this the probability of visiting each 'legitimate' node is assigned in such a way that the first legitimate node gets more probability than second one, and so on. The probability of visiting each 'legitimate' node is assigned as follows. Suppose the number of 'legitimate' nodes is k. The probability of visiting ith 'legitimate' node is

$$p_i = \frac{2(k-i+1)}{k(k+1)} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad 3.1$$

Why?

Thereafter, the cumulative probability (P_i) of each node being visited can be calculated by adding the individual probabilities from the left of the list. So, P_0=0 and P_k=1. Now, a random number between 0 to 1 is generated and the node that represents the chosen random number in the cumulative probability range for that node is accepted for the next stage of testing (i.e., bound calculation).

The procedure may be summarized as follows:

Step 0: - Construct the 'alphabet table' based on the given cost matrix (i.e., without bias removing). Initialize the 'current trial solution value' to a large number.

Step 1: - Start from 1st row (node 1) (i.e., current node i=1).

Step 2: - Visit probabilistically jth node of the ith row (as described above). That is going to the node 'j' next.

Step 3: - Compute the (incomplete) travels cost. If this travel cost is already greater than or equal to the 'current trial solution value', then go to **step 6** and then try to generate another tour; *else* go to **step 4**.

Step 4: - Compute the Bound for the 'residual part of the tour'.

Step 5: - If (incomplete travel cost + bound) is greater than or equal to the 'current trial solution value', then go to **step 6,** *else* got to **step 7.**

Step 6: - Repeat **step 2** to **step 5** at most k times, where k is number of 'legitimate' nodes. If within this many trials no improvement is possible then go to **step 1** and then try to generate another tour; *else,* go to **step 7.**

Step 7: - If all the nodes have been added to the path, add an edge connecting the starting node to the last one. If the total travel cost is less than the 'current trial solution value', then replace the 'current trial solution value' by the travel cost, *else* go to **step 1.** Else rename the node 'j' as node 'i', go to **step 2.**

When the number of (incomplete or complete) tours generated is 'large enough', **stop** sampling and accept the 'current trial solution value' as the estimate of the best solution value and the corresponding tour as the best tour.

Obviously, the crucial question is, how large is 'large enough'?

No exact theoretical answer to this general question is statistical inference seems to be available. The stopping rule adopted depend essentially on the judgment of the solution-seeker and are often determined by the 'time and computational effort' incurred so far. They may also be guided by 'experience gained by simulational studies'.

In this present study (also for all of the problems in this present dissertation) we have considered a sample of size $5n^3$, i.e., the best solution value obtained within this $5n^3$ trials is considered as the best solution value of the problem.

3.4.1 BOUND CALCULATION

Arrange the entire matrix elements in ascending order and store them in an array, say D, also their corresponding row and column indices are stored in other two arrays, say R and K respectively. Also the cumulative sums of the elements in D are stored in an array, say CD. Let the length of the partial array be l, so n-l+1 steps are still required. Then check the elements (nodes) in R sequentially, whether they are already in the partial word (tour) (except the latest node). If one of them is not in the partial word, then check the corresponding element (node) in K, whether it is already present in the partial word. If no and the element is the i^{th} element in K, then the bound will be CD(n-l+i)-CD(i-1), else check the next element of R, else check within first $(m-1)^{th}$ elements in the array R, otherwise take the bound as

CD(n-l+m)-CD(m-1).

In this present study (also for all of the problems studied in this present dissertation) we take m=10. This bound calculation technique, without introducing constraints, is considered for the other Restricted TSPs also.

3.4.2 ILLUSTRATION

Let us illustrate the process through an example of seven-city problem given in **Table-3.1**. The 'alphabet tables' for building the tour and calculating the bound are given in **Table-3.11** and **Table-3.12** respectively.

TABLE-3.11: - The Alphabet Table

	N-V	N-V	N-V	N-V	N-V	N-V	N-V
1	7-8	4-9	5-35	6-63	2-75	3-99	1-999
2	7-20	6-29	4-46	1-51	3-86	5-88	2-999
3	2-5	4-16	5-28	7-28	6-35	1-100	3-999
4	3-11	1-20	2-45	7-49	6-53	5-59	4-999
5	3-33	2-63	4-65	7-72	6-76	1-86	5-999
6	5-21	4-31	1-36	7-52	2-53	3-89	6-999
7	2-31	3-43	5-52	1-58	6-60	4-67	7-999

TABLE-3.12: - The Alphabet Table for Bound Calculation

Sl.	Value	Cum. Value	Row	Col.	Sl.	Value	Cum. Value	Row	Col.
1	5	5	3	2	**22**	51	630	2	1
2	8	13	1	7	**23**	52	682	6	7
3	9	22	1	4	**24**	52	734	7	5
4	11	33	4	3	**25**	53	787	4	6
5	16	49	3	4	**26**	53	740	6	2
6	20	69	2	7	**27**	58	798	7	1
7	20	89	4	1	**28**	59	857	4	5
8	21	110	6	5	**29**	60	917	7	6
9	28	138	3	5	**30**	63	980	1	6
10	28	166	3	7	**31**	63	1043	5	2
11	29	195	2	6	**32**	65	1108	5	4
12	31	226	6	4	**33**	67	1175	7	4
13	31	257	7	2	**34**	72	1247	5	7
14	33	290	5	3	**35**	75	1322	1	2
15	35	325	1	5	**36**	76	1398	5	6
16	35	360	3	6	**37**	86	1484	2	3
17	36	396	6	1	**38**	86	1570	5	1
18	43	439	7	3	**39**	88	1658	2	5
19	45	484	4	2	**40**	89	1747	6	3
20	46	530	2	4	**41**	99	1846	1	3
21	49	579	4	7	**42**	100	1946	3	1

Set the 'current trial solution value' a large number, say $n \times \max(c_{ij})$. Now since the starting node is '1', the number of 'legitimate' nodes in 1st row is 6. These nodes in the order appearing in 1st row (of the alphabet table) and the probabilities, with which they are to be selected, are given below.

Legitimate Nodes	Probabilities	Cumulative Probabilities	Random Number	Node to be concatenating
7	0.286	0.286	0.572	5
4	0.238	0.524		
5	0.190	0.714		
6	0.143	0.857		
2	0.095	0.952		
3	0.048	1.000		

So, the partial tour will be (1, 5) with value 35 and bound of this leader is 69. Since (bound + travel cost) is less than the 'current trial solution value', we accept the latest node and go ahead. Number of 'legitimate' nodes in 5th row is 5. So, we proceed in a similar way. The following table gives idea how the tour is built.

Row	Legitimate Nodes	Random Number	Node to be concatenating	Bound	Partial tour	Tour value
5	3, 2, 4, 7, 6	0.753	4	49	(1, 5, 4)	100
4	3, 2, 7, 6	0.335	3	44	(1, 5, 4, 3)	111
3	2, 7, 6	0.932	6	28	(1, 5, 4, 3, 6)	147
6	7, 2	0.543	7	64	(1, 5, 4, 3, 6, 7)	199
7	2	---	2	----	(1, 5, 4, 3, 6, 7, 2)	281*

*Note: - The value of the partial tour is 230. Since all nodes in the network are present in the current tour, so add the edge connecting to the node '1' from the latest node. Then the value of the tour will be 281. For the other problems also we will report the final tour value in this stage.

Since the final tour value is less than the 'current trial solution value', so replace this 'current trial solution value' by this travel cost. This completes one trial.

Repeat the whole process $5n^3$ times.

Then the algorithm is slightly modified. Instead of considering all the 'legitimate' nodes, we also consider $k = \frac{l}{10}+2$ at most, where l is the number of 'legitimate' nodes. It is interesting to see that the algorithm with **restricted** number of 'legitimate' nodes, where $k = \frac{l}{10}+2$, is more efficient than that of with all 'legitimate' nodes. So, we shall follow the latter approach for the present problem and also for the problems considered in this present dissertation. Let us have a look at the comparative study of the programs in **Table-3.13**. Relative efficiency analysis is carried out for three sets of randomly generated problems of sizes 20, 30 and 40. Each set contains 5 randomly generated problems.

Table-3.13: - Comparative study of SCS programs.

Seed	N=20				N=30				N=40			
	Without restriction		With restriction		Without restriction		With restriction		Without restriction		With restriction	
	Sol.	Time	Sol.	Time	Sol.	Time	Sol.	Time	Sol.	Time	Sol.	Time
930	**361**	**1.92**	177	1.10	**527**	**14.61**	238	7.97	**750**	**64.97**	260	29.88
160	356	1.93	193	1.27	539	13.51	254	7.91	780	65.69	293	31.36
681	363	1.92	163	1.09	565	13.78	200	7.69	755	62.29	290	29.88
418	376	1.87	182	1.21	583	14.56	253	7.96	784	63.00	289	30.26
522	328	1.81	210	1.21	560	14.00	248	7.96	793	64.59	285	30.15
Mean		**1.89**		**1.18**		**14.09**		7.90		**64.11**		**30.15**
S.D.		**0.05**		**0.07**		**0.43**		**0.11**		**1.27**		**0.55**

Considerations of trade off between possible gain in capturing a 'better' solution and the cost of 'hidden computation' involved in implementing the chance-selection procedure suggests that restricting choice of chance-selection to only the first few of the 'legitimate nodes' in the alphabet table seem to give better results than the other. This is clearly brought out by the computational experience, reported in **Table-3.13.**

In this SCS approach the motivation for using ranks to the bias probabilities is to select the nodes, which have lesser values. Now, let us have a comparative study of this approach and the approach having no ranks to the probabilities of selecting a node. The comparative study is carried out for the problems of sizes 20, 30 and 40, and is reported in the **Table-3.14.**

Table-3.14: - Comparative study of SCS programs with ranks and without ranks to the bias probabilities.

Seed	N=20				N=30				N=40			
	Without ranks		With Ranks		Without ranks		With ranks		Without ranks		With ranks	
	Sol.	Time	Sol.	Time	Sol.	Time	Sol.	Time	Sol.	Time	Sol.	Time
930	214	1.10	177	1.10	281	7.58	238	7.97	316	29.33	260	29.88
160	210	1.20	193	1.27	272	7.36	254	7.91	359	30.70	293	31.36
681	217	1.16	163	1.09	271	7.20	200	7.69	359	29.66	290	29.88
418	204	1.10	182	1.21	293	7.41	253	7.96	299	29.11	289	30.26
522	219	1.15	210	1.21	294	7.45	248	7.96	349	20.39	285	30.15
Mean		**1.14**		**1.18**		**7.46**		**7.90**		**29.64**		**30.15**
SD.		**0.04**		**0.07**		**0.19**		**0.11**		**0.56**		**0.55**

From this **Table-3.14**, it is seen that the ranks to the bias probabilities is better than the other, in the context of solution value, which is our main aim in the context of the sequential constructive sampling approach.

3.4.3 MODIFIED 2-OPT MOVE

To improve the quality of the solutions obtained after the SCS program, we apply the following Modified 2-Opt move. As it is seen in the 2-Opt Move, in section 2.2.2.1, that given a tour

$$\{1\rightarrow\alpha_1\rightarrow\alpha_2\rightarrow\ldots\rightarrow\alpha_i\rightarrow\alpha_{i+1}\rightarrow\alpha_{i+2}\rightarrow\ldots\rightarrow\alpha_{j-1}\rightarrow\alpha_j\rightarrow\alpha_{j+1}\rightarrow\ldots\rightarrow\alpha_{n-1}\rightarrow 1\},$$

a neighboring solution can be easily generated at random by generating randomly 'i' and 'j'. So, the new tour will be

$$\{1\rightarrow\alpha_1\rightarrow\alpha_2\rightarrow\ldots\rightarrow\alpha_i\rightarrow\alpha_j\rightarrow\alpha_{i+2}\rightarrow\ldots\rightarrow\alpha_{j-1}\rightarrow\alpha_{i+1}\rightarrow\alpha_{j+1}\rightarrow\ldots\rightarrow\alpha_{n-1}\rightarrow 1\}.$$

In order to check the preferability of the new tour, just to calculate

$$\{c(\alpha_i, \alpha_j) + c(\alpha_{i+1}, \alpha_{j+1})\}-\{c(\alpha_i, \alpha_{i+1}) + c(\alpha_j, \alpha_{j+1})\}.$$

If this has a negative value, the new tour is preferable to the current one (cf, Reeves 1993).

Here, only the pairs of new and old edges are considered to measure the preferability of the new tour. But, their preceding and succeeding edges before and after edge exchange are not considered, which should also be considered. We have considered those edges also. Hence the modified measure of preferability of the new tour is

$\{c(\alpha_i, \alpha_j) + c(\alpha_j, \alpha_{i+2}) + c(\alpha_{j-1}, \alpha_{i+1}) + c(\alpha_{i+1}, \alpha_{j+1})\}-\{c(\alpha_i, \alpha_{i+1}) + c(\alpha_{i+1}, \alpha_{i+2}) + c(\alpha_{j-1}, \alpha_j) + c(\alpha_j, \alpha_{j+1})\}$.

This is our Modified 2-Opt move.

Now, to see how much this approach improves the solution quality, let us have a comparative study of the SCS and SCS+ Modified 2-Opt move (which is named as MSCS) for three sets of the problems of sizes 20, 30 and 40 (see in **Table-3.15**).

Table-3.15: - Comparative study of SCS and MSCS.

Seed	N=20				N=30				N=40			
	SCS		MSCS		SCS		MSCS		SCS		MSCS	
	Sol.	Time	Sol.	Time	Sol.	Time	Sol.	Time	Sol.	Time	Sol.	Time
930	177	1.10	177	1.16	238	7.97	238	7.97	260	29.88	250	29.88
160	193	1.27	193	1.32	254	7.91	253	7.97	293	31.36	287	31.42
681	163	1.09	163	1.10	200	7.69	200	7.69	290	29.88	262	29.93
418	182	1.21	182	1.21	253	7.96	253	7.96	289	30.26	284	30.32
522	210	1.21	210	1.21	248	7.96	234	7.97	285	30.10	285	30.15
Mean		**1.18**		**1.19**		**7.90**		**7.91**		**30.15**		**30.33**
SD.		**0.07**		**0.07**		**0.11**		**0.11**		**0.55**		**0.57**

From this **Table-3.15**, it is seen that as the size of the problem increases the Modified 2-Opt move improves the solution values obtained by SCS. Though the improvement is not so large, still for all of the problems considered in this present dissertation, we shall incorporate the Modified 2-Opt move and the name will be same as SCS. For the other Restricted TSPs also, this modified 2-opt move, taking constraints into account, is incorporated to the solution as done in the usual salesman case.

CHAPTER IV

GENETIC ALGORITHM APPROACH

4.1 INTRODUCTION

Genetic Algorithms (GAs) are search and optimization algorithms; they were first developed by John Holland (cf, Holland 1975). They are based essentially on 'mimicking' the 'survival of the fittest among the species' generated by random changes in the gene-structure of the 'chromosomes' in the evolutionary biology (cf, Goldberg, 1989).

GAs were designed by exploiting the metaphor of evolution. Generation after generation the individual species compete with each other to survive (Darwinian selection). Better members are likely to mate with one another in subsequent generations. This mating leads to recombination of the genetic materials of the fit members and often leads to a sequence of generations, which are successively fit. To transform this evolutionary metaphor into a way of developing heuristics, the essential idea is to treat each solution for a problem as an 'individual', whose fitness typically is either the corresponding objective function value or a closely related function. The solutions for a problem are represented or coded as strings. The string representing an individual is called a chromosome. Also in genetic terminology, variables are often called genes, the possible values of a variable are called alleles, and the position of a variable (e.g., solution value in an array) in a string is called its locus. In natural systems, one or more chromosomes combine to form the total genetic prescription for the construction and operation of some organism. In genetics, the total genetic package is called the *genotype* and the organism formed by the interaction of the total genetic package with its environment is called the *phenotype*. In terms of a GA, the coded string, which is processed by the algorithm is called genotype, while the decoded set of parameters represents the phenotype.

GAs differ from traditional optimization techniques in many aspects. They work with an encoding of the variables (typically as strings) rather than variables themselves, and use probabilistic transition rules to move from one population of solutions to another rather than a single solution to another. The most important and interesting characteristic of GAs is that they use only objective function evaluations. That is, they do not use any information on differentiability, convexity or other auxiliary characteristics of the function to be optimized. This property makes GAs easy to use and implement for wide variety of optimization problems. They are robust search algorithms, which are suited for problems having comparatively larger solution spaces. However, they are essentially heuristics, and by themselves, can not guarantee the optimality of the solutions they produce.

The main feature of GAs is that information is passed through generations. Moreover, in domain independent algorithms, like GAs, the learning process can be viewed as a search whose main function is to exploit the knowledge embodied in the good structures so far created and the exploration, through the combining operators (selection, crossover, mutation and inversion), of regions in the solution space. All this process is performed in parallel, in a sense, on a pool of strings; this prevents the algorithm from being trapped in a local minimum and, perhaps most importantly, by allowing for movements in the search space, which are non-optimal, to reach regions which could not be reached by a conventional descent algorithm. In this respect GAs are similar to Simulated Annealing algorithm.

4.2 GENETIC ALGORITHM IMPLEMENTATION

In order to solve any problem by GA, two main requirements are to be satisfied:

(i) a (possibly binary) string can represent a solution in the solution space,

(ii) an objective function and hence a fitness function which measures the goodness of a solution can be constructed / defined.

The work on GAs uses binary representation as well as decimal representation (cf, Goldberg 1989). The fitness function can be seen as the bridge connecting the GA to the real world problem under study. GAs do not start from one individual point in the search space, but from a population of strings, usually referred as gene pool.

The search of the solution space in a simple GA is performed by means of the following operators:

1. **Reproduction / selection,**
2. **Crossover,**
3. **Mutation,**

In the reproduction process, strings are copied into the next generation mating pool with a probability associated with their fitness value. By assigning to the next generation a higher portion of the highly fit strings, reproduction mimics the Darwinian survival-of-the-fittest in the natural world. In natural population, fitness is determined by a creature's ability to survive predators, pestilence, and other obstacles to adulthood and subsequent reproduction. In this phase no new string/ chromosome is produced. The commonly used reproduction operator is the proportionate reproduction operator, where a string is selected for the mating pool with a probability proportional to its fitness. More specifically, if $f(.)$ is the fitness function, and P_s is the size of population, then one selection process is performed using *roulette wheel* having P_s slots. Each slot corresponds to a population member, and the width of a slot is proportional to its fitness. There are P_s fixed pointers uniformly spaced around the slotted wheel. The wheel is spun once, and the positions marking these P_s pointers dictate the new population (cf, Goldberg 1989, Deb 1995). Although this *roulette wheel* selection is easier to implement, it is noisy. Hence, a more suitable version of this selection operator, called *Stochastic Remainder Selection method* (cf, Goldberg 1989, Deb 1995), is sometimes used. This operator can be discussed as follows.

The GAs are used for maximization problem. For the maximization problem the fitness function is same as the objective function. But, for minimization problem, one way of defining a 'fitness function' is as $F(x)=\frac{1}{1+f(x)}$, where $f(x)$ is the objective function. Now, for the algorithm, compute the "expected count" of each chromosome by dividing the corresponding fitness function value with the average fitness function value. After these expected count for each individual chromosome is calculated, the chromosomes are assigned copies exactly equal to the mantissa of the expected count. Then subtract the mantissa from the expected count of the corresponding individuals. Now, all of the expected counts are having value less than 1 (one). Then randomly select a chromosome and a random number (r). If r is less than the expected count of the corresponding chromosome, then insert the chromosome into the new mating pool and subtract 0.5 from the corresponding expected count. Repeat this process until the number of the chromosomes in the new mating pool equals to P_s.

The search of the solution space is done by creating new chromosomes from old ones. The most important search process is **crossover**. Firstly, a pair of parents is randomly selected from the mating pool. Secondly, a point, called crossover site, along their common length is randomly selected, and the informations after the crossover site of the two parent strings are swapped, thus creating two new children.

Let P1 and P2 be two parent chromosomes of length 7 and the randomly selected crossover site be say, between 3rd and 4th genes. Then the crossover operator gives the children (offsprings) O1 and O2 created as follows.

Parents	P1 :	a_1	a_2	a_3	$\vert$	a_4	a_5	a_6	a_7
	P1 :	b_1	b_2	b_3	$\vert$	b_4	b_5	b_6	b_7
Children	P1 :	a_1	a_2	a_3		b_4	b_5	b_6	b_7
	P1 :	b_1	b_2	b_3		a_4	a_5	a_6	a_7

As can be seen, crossover does not involve any change in the length of the chromosome; only an exchange of alleles of a string takes place. This exchange of information together with reproduction is the most powerful process in the GA search. This search is not just a simple random search, because through reproduction, the most promising region of the solution space are explored.

The **mutation** operator randomly selects a position in the chromosome and changes the corresponding allele, thereby modifying information. The need for mutation comes from the fact that as the less fit members of successive generations are discarded; some aspects of genetic material could be lost forever. By performing occasional random changes in the chromosomes, GAs ensure that new parts of the search space are reached, which reproduction and crossover alone couldn't fully guarantee. In doing so, mutation ensures that no important features are prematurely lost, thus maintaining the mating pool diversity.

The frequency of mutation is usually chosen to be considerably less than the frequency of crossover. So, mutation plays a secondary role in the GA search.

The GA operators, discussed above, exploit the similarities in string-structures to make an effective search. A schema (plural schemata) or similarity template represents a number of strings with similarities at certain string positions. For example in 7-bits situation, the schema

$$H = 1 \quad * \quad * \quad 1 \quad 0 \quad * \quad *$$

(a * denotes either 0 or 1), represents all the strings with 1 in the first position and, 1 and 0 in the 4th and 5th positions respectively. This is analogues to the leader defining a block, the difference being that the positions whose 'contents' are defined (to be 0 or 1 as the case may be) need not be consecutive. For strings of length l and binary alphabet $\{0,1\}$, there are 2^l-defined strings, but 3^l schemata. Also for each string there are 2^l schemata, and in a population of size p, there are at most p 2^l schemata. The length of a schema H, δ (H), is the difference in the outermost defined positions and the order of schema, o(H), is the number of fixed positions in the schema. For example, the schema H, considered above, has $\delta(H)= 4$ and $o(H)=3$ since the gap between the 1st and last defined (unambiguously defined) positions is 5-1=4 while only 3-bits positions are defined in value. In the gene pool the fittest chromosomes share some common features: the genes which make them successful chromosomes. Schemata define these shared features, and can be regarded as the building blocks of the different chromosomes. The fundamental role of crossover is to shuffle the different building blocks. As a result, short schemata (i.e., those with smaller o(H)) will be more likely to survive as opposed to long schemata. The ideal situations for a GA are those where short, low-ordered schemata combine with each other to form better and better solutions. This is the building block hypothesis of GA (cf, Goldberg 1989). The combined action of reproduction and crossover defines the corner stone of the schemata theory. Of course, this schema theory has been heavily criticized in the past 10 years.

4.3 CONTROL PARAMETERS IN GA

These are the parameters that govern the GA search process. Some of them are:

(i) **Population size:** - It determines how many chromosomes and thereafter, how much genetic material is available for use during the search. If there is too little, the search has no chance to adequately cover the space. If there is too much, the GA wastes time evaluating chromosomes.

(ii) **Crossover rate**: - It specifies the probability of crossover occurring between two chromosomes.

(iii) **Mutation rate**: - It specifies the probability of doing bit-wise mutation

(iv) **Termination criteria**: - It specifies when to terminate the genetic search. Any heuristic solution is essentially 'trial and error' solution; when 'enough' number of trials are made (i.e., when the number of solutions, generated according to the heuristic rules employed, is felt large enough), one stops further solution generation and accepts the best solution found so far as the (reasonable near-) optimal solution to the given problem. It may include a specified number of generations being exceeded, or certain high fitness value chromosome is found, or population has become sufficiently homogeneous or any other meaningful termination criterion.

4.4 GENETIC ALGORITHMS FOR THE TSP- A REVIEW

Application of GAs to the operations research problems has been limited due to the complexity of feasible domains. Given an optimization problem, often the hardest step in applying a GA is the encoding the solutions as strings so that the crossovers of feasible solutions result in feasible solutions. However, several genetic based algorithms have been reported (cf, Goldberg 1989, Reeves 1993, Michalewicz 1994).

The techniques for encoding solutions vary by problem and, involve a certain amount of art. The binary representation is not suited for the TSP. For the TSP, solutions are typically represented by integer numbers of length equal to the total number of the nodes in the network, n, where each string position can take any integer in {1,2,......,n} without any repetition.

There are three representations considered for representing a tour of the TSP in GAs. These are as follows:

(i) **Adjacency representation:** - The adjacency representation does not support the classical crossover operator. A repair algorithm might be necessary. Grefenstette et al. (1985) developed three crossover operators for this representation, viz., alternative edges, subtour chunks, and heuristic crossover. Among them the heuristic crossover is the best operator. However, as reported by them, performance of the heuristic crossover is not outstanding for the 50, 100 and 200 city problems. For 50, 100 and 200 cities, the algorithm found tours within 25%, 16% and 27% of the optimum, in approximately 15000, 20000 and 25000 generations, respectively.

(ii) **Ordinal representation:** - The ordinal representation represents a tour as a list of n-cities; the i^{th} element of the string is a number in the range from 1 to n-i+1. The idea behind the ordinal representation is as follows. There is some ordered list of cities C, which serves as reference point for string for ordinal representation. For example, let C be the ordered list of cities C={1,2,3,4,5,6,7}. A tour {1→3→4→2→5→7→6→1} is represented as a string S=(1,2,2,1,1,1,1) and can be interpreted as follows:

The first number on the string S is 1, take the first city from the list C as the first city of the tour (i.e., city 1), and remove it from C. The next number is 2, so take the 2^{nd} city of the current list C as the next city of the tour (i.e., city 3) and remove it from list C, and so on.

The main advantage of the ordinal representation is that classical crossover works. However, experimental results show that this representation together with classical crossover does not give good result for the TSP (cf, Grefenstette et al. 1985).

(iii) **Path representation:** - This path representation also does not support the classical crossover operator. Several authors have tried to define a crossover-like operators, which produce a legal tour.

Goldberg and Lingle (1985) defined an operator called PMX (partially mapped crossover), which used two crossover points. The section between these points defines an interchange mapping. For example, two parents P1 and P2 of length 7 at crossover points 3 and 5, produce offspring O1 and O2 as follows:

```
P1:   1     5     7  |  3     6  |  4     2
P2:   1     6     2  |  4     3  |  5     7
```

First the segments between crossover points are swapped (the symbol 'x' can be interpreted as 'at present unknown'):

```
O1:   x     x     x  |  4     3  |  x     x
```

O2: x x x | 3 6 | x x

This swap defines also a series of mappings $4 \leftrightarrow 3$ and $3 \leftrightarrow 6$. Then we can fill further cities (from the original parents), for which there is no conflict:

O1: 1 5 7 4 3 x 2

O2: 1 x 2 3 6 5 7

Finally, the x in the offspring O1 (which should be 4, but there was a conflict) is replaced by 6, because of the mapping $4 \leftrightarrow 3$, $3 \leftrightarrow 6 \Rightarrow 4 \leftrightarrow 6$. Similarly, the x in the offspring O2 is replaced by 4. The offspring are

O1: 1 5 7 4 3 6 2

O2: 1 4 2 3 6 5 7

This PMX operator of Goldberg and Lingle was the first attempt to apply GAs to the TSP, in which they found near-optimal solutions to a well-known 33-city problem. Operators similar to PMX have also been devised by others as well(cf, Davis 1985, Oliver et al. 1989).

The OX (ordered crossover) operator developed by Davis (1985) builds offspring by choosing a subsequence of a tour from one parent and preserving the relative order of cities from the other parent. For example, two parents P1 and P2 of length 7, at crossover points 3 and 5, produce offspring O1 and O2 as follows:

P1: 1 5 7 | 3 6 | 4 2

P2: 1 6 2 | 4 3 | 5 7

First the segment between crossover points are copied (the symbol 'x' can be interpreted as 'at present unknown'):

O1: x x x | 3 6 | x x

O2: x x x | 4 3 | x x

Next, starting from the second crossover point of one parent, the cities from the other parent are copied in the same order, omitting symbols already present. Reaching the end of the string, we continue from the first place of the string. The sequence of the cities in the second parent (from the second crossover point) is

{5,7,1,6,2,4,3}; after removal of cities 3 and 6, which are already in the first offspring, we get {5,7,1,2,4}. This sequence is placed in the first offspring (starting from the second crossover point, except that city 1 will have to appear in 1st position of the string):

O1: 1 2 4 3 6 5 7

O2: 1 7 6 4 3 2 5

Although PMX and OX are similar, they process different kinds of similarities. PMX leads to respect absolute city position, whereas OX leads to respect relative city position.

Another crossover operator, named CX (cycle crossover) operator was proposed by Oliver et al. (1987), where offspring are built in such a way that each city (ad its position) comes from one of the parents., This procedure may be illustrated as follows:

Let P1 and P2 below be two parents, of length 7.

P1: 1 5 7 3 6 4 2

P2: 1 6 2 4 3 5 7

The first offspring is produced by taking the 2nd city (as city 1 will have to appear in the 1st position of the offspring) from the first parent:

O1: 1 5 x x x x x

Since every city in the offspring should be taken from one of its parents (from the same position), the next city must be city 6, as the city from P2 just below the selected city 5. In P1 this city is at position '5'. Thus

O1: 1 5 x x 6 x x

in turn, implies city 3, as the city from P2 just 'below' the selected city 6. Thus

O1: 1 5 x 3 6 x x

Following this rule, the next city to be included is 4. However, the selection of city 4 requires the selection of city 5, which is already in the list- thus we have completed a cycle

O1: 1 5 x 3 6 4 x

The remaining cities are filled from the other parent:

O1:	1	5	2	3	6	4	7

Similarly,

O2:	1	6	7	4	3	5	2

The experimental investigation of Oliver et al. (1987) shows that OX does 11% better than PMX, and 15% better than CX.

Whitley et al. (1991) proposed a crossover operator, named as ERX (Edge Recombination Crossover), which uses an 'edge map' to construct an offspring that inherits as much information as possible from the parent structures. This edge map stores all the connections from the two parents that lead into and out of a city. The following example illustrates this procedure. Consider two parents P1 and P2 as

P1:	1	5	7	3	6	4	2
P2:	1	6	2	4	3	5	7

The edge map for these parents is as follows:

1 has edges to: 5, 2, 6, 7	2 has edges to: 4, 1, 6
3 has edges to: 7, 6, 4, 5	4 has edges to: 6, 2, 3
5 has edges to: 1, 7, 3	6 has edges to: 3, 4, 1, 2
7 has edges to: 5, 3, 1	

The ER algorithm is as follows:

Step 1: Choose the initial city from one of the two parents. This is the 'current city'.

Step 2: Remove all the occurrence of the 'current city' from the left hand side of the edge map.

Step 3: If the 'current city' has entries in its edge-list go to **step 4**; else go to **step 5**.

Step 4: Determine which of the cities in the edge-list of the 'current city' has the fewest entries in its own edge-list. The city with the fewest entries becomes the 'current city'. Ties are broken randomly. Go to **step 2.**

Step 5: If there are no remaining 'unvisited' cities, then **STOP**, otherwise, randomly choose an 'unvisited' city and go to **step 2**.

As an illustration, the new offspring is initialized with city 1, as it is our starting city. The edge-list for '1' indicates the candidates for the next city are 5, 2, 6 and 7. The city 5 and 7 have two edges: initial three minus 'city 1', and the cities 2 and 6 each have three edges: initial four minus 'city 1'. City 2 and 6 have three edges and thus are not considered. Assume city 5 is randomly chosen.

City 5 now has edges to city 7 and 3. City 7 is chosen next, since it has fewest edges.

City 7 only has an edge to city 3, so city 3 is chosen next.

City 3 has edges to city 6 and 4, both of which have two edges left. Randomly choose city 4.

City 4 only has edges to city 6 and 2, both of which have one edge left. Randomly choose city 6.

City 6 only has an edge to city 2, of course this is the last city to be taken, so city 2 is chosen next.

The resulting offspring is

O: 1 5 7 3 4 6 2

The experimental result as reported by Whitley et al. (1991) shows that ERX gives equal solution value on 15 out of 30 runs to the best known solution of a 105 city problem, and on remaining 15 runs, the solution found was always within 1 percentage of the best known solution.

Another crossover operator, named C1-Crossover (C1X) operator, was proposed by Reeves (1993), which is as follows:

Choose a crossover point x randomly, take the pre-x section of the first parent, and fill up the chromosome by taking in order each 'legitimate' element from the second parent. For example, two parents P1 and P2 of length 7, at crossover point 3, produce offspring O1 and O2 as follows:

```
P1:  1   5   7 | 3   6   4   2
P2:  1   6   2 | 4   3   5   7
```

First the segments before crossover point are copied (the symbol 'x' can be interpreted as 'at present unknown'):

```
O1:  1   5   7 | x   x   x   x
O2:  1   6   2 | x   x   x   x
```

Then fill up the 'x's of O1 by taking 'legitimate' alleles from P2 in order. So,

```
O1:  1   5   7   6   2   4   3
```

Similarly,

```
O2:  1   6   2   5   7   3   4
```

The rationale for C1-operator is that it preserves the absolute positions of the cities taken from P1, and the relative positions of those taken from P2.

A new crossover operator based on the conventional N-point crossover operator, named as GNX (Generalized N-point Crossover), was proposed by Radcliffe and Surry (1995), which is as follows: let $L = \{l_1, l_2, ..., l_m\}$ be a set of crossover points, with $0 < l < l_2 < .... < l_m < n$. This breaks a parent chromosome X into m+1 segments

$$(X_1, X_{2,}...., X_{l_1-1}), (X_{l_1}, X_{l_{1+1}}, ... X_{l_2-1}), ..., (X_{l_m}, X_{l_{m+1}}, ... X_n),$$

and breaks the second chromosome, Y, into corresponding segments.

The first phase of GNX operation uses the same genetic material as ordinary N-point crossover, i.e., alternate segments from two parents. It proceeds by picking a random order to visit m+1 segments (irrespective of the parents to which these segments are assigned). Within each segment, the alleles are 'tested' in a random order. An allele is 'tested' by seeing whether it can be placed in the child – i.e., whether it is compatible with those alleles that have already been accepted. If compatible, the new allele is inserted, otherwise, it is discarded. Then the complementary alternative sections are used to fill up the gaps, if any. The segments are again visited in a random order and the alleles within them are tested in a random sequence. If the child is still not fully specified after this, it is completed at random from amongst the legal combination of alleles.

Consider two parents P1 and P2, and G2X with cross points 3 and 5, where the

```
P1:  1   5   7  | 3   6  | 4   2
P2:  1   6   2  | 4   3  | 5   7
```

bold alleles are the ones that would normally be chosen by N-point crossover. Suppose the order in which the segments are tested is (2, 3, 1). Then the 2nd segment of P1 will be inserted whole, give the proto-child (x x x 3 6 x x). Alleles in the 3rd segment from P2 will then be tested in a random order. Both the city 5 and 7 will be accepted, giving the proto-child (x x x 3 6 5 7). The 1st segment of P2 is then tested, and 1 and 2 will be accepted, giving final proto-child at the end of the 1st phase as

(1 x 2 3 6 5 7).

The untested segments are the visited in random order. Only the 1st segment for P1 is relevant here. All the cities are rejected. So, the proto-child at the of the 2nd phase is as

(1 x 2 3 6 5 7).

Since this child is still incomplete, it must be randomly filled up. In this case however, only one legal chromosome has the required allele pattern, so the final child is given by

O: 1 4 2 3 6 5 7.

As reported by Radcliffe and Surry (1995), the results obtained with GNX are at least competitive with, and arguably superior to, those obtained with edge recombination.

The classical mutation operator does not support any of the tour representation for the TSP. So, mutation operator also needs to be re-defined in the context of the TSP. Several researchers developed several mutation operators (cf, Michalewicz 1994) for the TSP, they are as follows:

(a) **Inversion-** select two points along the length of chromosome, which is cut at these points, and reverse the substring between these points.

(b) **Insertion-** select a city and insert it in a random place.

(c) **Displacement-** select a substring and insert it in a random place.

Reciprocal exchange- select two cities randomly and swap them.

Traditional GAs focus on the global aspects of an optimization task, whereas local search methods in contrast focus on the local aspects of optimization task. The hybridization of both, genetic algorithm and local search methods has been shown to be an effective route to follow for finding high quality solutions for the TSP (cf, Mühlebin et al. 1988, Ulder et al. 1991, Freisleben and Merz 1996). Ulder et al. (1991) developed GA by incorporating 2-Opt, Or-Opt, 3-Opt and LK local search heuristics. Keeping the computational time fixed they showed that the usage of LK-heuristic gave the best quality solutions; for the ATT 532-cities problem their algorithms found tour within 17% of the optimum.

4.5. THE SEQUENTIAL CONSTRUCTIVE OPERATOR

We represent a tour as a path (full cycle) rather than as ‘permutation’. We have developed a genetic operator that generates high quality solutions to the TSP; this operator gives results which are remarkably better than the others listed above. It constructs an offspring using better links on the basis of their values present in the parents structure. It also uses the better links, which are present neither in the parents structure. We refer to this operator as sequential constructive (SC) operator.

As the ERX and GNX, the SC operator does not depend only on the parents structure, it sometimes introduces new, but good, links to the offspring, which are not event present in the present population. Hence, the chances of producing a better offspring are more than those of ERX and GNX. The algorithm for the sequential constructive operator is as follow:

Step 1*:* - Start from node '1' (i.e., current node i=1).

Step 2*:* - Select two 'legitimate' nodes appeared immediately after node 'i', one from each of the chromosomes'. Between these two nodes, select node 'j' such that the cost to go to node 'j' from node 'i' is minimum. If in one of the parent chromosomes, no any 'legitimate' node is present after node 'i', then select the first 'legitimate' node (i.e., node 'j') of other parent chromosome. Also, if node 'i' is the last allele of both the chromosomes (or there is no any 'legitimate' node after node 'i') and there are some nodes which are still to be visited, then select the node amongst the 'legitimate' nodes which has least cost from node 'i'. The tie is broken randomly. That is, go to node 'j' next and then rename the node 'j' as node 'i'.

Step 3*:* - Repeat **Step 2** until all nodes have been visited.

Let us illustrate this sequential constructive operator through our earlier example (see **Table-3.1** of section 3.2.2). Let a pair of selected chromosomes be A1 and A2, with values 312 and 331 respectively.

A1:	1	5	7	3	6	4	2
A2:	1	6	2	4	3	5	7

Select node 1, the 1st allele and the 'legitimate' nodes after node 1 in A1 and A2 are 5 and 6 respectively with $c_{15}=35$ and $c_{16}=63$. Since $c_{15} < c_{16}$, accept node 5. So, the partially constructed chromosome will be (1,5).

The 'legitimate' nodes after node 5, in A1 and A2 are 7 both. So, accept the node 7, and the partially constructed chromosome will be (1,5,7).

The 'legitimate' nodes after node 7, in A1 is 3 but in A2 is not available. So, the only option is to accept node 3. Thus, the partially constructed chromosome will be (1,5,7,3).

Again, the 'legitimate' nodes after node 3, in A1 is 6 but in A2 is not available. So, the only option is to accept node 6. Thus, the partially constructed chromosome will be (1,5,7,3,6).

The 'legitimate' nodes after node 6, in A1 and A2 are 4 and 2 respectively with c_{64}=31 and c_{62}=52. Since $c_{64} < c_{62}$ So, accept the node 4, and the partially constructed chromosome will be (1,5,7,3,6,4).

The 'legitimate' nodes after node 4, in A1 is 2 but in A2 is not available. So, the only option is to accept node 2. Thus, the full chromosome will be (1,5,7,3,6,4,2), with value 312.

Now, let us compare our SCX operator with ERX, C1X and GNX operators. To compare them, we have considered a randomly generated initial population scheme, the stochastic remainder selection method and the required crossover operator with crossover probability as **1.00**. Also, we have considered a population of size **400** and a maximum number of **4n** generations as the stopping rule. **Table-4.1(a)** gives the same for the randomly generated problems of sizes 30, 35 and 40.

Considering the solution quality, it is seen from the **Table-4.1(a)** that the sequential constructive operator is far superior to ERX, C1X and GNX. Of course, time-wise $4C1X \equiv 2GNX \equiv 1SCX$. So, we tried out the same problems by C1X with **16n** generations and GNX with **8n** generations. But the solution values are not improved. Again, we start with four 'initial' populations in case of C1X and two 'initial' populations in case of GNX, and retain the best solution so far as the (estimated) optimum (see **Table-4.1(b)**). It is still seen from the **Table-4.1(a & b)** that SCX is better than C1X, GNX and of course ERX.

Table-4.1(a):- Comparative study of different crossover operators.

N	Seed	ERX		C1X		GNX		SCX	
		Sol.	Time	Sol.	Time	Sol.	Time	Sol.	Time
30	**930**	422	11.37	453	0.44	364	0.77	201	1.81
	160	436	11.26	407	0.44	407	0.77	217	1.87
	681	395	11.64	514	0.49	403	0.77	202	1.87
	418	510	11.81	482	0.44	457	0.82	239	1.86
	522	512	11.43	490	0.44	423	0.77	222	1.87
	Mean		**11.50**		**0.45**		**0.78**		**1.86**
	S.D.		**0.20**		**0.02**		**0.02**		**0.02**
35	**930**	586	18.46	549	0.60	380	1.04	203	2.80
	160	513	18.07	488	0.61	363	1.04	251	2.81
	681	538	17.30	584	0.61	356	1.05	268	2.80
	418	506	17.47	432	0.60	412	1.04	195	2.74
	522	559	18.18	392	0.60	476	1.05	229	2.80
	Mean		**17.90**		**0.60**		**1.04**		**2.79**
	S.D.		**0.44**		**0.01**		**0.00**		**0.03**
40	**930**	637	25.38	632	0.77	540	1.32	222	3.96
	160	632	25.49	520	0.77	574	1.31	216	3.95
	681	605	25.37	575	0.82	551	1.38	237	3.96
	418	589	25.16	603	0.77	482	1.37	235	3.95
	522	608	25.26	587	0.77	623	1.32	242	3.90
	Mean		**25.33**		**0.78**		**1.34**		**3.94**
	S.D.		**0.11**		**0.02**		**0.03**		**0.02**

Table-4.1 (b): - Comparative study of different crossover operators.

N=30				N=35				N=40			
C1X		GNX		C1X		GNX		C1X		GNX	
Sol.	Time	Sol.	Time	Sol.	Time	Sol.	Time	Sol.	Time	Sol.	Time
364	1.70	364	1.56	399	2.45	380	2.10	495	3.29	540	2.70
407	1.76	395	1.60	457	2.49	363	2.13	520	3.22	574	2.66
403	1.69	403	1.61	460	2.40	356	2.05	562	3.19	525	2.71
377	1.81	443	1.57	431	2.51	351	2.15	565	3.23	455	2.70
369	1.70	423	1.61	392	2.43	455	2.17	497	3.25	551	2.69
Mean	**1.73**		**1.59**		**2.46**		**2.12**		**3.24**		**2.69**
S.D.	**0.05**		**0.02**		**0.04**		**0.04**		**0.03**		**0.02**

4.6 HYBRID GENETIC ALGORITHM (HGA) FOR THE TSP.

We randomly generate P_s initial feasible solutions (or chromosomes). Each 'chromosome' so generated being a salesman's tour has a corresponding cost, which is evaluated, and the best among them is treated as 'current trial solution value'.

So, given a current generation of population, the next generation of our hybrid genetic algorithm is created as follows.

(i) Reproduction: - *Stochastic Remainder* selection method (cf, Goldberg 1989, Deb 1995) is used here. As the reproduction operator does not create any new chromosomes, so it obviously does not violate any constraints also. Hence, for the Restricted TSPs, we shall use this operator and we need not have to mention about this operator there.

(ii) Crossover: C1-Crossover operator (C.f. Reeves 1993) is used for our HGA, which is as follows: choose a crossover point x randomly, take the pre-x section of the first parent, and fill up the chromosome by taking in order each 'legitimate' element from the second parent.

Though it is said that the GAs are robust search techniques, the global optimum with reasonable effort has remained a moot point. One of the reasons may be the random selection of the crossover point, which may result in the indiscriminate breaking down of the 'building block'. Therefore, for a pair of parents, select randomly several (=10, in this present study and also for all of the problems considered in this present dissertation) alternative crossover points and create several offspring as above. Once the offspring are created, they are evaluated, and only the one with the best objective value is included in the new generation.

(iii) Mutation: For each of the chromosomes of the P_s populations, we do this operation. Generate a random number (r) in the interval (0,1). If r is less than the prescribed probability of mutation (P_m), then select two genes (except the first gene) and then swap the alleles. Continue the process $\frac{n}{2}$ times for each chromosome.

(iv) Sequential Constructive operator: To improve the fitness of the chromosomes created after reproduction, crossover and mutation (i.e., the simple GA), this sequential constructive operator is incorporated. The motivation to use this approach is to build chromosomes such that the solution is likely to converge to the optimal solution quickly. Here we select two chromosomes and produce a single new chromosome from them.

Hence our HGA may be summarized as follows:

```
HGA( )
 { Initialize random population;
   Evaluate the population;
   Generation = 0;
     While stopping rule is not satisfied
       { Generation = Generation + 1;
         Select good solutions by reproduction procedure;
         Perform crossover with probability of crossover ($P_c$);
         Perform mutation with probability of mutation ($P_m$);
         Perform sequential constructive search technique;
         Evaluate the population;
       }
 }
```

We have allowed a maximum of 4n (n being the total number of nodes in the network) number of generations, and the best solution value obtained within these generations is retained as the best solution value.

The GA approach has been claimed to lead to very good, near-optimal solutions. However, the approach is obviously 'controlled or guided' by choice of parameters: viz. probability of crossover (P_c), probability of mutation (P_m), and population size (P_s). As Deb (1995) points out: successful working of GAs depends on a proper selection of these parameters, but often one is in the dark as to what values should be taken for these parameters. For HGA, several runs were executed with different settings of parameters for different problems. These runs were used to fine-tune the parameters and to set their values at P_c= 0.95, P_m= 0.09 (for all of the problems considered in this present dissertation) and different population sizes (P_s) for different problem sizes which will be reported in the corresponding tables showing the comparative study (for all of the problems).

To see how near-optimal solution is obtained by the HGA, we have tested some of the benchmark problems given in the TSPLIB (cf, Reinelt, 1995). In order to investigate the robustness of the HGA, 50 runs from different random initial solutions were performed, and the various performance measures (solution quality, running time etc) were averaged. The solution quality was measured by the percentage excess above the best known solution (or optimal solution, if known), as given by the formula

$$Excess = \frac{Solution\ Value - Best\ Known\ Solution\ Value}{Best\ Known\ Solution\ Value} x\,100.$$

Table-4.2 reports the best found solution by HGA, the percentage of excess over the best-known solution value (reported in the TSPLIB). It also reports the percentage of excess of the average solution value and the worst solution value over the best-known solution value, and standard deviation of the percentages of excess of 50 runs. The size of the populations is considered as 200 for solving problems in the TSPLIB.

Table-4.2: Results of benchmark problems by HGA only.

Problem	Reported Soln.	Best Soln.	Best (%)	Avg. (%)	Worst (%)	S.D. (%)	Avg. (Time)	S.D. (Time)
Ftv33	1286	1286	0	5.21	7.9	1.97	3.77	0.66
Ftv35	1473	1475	0.14	1.35	3.7	1.03	4.42	0.03
Ftv38	1530	1530	0	2.09	6.5	1.89	5.21	0.29
Ftv44	1613	1625	0.74	3.53	7.2	1.85	7.31	0.17
Ftv47	1776	1780	0.23	2.86	7.7	1.61	8.44	0.14
Ftv55	1608	1608	0	3.17	9.3	1.83	11.65	1.68
Ftv64	1839	1846	0.38	3.01	10.2	2.20	17.28	0.52
Ftv70	1950	1957	0.36	3.35	8.0	2.04	21.54	0.96
Ft53	6905	6923	0.26	3.99	11.0	2.11	10.79	0.36
Ft70	38673	38725	0.13	1.61	3.6	0.63	20.87	0.77
P43	5620	5624	0.07	0.30	0.5	0.08	6.94	0.26
Ry48p	14422	14511	0.62	4.08	7.3	1.55	8.61	0.28

It is seen (from **Table-4.2**) that the best tours found in 50 runs never exceeded 0.74% over reported best-known solution in TSPLIB. The average percentage over the optimal solution is also **very less,** not more than 5.21% only. So, the HGA is good for the usual TSP and hence we can consider it as a good basis for the Restricted TSPs also. Also, it is difficult to compare with other reported running times for different approaches due to the variety of machines used. It is to be noted that though the benchmark problems are available for the usual TSP, but they are not available for the other Restricted TSPs. Hence, we consider some randomly generated test problems of different sizes for the usual TSP as well as the Restricted TSPs and solve them by different approaches considered in this present dissertation.

4.7 RELATIVE EFFICIENCY OF DIFFERENT APPROACHES

Relative efficiency analysis was carried out for three sets of randomly generated problems of sizes 34, 36 and 50. Each set contains 20 randomly generated problems. For each of the problems generated, the exact optimal solution obtained by DGLS, and the best solution obtained by QE and SCS are tabulated.

Also are tabulated the times taken for obtaining the same. In case of HGA, the best-found solution; the solution-ratio of it to the optimal solution obtained by DGLS along with the average solution-ratio, worst solution-ratio and standard deviation of solution-ratios of 50 runs are reported. These are reported in **Table-4.3 (a, b & c).**

TABLE-4.3: - Solution values and time taken (in Seconds) by different algorithms, for twenty randomly generated problems of sizes 34, 36 and 39.

(a) N=34 and P_s=200.

Seed	DGLS		QE			SCS			HGA						
	Sol.	**Time**	**Sol.**	**Time**	**SR**	**Sol.**	**Time**	**SR**	**Best Sol.**	**Best SR**	**Avg. SR**	**Worst SR**	**S.D. SR**	**Avg. Time**	**S.D. Time**
930	174	0.60	174	0.22	1.00	258	14.28	1.48	174	1.00	1.12	1.34	0.07	3.59	0.36
160	151	9.83	151	4.28	1.00	246	14.28	1.63	154	1.02	1.14	1.34	0.07	3.62	0.03
681	209	1.38	209	0.54	1.00	287	14.67	1.37	210	1.01	1.10	1.26	0.06	3.66	0.03
418	153	1.48	157	0.77	1.03	243	13.90	1.59	163	1.07	1.19	1.39	0.08	3.66	0.03
522	179	16.09	179	6.21	1.00	256	14.17	1.43	187	1.05	1.12	1.27	0.05	3.68	0.04
667	215	0.75	215	0.33	1.00	293	14.09	1.36	224	1.04	1.15	1.33	0.06	3.61	0.03
264	193	18.08	197	12.71	1.02	282	14.23	1.46	194	1.01	1.10	1.24	0.06	3.62	0.03
826	125	0.16	125	0.06	1.00	190	13.97	1.52	128	1.02	1.20	1.35	0.08	3.61	0.03
15	198	14.24	198	6.49	1.00	264	13.89	1.33	200	1.01	1.08	1.20	0.04	3.63	0.03
85	165	10.11	165	5.45	1.00	254	14.08	1.54	165	1.00	1.17	1.34	0.08	3.64	0.34
855	156	0.82	156	0.45	1.00	234	14.19	1.50	160	1.03	1.18	1.34	0.08	3.65	0.03
334	179	19.55	183	10.56	1.02	249	14.11	1.39	180	1.01	1.12	1.29	0.06	3.66	0.03
597	182	4.45	186	4.47	1.02	263	13.98	1.45	186	1.02	1.16	1.34	0.07	3.64	0.03
493	147	0.22	147	0.06	1.00	214	13.87	1.46	151	1.03	1.15	1.31	0.06	3.61	0.03
348	172	0.17	173	0.11	1.01	261	14.01	1.52	176	1.02	1.17	1.36	0.09	3.66	0.03
19	178	1.27	178	0.48	1.01	256	14.50	1.44	180	1.01	1.16	1.43	0.06	3.61	0.03
802	201	9.24	208	4.47	1.03	288	14.44	1.43	201	1.00	1.13	1.24	0.06	3.63	0.03
795	197	49.86	201	21.90	1.02	295	15.16	1.50	203	1.03	1.15	1.29	0.08	3.64	0.03
102	204	1.51	207	1.12	1.01	271	14.24	1.33	208	1.02	1.13	1.29	0.04	3.65	0.04
28	146	2.78	150	1.04	1.03	221	13.04	1.51	146	1.00	1.16	1.42	0.08	3.66	0.03
Mean		**8.13**		**4.09**	**1.01**		**14.16**	**1.46**		**1.02**	**1.14**	**1.32**		**3.64**	
S.D.		**11.52**		**5.44**	**0.01**		**0.39**	**0.08**		**0.02**	**0.03**	**0.06**		**0.02**	

Conclusion ?

Seed	DGLS		QE			SCS			HGA						
	Sol.	Time	Sol.	Time	SR	Sol.	Time	SR	Best Sol.	Best SR	Avg. SR	Worst SR	S.D. SR	Avg. Time	S.D. Tim
930	168	0.27	168	0.11	1.00	279	19.00	1.66	168	1.00	1.17	1.44	0.08	6.14	0.65
160	186	31.53	189	7.91	1.02	284	18.95	1.53	189	1.02	1.15	1.34	0.06	6.27	0.03
681	168	1.43	169	0.49	1.01	254	18.01	1.51	171	1.02	1.10	1.24	0.06	6.25	0.03
418	156	4.67	159	1.98	1.02	221	18.98	1.42	160	1.03	1.16	1.38	0.08	6.25	0.02
522	205	190.42	205	82.77	1.00	292	14.06	1.42	209	1.02	1.08	1.19	0.04	6.25	0.02
667	194	3.01	195	0.89	1.01	283	18.02	1.46	199	1.03	1.10	1.22	0.04	6.22	0.02
264	198	1.95	204	1.56	1.03	281	18.21	1.42	203	1.03	1.11	1.23	0.04	6.19	0.02
826	139	1.05	142	0.49	1.02	236	17.98	1.70	144	1.04	1.14	1.35	0.07	6.20	0.03
15	177	50.06	178	14.42	1.01	253	18.32	1.43	179	1.01	1.08	1.20	0.04	6.22	0.02
85	172	3.90	172	1.68	1.00	270	17.77	1.57	176	1.02	1.13	1.32	0.06	6.23	0.03
855	168	7.09	173	4.56	1.03	235	15.09	1.40	173	1.03	1.13	1.33	0.07	6.19	0.03
334	183	28.19	183	13.21	1.00	269	15.86	1.47	191	1.04	1.12	1.28	0.05	6.22	0.03
597	154	11.05	158	8.18	1.03	254	16.98	1.65	156	1.01	1.21	1.37	0.08	6.25	0.02
493	149	1.75	151	0.95	1.01	222	17.10	1.49	153	1.03	1.18	1.40	0.09	7.13	1.66
348	186	43.28	186	13.25	1.00	245	15.01	1.32	186	1.00	1.09	1.19	0.06	6.52	1.22
19	215	206.19	216	63.23	1.00	300	19.70	1.40	217	1.01	1.08	1.27	0.06	6.29	0.03
802	149	1.91	152	0.87	1.02	250	18.06	1.68	154	1.03	1.16	1.40	0.08	6.29	0.03
795	167	4.70	167	1.36	1.00	240	18.06	1.44	170	1.02	1.10	1.28	0.05	6.29	0.03
102	194	194.49	196	76.30	1.01	273	19.12	1.41	199	1.03	1.13	1.25	0.05	6.33	0.04
28	157	13.53	158	2.86	1.01	262	18.52	1.67	167	1.06	1.16	1.36	0.07	6.35	0.03
Mean		**40.02**		**14.85**	**1.01**		**17.64**	**1.50**		**1.02**	**1.13**	**1.30**		**6.30**	
S.D.		**67.52**		**25.48**	**0.01**		**1.49**	**0.11**		**0.01**	**0.04**	**0.08**		**0.20**	

(c) N=39 and P_s=300.

Seed	DGLS		QE			SCS			HGA						
	Sol.	Time	Sol.	Time	SR	Sol.	Time	SR	Best Sol.	Best SR	Avg. SR	orst SR	S.D. SR	Avg. Time	S.D. Time
930	188	71.79	188	28.56	1.00	275	27.24	1.46	192	1.02	1.12	1.26	0.05	7.80	0.03
160	150	17.41	150	8.35	1.00	245	26.70	1.63	151	1.01	1.14	1.34	0.08	8.14	0.03
681	158	14.06	160	4.07	1.01	253	26.96	1.60	165	1.04	1.15	1.29	0.06	7.93	0.03
418	157	152.75	157	49.38	1.00	278	26.31	1.77	164	1.05	1.18	1.34	0.07	7.92	0.03
522	185	4.12	186	3.57	1.01	277	26.64	1.50	192	1.04	1.12	1.39	0.07	7.99	0.02
667	221	31.05	224	13.69	1.01	331	25.96	1.50	225	1.02	1.09	1.19	0.05	7.89	0.03
264	213	1257.8	220	651.8	1.03	303	25.32	1.42	218	1.02	1.14	1.30	0.06	7.82	0.22
826	174	53.31	175	5.88	1.01	270	26.32	1.55	179	1.03	1.14	1.29	0.05	7.91	0.02
15	177	8.75	182	6.79	1.03	256	26.11	1.45	180	1.02	1.13	1.33	0.05	7.92	0.03
85	185	25.96	185	7.01	1.00	234	26.07	1.26	186	1.01	1.10	1.25	0.06	7.86	0.03
855	170	425.80	173	136.8	1.02	249	25.12	1.46	173	1.02	1.11	1.21	0.05	7.89	0.03
334	179	256.03	179	112.2	1.00	267	27.32	1.49	187	1.05	1.14	1.37	0.06	7.87	0.03
597	184	4.21	188	2.03	1.02	315	26.95	1.71	186	1.01	1.15	1.30	0.07	7.89	0.03
493	185	174.31	190	65.92	1.03	284	26.85	1.54	190	1.03	1.15	1.39	0.07	7.93	0.02
348	166	13.53	167	6.59	1.01	291	26.36	1.75	171	1.03	1.13	1.23	0.05	7.99	0.03
19	195	2574.1	202	936.6	1.04	314	26.10	1.61	207	1.06	1.17	1.33	0.06	7.98	0.02
802	196	8.37	198	4.63	1.01	311	26.95	1.59	197	1.01	1.12	1.31	0.06	7.89	0.03
795	196	5403.8	205	2347.	1.00	271	26.38	1.38	196	1.00	1.12	1.26	0.06	7.68	0.03
102	168	361.33	168	143.4	1.00	268	26.36	1.60	176	1.05	1.16	1.36	0.07	7.99	0.02
28	181	97.00	185	42.60	1.02	294	26.49	1.62	182	1.01	1.10	1.22	0.05	7.86	0.02
Mean		**547.77**		**228.9**	**1.01**		**26.43**	**1.54**		**1.03**	**1.13**	**1.30**		**7.91**	
S.D.		**1262.1**		**539.9**	**0.01**		**0.55**	**0.12**		**0.02**	**0.02**	**0.06**		**0.09**	

CHAPTER V

THE TRAVELLING SALESMAN PROBLEM WITH PRECEDENCE CONSTRAINTS

5.1 INTRODUCTION

Good algorithms are available for the usual TSP. However quite a few situations require special restrictions on the acceptability of a tour as solution. The Travelling Salesman Problem with Precedence Constraints (TSP-PC, for short) is such a variation of the usual TSP, in which each admissible solution is to satisfy a set of k precedence relations denoted by $i_r < j_r$;r=1,2,....,k. That is, the node j_r should not be visited unless node i_r is already visited. So, node j_r is called predecessor node of corresponding node i_r. This chapter formally presents this variation of the problem and develops solution algorithm for the same.

The problem may be stated formally as follows:

A network with *n* nodes, with node 1 as 'headquarters' and a cost (or distance or time etc.) matrix $C=[c_{ij}]$ of order n associated with ordered node pairs (i, j) is given. Also is given a set of k precedence relations between node pairs (i.e., $i_r < j_r$; r=1,2,...,k) representing the restrictions that any cycle (tour) is to satisfy for its acceptability. The problem is to obtain the tours

$$\{1=\alpha_0, \alpha_1, \alpha_2, \ldots\ldots, \alpha_{n-1}, \alpha_n=1\} \equiv \{1\rightarrow\alpha_1\rightarrow\alpha_2\rightarrow\alpha_3\ldots\ldots\rightarrow\alpha_{n-1}\rightarrow 1\}$$

representing irreducible permutations interpreted as simple cycles in which the node i_r (as one of the α's) is to come before node j_r (as one of the α's in the cycle) for which the total travel cost

$$C(1=\alpha_0, \alpha_1, \alpha_2, \ldots\ldots, \alpha_{n-1}, \alpha_n=1) \mathrel{\hat{=}} \sum_{i=0}^{n-1} c(\alpha_i, \alpha_{i+1})$$

is minimum.

The importance of the TSP-PC is well explained by Scroggs and Therp (1972), Das (1976). For example, an executive may have to visit plant X before plant Y, since the information he gains at plant X may influence his discussions at plant Y, or he may personally want to deliver something from plant X to plant Y. Similarly, starting from a factory a truck has to deliver items to a number of depots. From certain depots the truck must have to pick up goods to be delivered to some other depots. Here, 'factory' means the 'headquarters' and the 'depots' mean the 'nodes' of the salesman problem. The tour, truck will travel must be of a minimum length (or cost or time), but certain depots must be visited before some others (cf, Lokin 1978). The TSP-PC has many other practical applications to sequencing and transportation problems (cf, Bianco et al. 1994).

Das (1976) proposed a lexisearch algorithm for obtaining optimal solution to the TSP-PC. However, the lower bound calculation method adopted therein is not efficient and hence the procedure is not practicable.

Lokin (1978) developed a branch and bound approach, based on the branch and bound approach of Little et al. (1963). A zero-one matrix P is introduced to indicate the precedence relations between the points (nodes) as follows:

$p_{ij}=1$; if point 'i' must be visited before point 'j'.

$=0$; otherwise.

The matrix P may not explicitly give all the transitive precedence relations. Hence, another matrix Q is to be derived in which all the transitive relation implied by the explicitly stated precedence constraints occur. For example, while $p_{ij}=1$, $p_{jk}=1$ and $p_{ik}\neq 0$, by implication, i must precede k and hence $q_{ik}=1$. Matrix Q can be derived from matrix P by Boolean matrix multiplication of P. If after a certain stage 'l', the matrix $P^{l-1}= P^{l}=0$, then the matrix Q equals P^{l-1}. In developing this branch and bound approach, the given 'cost' matrix is restructured at each stage so that it becomes impossible for branching forward by taking a link such that precedence relations are violated. This modification of the matrix is done by giving the value infinity to the elements of the matrix based on the precedence relations such that in selecting the links for a feasible solution the precedence relations are always obeyed. However, Lokin did not report any computational experience to the TSP-PC. A particular version of the TSP-PC with a generalized objective function, called *dial-a-ride problem*, is discussed by Psaraftis (1980, 1983).

Savelsbergh (1990) proposed modified 2-Opt and Or-Opt moves for the TSP, but he did not report any computational experience for the same. Bianco et al. (1994) developed exact and heuristic procedures, based on dynamic programming, to solve the TSP-PC. Two types of precedence constraints generation have been investigated by Bianco et al.(1994), which are as follows.

(i) **Dial-a-ride problems**: - The set of nodes (except the starting node) is randomly split in the set of origins and the set of destinations such that the number of nodes in these two subsets is equal. A random assignment of the origins to the destinations is made. Then a random feasible tour is generated, where the node 'i' will occupy the position α_i in the tour and each node 'i' can stay only in the range of positions $[e_i, u_i]$, where $e_i = \max\{2, \alpha_i - MPS\}$, $u_i = \max\{\alpha_i + MPS, n\}$ and MPS be the Maximum Position Shift from α_i accepted by node 'i'. The introduction of MPS reflects that, in some *dial-a-ride problems* for a vehicle starting from a depot and visiting all customers of a given *calling list*, it is required that the difference between the positions occupied by any customer in the vehicle tour and in the *calling list*, is within a fixed limit.

The computational report given by them shows that these problems with MPS=5 can be solved optimally up to n=105 in less than 300 seconds, but the size of the problems that can be solved decreases for increasing value of MPS.

(ii) **General precedence problems**: - A Maximum Number of Precedence (MNP) are randomly generated for each node 'i' (except the starting node) with the guarantee that a feasible solution exists. This is achieved by generating a random and choosing for each node 'i' (except the starting node), a subset A_i of those nodes that in the tour precede 'i' so that $|A_i| \leq MNP$.

The computational report shows that these problems become easier for increasing values of MNP and the problems of size 100 can be solved with MNP=20. Of course, as reported by them, some of the problems could not be solved optimally.

Mingozzi et al. (1997) also developed an exact algorithm based on the dynamic programming procedure for both the generalized precedence constraints and dial-a-ride problem, which outperforms the algorithms proposed by Bianco et al. (1994). But the computational experience shows that the problems involving the dial-a-ride precedence constraints can be solved to optimality only up to n=25 and the size of the problems involving the generalized precedence constraints can be solved decreases for decreasing values of MNP.

But our precedence constraint generation is completely different with that of their approach.

5.2. LEXISEARCH APPROACH (PATH APPROACH) AND ILLUSTRATION.

The procedure is essentially same as the lexisearch algorithms for the usual TSP of section 3.3, with appropriate modifications to include checking for precedence constraints.

At first, set the cost of $(1, j_r)$, $(i_r, 1)$ and (j_r, i_r) to large values in the given cost matrix C, for all r=1,2,....,k. Since the precedence relations are transitive, we can derive possibly many more precedence relations, which follow typically from the explicitly stated constraints. For instance two constraints: node α < node β and node β < node γ, lead to three constraints in all as one more constraint viz., node α < node γ is also implied by the two constraints.

Determination of all constraints from the explicitly given constraints can be achieved as follows:

For each r=1,2,...,k, we check whether i_r equals to j_s for any s=1,2,....,k. If any $i_r=j_s$, then increment k by 1 (one) and set $i_k=i_s$ and $j_k=j_r$. So, when the number of constraints is specified, we are only referring the explicitly stated constraints. Then we check the precedence-feasibility of the solutions, i.e., we test whether any node i_r equals to its corresponding node j_r, for any r=1,2,....,k. If yes, the feasible solution of that problem for these precedence constraints is not possible, then we generate another set of precedence constraints for the same problem; *else*, we do the lexi search approach.

Now, for convenience in checking the constraints, a predecessor table for all nodes in the network is derived. For this we generate initially a zero matrix P of order n. Then we enter the predecessors of the nodes in the corresponding row sequentially. This is done by the following method:

Let $p_{st}=0$; for all s, t=1,2,......,n. Then for all s=1,2,...,n, set first u=0 and then check whether for any r=1,2,....,k, j_r equals to s. If yes, increment u by one and put $p_{su}=i_r$. For a 10-node problem with constraints node 2 < node 5, node 6 < node 7, node 7 < node 9, node 10 < node 2 and node 8 < node 9, the 'predecessor table' will be as follows:

Node α	Predecessors of node α
1	0, 0, 0, 0, 0, 0, 0, 0, 0, 0
2	10, 0, 0, 0, 0, 0, 0, 0, 0, 0
3	0, 0, 0, 0, 0, 0, 0, 0, 0, 0
4	0, 0, 0, 0, 0, 0, 0, 0, 0, 0
5	5, 10, 0, 0, 0, 0, 0, 0, 0, 0
6	0, 0, 0, 0, 0, 0, 0, 0, 0, 0
7	6, 0, 0, 0, 0, 0, 0, 0, 0, 0
8	0, 0, 0, 0, 0, 0, 0, 0, 0, 0
9	7, 6, 8, 0, 0, 0, 0, 0, 0, 0
10	0, 0, 0, 0, 0, 0, 0, 0, 0, 0

Now, for a node to be concatenated to a leader, check whether the row elements of the corresponding node (row) are non-zero.

If yes, check whether all (or some of) the corresponding elements of V (as defined in the section 3.2) are zero.

If yes, then go for the next stage of testing (e.g., bound calculation), *else*, jump the block.

else (i.e., if the first element is zero), go for the next stage of testing (e.g., bound calculation).

Now, for the lexisearch algorithm of the TSP-PC, following steps are replaced in the lexisearch algorithm for usual TSP, described in section 3.3.

Step 0: - Form the 'alphabet table' based on the 'reduced' (i.e., bias-removal) cost matrix **after incorporating all precedence constraints.** Initialize the 'current trial solution value' to a large number. Since our starting node is '1', we start our computation from 1st row. Put r=1.

Step 2: - If the (incomplete) word forms a sub-tour or if any prescribed precedence constraints is violated, drop the city added in **step 1** and increment r by 1, and then go to **step 6**; *else,* go to **step 3**.

5.2.1 ILLUSTRATION

Working of this algorithm is explained through the seven city example considered earlier, with the two explicit constraints node 6 < node 5, node 5 < node 2 added. The number of constraints is three, as one more constraint, viz., node 6 < node 2, is derived from the explicitly stated constraints. So, set the cost of the edges (5, 6), (2, 5), (2, 6), (1, 5), (1, 2), (6, 1) and (5, 1) to a large number (for the convenience we take as 999). **Table-5.1** and **Table-5.2** give the modified cost matrix with bias and the reduced cost matrix respectively, while **Table-5.3** and **Table-5.4** give the 'alphabet table' and the logic-flow of the algorithm at various stages, which sequentially records the intermediate results, with decision taken (i.e., remarks) at these steps in every column respectively. In addition to the symbols given in section 3.2.2, the following symbol is also used therein.

RV: Restriction is violated, JB.

TABLE-5.1: - The Modified Cost Matrix with Bias

	1	2	3	4	5	6	7	Row Min
1	999	999	99	9	999	63	8	**8**
2	51	999	86	46	999	999	20	**20**
3	100	5	999	16	28	35	28	**5**
4	20	45	11	999	59	53	49	**11**
5	999	63	33	65	999	999	72	**33**
6	999	53	89	31	21	999	52	**21**
7	58	31	43	67	52	60	999	**31**
Col. Min	**9**	**0**	**0**	**1**	**0**	**29**	**0**	

Total Bias = 129 + 40 = 169.

TABLE-5.2:- The Reduced Cost Matrix

	1	2	3	4	5	6	7
1	982	991	91	0	991	26	0
2	22	979	66	25	979	950	0
3	86	0	994	10	23	1	23
4	0	34	0	987	48	13	38
5	957	30	0	31	966	937	39
6	969	32	68	9	0	949	31
7	18	0	12	35	21	0	968

TABLE-5.3: - The Alphabet Table

	N-V	N-V	N-V	N-V	N-V	N-V	N-V
1	4-0	7-0	6-26	3-91	1-982	2-991	5-991
2	7-0	1-22	4-25	3-66	6-950	2-979	5-979
3	2-0	6-1	4-10	5-23	7-23	1-86	3-994
4	1-0	3-0	6-13	2-34	7-38	5-48	4-987
5	3-0	2-30	4-31	7-39	6-937	2-957	5-966
6	5-0	4-9	7-31	2-32	3-68	6-949	1-969
7	2-0	6-0	3-12	1-18	5-21	4-35	7-968

TABLE-5.4: SEARCH TABLE

$1\to\alpha_1$	$\alpha_1\to\alpha_2$	$\alpha_2\to\alpha_3$	$\alpha_3\to\alpha_4$	$\alpha_4\to\alpha_5$	$\alpha_5\to\alpha_6$	$\alpha_6\to1$
$1\to4_{(0)}$ (0)+0,GS	$4\to3_{(0)}$ (0)+30,GS	$3\to2_{(0)}$ RV				
		$3\to6_{(1)}$ (1)+30,GS	$6\to5_{(0)}$ (1)+30,GS	$5\to2_{(30)}$ (31)+18,GS	$2\to7_{(0)}$ (31)+18,GS	$7\to1_{(18)}$ **TRVL=57** **JO**
				$5\to7_{(39)}$ (40)+22,JO		
			$6\to7_{(31)}$ (32)+52,JB			
			$6\to2_{(32)}$ RV, JO			
		$3\to5_{(23)}$ RV				
		$3\to7_{(23)}$ (23)+30,JO				
	$4\to6_{(13)}$ (13)+0,GS	$6\to5_{(0)}$ (13)+0,GS	$5\to3_{(0)}$ (13)+0,GS	$3\to2_{(0)}$ (13)+18,GS	$2\to7_{(0)}$ (13)+18,GS	$7\to1_{(18)}$ **TRVL=31** **JO**
				$3\to7_{(23)}$,JO		
			$5\to2_{(30)}$,JO			
		$6\to7_{(31)}$,JO				
	$4\to2_{(34)}$ RV,JO					
	$7\to2_{(0)}$ RV					
$1\to7_{(0)}$ (0)+22,GS	$7\to6_{(0)}$ (0)+22,GS	$6\to5_{(0)}$ (0)+22,GS	$5\to3_{(0)}$ (0)+22,GS	$3\to2_{(0)}$ (0)+22,GS	$2\to4_{(25)}$ (25)+0,GS	$4\to1_{(0)}$ **TRVL=25** **JO**
				$3\to4_{(10)}$,JO		
			$5\to2_{(30)}$, JO			
		$6\to4_{(9)}$ (9)+22,JB				
		$6\to2_{(32)}$ RV,JO				
	$7\to3_{(12)}$ (12)+52,JB					
	$7\to5_{(21)}$ RV					
	$7\to4_{(35)}$, JO					
$1\to6_{(26)}$ STOP						

So, the optimum tour is $\{1\to7\to6\to5\to3\to2\to4\to1\}$, and optimal solution = bias + trial value = 169 + 25 = 194.

We have seen in the case usual TSP that the adjacency approach is better than the path approach in the context of the lexisearch. But, for the Restricted TSPs, the restriction checking is much more easier in path approach (PLS) than in adjacency approach while forming the tour. In fact, in the adjacency approach (ALS), we can check the feasibility of a solution only after a complete permutation is obtained. A comparative study of both these approaches is carried out for two sets of randomly generated problems of sizes 20 and 25 (see **Table-5.5**).

Table-5.5: Comparative study of PLS and ALS for TSP-PC.

N	Seed	Sol	Time		N	Seed	Sol.	Time	
			PLS	ALS				PLS	ALS
20	**930**	153	0.06	0.44	25	**930**	173	1.10	>60
	160	170	0.00	0.11		**160**	204	0.44	>60
	681	178	0.05	0.06		**681**	171	0.11	>60
	418	198	0.33	0.11		**418**	198	7.80	>60
	522	178	0.00	0.05		**522**	181	1.21	>60
	Mean		**0.09**	**0.15**		**Mean**		**2.13**	----
	S.D.		**0.12**	**0.15**		**S.D.**		**2.86**	

It is seen from the **Table-5.5** that though the path approach was worse for the usual TSP than adjacency approach, but it is better for the TSP-PC. Hence, we consider the path approach for this problem and the quasi-exact (QE) approach is obtained by modifying the bound in the path approach described above.

In order to examine the possible effect of number of constraints on the value of solution and time taken, we have considered different number of precedence constraints for a same problem (see **Table-5.6**). Here, as the number of constraints increases, the only additional number of constraints is added to the old existing constraints, in spite of taking a new set of constraints. Hence, **as expected**, it is seen from the **Table-5.6** that as the number of constraints increases the solution value, for a same problem, also monotonically non-decreases. In the context of time taken to solve the problem, one may expect that as the number of constraints increases, the time taken should decrease. However, there is no logical reason why such monotonicity is expected. Time taken and the value of the solution depend upon particular set of constraints, not the numbers. So, as the number of the constraints increases, if one considers a new set of constraints, then one **can not expect** about the monotonicity of the solution value also.

Table-5.6: - Comparative study of PLS considering various number of constraints for the TSP-PC

N	Seed	Const=4		Const=5		Const=6		Const=7		Const=8		Usual TSP	
		Sol.	Time	Sol.	Time	Sol.	Time	Sol.	Time	Sol.	Time	Sol.	Time
30	930	169	7.52	176	11.48	176	9.45	176	8.07	176	6.53	156	2.36
	160	191	6.87	192	6.98	195	12.03	195	13.73	206	7.75	184	2.85
	681	186	5.82	188	25.38	188	21.26	188	21.97	188	11.64	166	2.64
	418	193	2.15	196	3.46	203	11.59	203	9.23	203	7.96	187	33.61
	522	200	3.51	200	2.85	200	1.15	208	1.70	208	1.32	199	3.63
	Mean		5.20		10.03		11.10		10.94		7.04		9.02
	S.D.		2.04		8.27		6.42		6.73		3.33		12.30
34	930	178	32.68	179	34.50	180	34.44	180	26.15	181	19.66	174	7.36
	160	154	28.67	154	19.77	154	18.23	154	14.17	154	12.08	151	13.73
	681	210	13.13	221	45.15	221	35.32	223	42.02	227	59.93	209	46.85
	418	158	4.62	160	8.29	178	10.76	178	7.52	178	5.87	153	8.68
	522	186	30.97	186	22.68	186	19.39	186	23.73	186	21.42	179	9.56
	Mean		22.01		27.38		23.63		22.72		23.79		17.24
	S.D.		11.13		13.23		9.66		11.74		18.91		14.96

5.3. SEQUENTIAL CONSTRUCTIVE SAMPLING APPROACH AND ILLUSTRATION

At first, we set the cost of some appropriate edges to as large as possible (for simplicity, we take 999) due to the precedence constraints. In this case, k of **equation 3.1** is the number of 'legitimate' nodes in each row having value less than 999 and obey the constraints. So, while considering the 'legitimate' nodes we check whether any precedence constraint is violated. If it is violated, then we select another 'legitimate' node. For checking the precedence constraints we follow the rule described in section 5.2.

For the sequential constructive sampling algorithm of the TSP-PC, following modifications are carried out in the sequential constructive sampling for usual TSP, described in section 3.4.

Step 0: - Form the 'alphabet table', based on 'modified' cost matrix after incorporating all precedence constraints. Initialize the 'current trial solution value' to a large number.

5.3.1 ILLUSTRATION

Let us illustrate the process through the same example considered earlier, as given in **Table-5.1**in Section 5.2.1. The 'alphabet tables' for building the tour and calculating the bound are given in **Table-5.7** and **Table-5.8** respectively.

TABLE-5.7: - The Alphabet Table

	N-V	N-V	N-V	N-V	N-V	N-V	N-V
1	7-8	4-9	6-63	3-99	1-999	2-999	5-999
2	7-20	4-46	1-51	3-86	2-999	5-999	6-999
3	2-5	4-16	5-28	7-28	6-35	1-100	3-999
4	3-11	1-20	2-45	7-49	6-53	5-59	4-999
5	3-33	2-63	4-65	7-72	1-999	5-999	6-999
6	5-21	4-31	7-52	2-53	3-89	1-999	6-999
7	2-31	3-43	5-52	1-58	6-60	4-67	7-999

TABLE-5.8: - The Alphabet Table for Bound Calculation

Sl.	Value	Cum. Value	Row	Col.	Sl.	Value	Cum. Value	Row	Col.
1	5	5	3	2	**19**	51	530	2	1
2	8	13	1	7	**20**	52	582	6	7
3	9	22	1	4	**21**	52	634	7	5
4	11	33	4	3	**22**	53	687	4	6
5	16	49	3	4	**23**	53	740	6	2
6	20	69	2	7	**24**	58	798	7	1
7	20	89	4	1	**25**	59	857	4	5
8	21	110	6	5	**26**	60	917	7	6
9	28	138	3	5	**27**	63	980	1	6
10	28	166	3	7	**28**	63	1043	5	2
11	31	197	6	4	**29**	65	1108	5	4
12	31	228	7	2	**30**	67	1175	7	4
13	33	261	5	3	**31**	72	1247	5	7
14	35	296	3	6	**32**	86	1333	2	3
15	43	339	7	3	**33**	89	1422	6	3
16	45	384	4	2	**34**	99	1521	1	3
17	46	430	2	4	**35**	100	1621	3	1
18	49	479	4	7	**36**	--	---	---	---

Set the 'current trial solution value' a large number, say $n \times \max(c_{ij})$. Now since the starting node is '1', the number of 'legitimate' nodes in 1st row, taking the precedence constraints in to account, is 4. These nodes in the order appearing in 1st row (of the alphabet table) and the probabilities, with which they are to be selected, are given below.

Legitimate Nodes	Probabilities	Cumulative Probabilities	Random Number	Node to be concatenating
7	0.400	0.400	0.572	4
4	0.300	0.700		
6	0.200	0.900		
3	0.100	1.000		

So, the partial tour will be (1,4) with value 9 and bound of this leader is 69. Since (bound + travel cost) is less than the 'current trial solution value', we accept the latest node and go ahead. Number of 'legitimate' nodes in 4th row is 5. But taking the precedence constraints in to account, the number of 'legitimate' nodes is 3. So, we proceed in a similar way. The following table gives idea how the tour is built.

Row	Legitimate Nodes	Random Number	Node to be concatenating	Bound	Partial tour	Tour value
4	3, 7, 6	0.893	6	49	(1, 4, 6)	62
6	5, 7, 3	0.335	5	33	(1, 4, 6, 5)	83
5	3, 2, 7	0.331	3	22	(1, 4, 6, 5, 3)	116
3	2, 7	0.743	7	59	(1, 4, 6, 5, 3, 7)	144
7	2	----	2	---	(1, 4, 6, 5, 3, 7, 2)	226

Since the final tour value is less than the 'current trial solution value', so replace this 'current trial solution value' by this travel cost. This completes one trial.

Repeat the whole process $5n^3$ times.

5.4 HYBRID GENETIC ALGORITHM FOR THE TSP-PC

To start with, we choose a population P_s of chromosomes, which satisfy the specified precedence constraints, evaluate them and choose the best of them as the 'current trial solution value'. The crossover, mutation and the sequential constructive operators for the TSP-PC are as follows.

(i) Crossover: - C1-Crossover operator (cf, Reeves 1993) is used for our HGA, which is discussed in the section 4.4. The C1-crossover rule also does not violate any precedence restrictions. For example, two parents P1 and P2 of length 7 with constraints node 6 < node 5, node 5 < node 2., at crossover point 3,

P1: 1 4 6 | 7 5 3 2

P2: 1 6 5 | 3 7 4 2

produce the following offspring O1 and O2 as:

O1: 1 4 6 5 3 7 2

O2: 1 6 5 4 7 3 2

The offspring O1 and O2 do not violate the precedence constraints.

(ii) Mutation: The mutation operator for our problem is as follows. Select two consecutive genes of a chromosome, say i and $(i+1)^{th}$ genes, with prescribed probability of mutation (P_m) and interchange the alleles, taking the precedence constraints into account, where i=2, 3, ...,n-1.

(iii) Sequential Constructive operator: The method is essentially same as that of the usual salesman case, described in section 4.5, with additional checking for the precedence constraints: Let us explicitly describe the algorithm.

Select a pair of chromosomes randomly.

Step 1: - Start from node '1' (i.e., current node i=1).

Step 2: - Select two 'legitimate' nodes appear immediately after node 'i' (taking the precedence constraints into account), one from each of the chromosomes'. Between these two selected nodes, select node 'j' such that the cost to go to node 'j' from node 'i' is minimum. If in one of the parent chromosomes, no 'legitimate' node is present after node 'i', then select the first 'legitimate' node (i.e., node 'j') of the other parent chromosome, which does not violate the constraints. Also, if node 'i' is the last allele of both the chromosomes (or there is no 'legitimate' node after node 'i') and there are some nodes which are still to be visited, then select the node amongst the 'legitimate' nodes which has least cost from node 'i' and does not violate any precedence constraints. The tie is broken randomly. That is, go to node 'j' next and then rename the node 'j' as node 'i'.

Step 3: - Repeat **Step 2** until all nodes have been visited.

The following example illustrates this operator.

Suppose a pair of selected chromosomes be A1 and A2, with values 255 and 308 respectively.

A1:	1	4	6	7	5	3	2
A2:	1	6	5	3	7	4	2

Select node 1, as the first allele and the 'legitimate' nodes after node 1 in A1 and A2 are 4 and 6 respectively with c_{14}=9 and c_{16}=63. Since $c_{14} < c_{16}$, accept node 4. So, the partially constructed chromosome will be (1,4).

The 'legitimate' nodes after node 4, in A1 and A2 are 6 and 2 respectively. But the node 2 violates the constraints, and there is no any other 'legitimate' node after node 4 in A2. So, there is no other option except node 6. So, accept the node 6; the partially constructed chromosome will be (1,4,6).

The 'legitimate' nodes after node 6, in A1 and A2 are 7 and 5 respectively with c_{67}=52 and c_{65}=21. Since $c_{65} < c_{67}$, accept node 5. So, the partially constructed chromosome will be (1,4,6,5).

The 'legitimate' node after node 5 in both A1 and A2 is 3. So, accept the node 3. Thus, the partially constructed chromosome will be (1,4,6,5,3).

The 'legitimate' nodes after node 3, in A1 and A2 are 2 and 7 respectively with c_{32}=5 and c_{37}=28. Since $c_{32} < c_{37}$ So, accept the node 2, and the partially constructed chromosome will be (1,4,6,5,3,2).

The 'legitimate' nodes after node 2, in both A1 and A2 are not available and there is still one node to be visited. So, check the parent A1 sequentially and accept node 7. Thus, the full chromosome will be (1,4,6,5,3,2,7) with value 199.

5.5 RELATIVE EFFICIENCY OF DIFFERENT APPROACHES

Relative efficiency analysis was carried out for four sets of randomly generated problems of sizes 30, 34, 36 and 50. Each set contains 20 randomly generated problems. For each of the problems generated, the exact optimal solution obtained by PLS, and the best solution obtained by QE and SCS are tabulated. Also are tabulated the times taken for obtaining the same. In case of HGA, the best-found solution; the solution-ratio of it to the optimal solution obtained by PLS along with the average solution-ratio, worst solution ratio and standard deviation of solution ratios of 50 runs are reported. These are reported in **Table-5.9 (a, b, c & d).** It is to be noted that for the problem of size 50 (also for the other problems discussed in next chapters), the results of the quasi-exact method is not reported as the improvement in time taken is not significant.

TABLE-5.9: - Solution values and time taken (in Seconds) by different algorithms, for twenty randomly generated problems of sizes 30, 34, 36 and 50.

(a) N=30, No. of Constraints=6 and P_s=300.

	PLS		QE			SCS			HGA						
Seed	**Sol**	**Time**	**Sol**	**Time**	**SR**	**Sol**	**Time**	**SR**	**Best Sol.**	**Best SR**	**Avg. SR**	**Worst SR**	**S.D. SR**	**Avg. Time**	**S.D Tim**
930	176	9.28	181	5.00	1.03	250	8.07	1.42	189	1.07	1.25	1.43	0.09	5.53	0.0
160	195	11.92	200	6.04	1.03	286	8.52	1.47	210	1.08	1.18	1.39	0.06	5.54	0.0
681	188	21.04	191	7.03	1.02	272	8.07	1.45	199	1.06	1.24	1.41	0.08	5.57	0.0
418	203	11.31	203	4.06	1.00	284	8.51	1.40	220	1.08	1.21	1.38	0.08	5.70	0.0
522	200	1.21	203	0.50	1.01	288	8.52	1.44	203	1.02	1.17	1.28	0.06	5.57	0.0
667	212	5.01	212	1.72	1.00	249	8.65	1.17	222	1.05	1.21	1.46	0.09	5.59	0.0
264	225	11.81	226	4.10	1.00	321	8.01	1.43	248	1.10	1.26	1.42	0.07	5.61	0.0
826	187	15.42	192	2.35	1.03	320	7.95	1.71	202	1.08	1.22	1.41	0.08	5.65	0.0
15	172	0.72	177	0.35	1.03	261	7.85	1.52	192	1.12	1.23	1.51	0.09	5.61	0.0
85	225	58.07	233	26.06	1.04	314	8.19	1.40	225	1.00	1.14	1.28	0.06	5.59	0.7
855	199	55.69	204	16.32	1.03	297	8.62	1.49	212	1.07	1.19	1.39	0.07	5.51	0.0
334	169	0.85	169	0.22	1.00	264	7.51	1.56	169	1.00	1.23	1.44	0.08	5.51	0.5
597	197	84.50	197	32.06	1.00	314	8.12	1.59	215	1.09	1.23	1.44	0.10	5.59	0.0
493	182	4.21	182	2.01	1.00	262	8.23	1.44	188	1.03	1.20	1.31	0.06	5.58	0.0
348	227	2.64	232	2.20	1.02	294	8.01	1.30	227	1.02	1.18	1.32	0.06	5.57	0.0
19	175	5.92	181	3.37	1.03	256	6.93	1.46	189	1.08	1.26	1.48	0.09	5.18	0.0
802	175	1.32	175	0.31	1.00	263	6.88	1.50	185	1.06	1.17	1.37	0.08	5.18	0.0
795	182	5.92	188	3.46	1.03	262	9.49	1.44	189	1.04	1.28	1.49	0.11	5.09	0.0
102	213	1.76	213	0.55	1.00	323	10.22	1.52	217	1.02	1.14	1.31	0.06	5.06	0.0
28	206	18.91	210	6.23	1.02	266	9.51	1.29	213	1.03	1.22	1.33	0.06	5.08	0.0
Mean		**16.38**		**6.20**	**1.02**		**8.29**	**1.45**		**1.06**	**1.21**	**1.39**		**5.47**	
S.D.		**22.26**		**8.46**	**0.01**		**0.78**	**0.11**		**0.03**	**0.04**	**0.07**		**0.21**	

(b) N=34, No. of Constraints=6 and P_s=340.

Seed	PLS		QE			SCS			HGA						
	Sol	Time	Sol	Time	SR	Sol	Time	SR	Best Sol.	Best SR	Avg. SR	Worst SR	S.D. SR	Avg. Time	S.D Tim
930	180	34.83	185	13.51	1.03	282	14.67	1.57	193	1.07	1.27	1.45	0.09	8.17	0.0
160	154	18.40	161	7.15	1.05	284	15.38	1.84	165	1.07	1.26	1.43	0.12	8.20	0.0
681	221	35.70	221	8.29	1.00	313	15.48	1.42	235	1.06	1.23	1.43	0.07	8.59	0.0
418	178	10.87	183	4.45	1.03	287	14.50	1.61	204	1.15	1.23	1.46	0.10	8.29	0.0
522	186	19.50	186	6.09	1.00	277	14.88	1.49	195	1.05	1.27	1.42	0.08	8.77	0.0
667	215	1.45	215	0.44	1.00	343	15.06	1.60	231	1.07	1.27	1.39	0.10	8.23	3.6
264	209	5.80	212	2.21	1.01	317	15.32	1.52	219	1.05	1.20	1.36	0.09	8.35	0.0
826	133	3.62	137	1.46	1.03	223	15.21	1.68	141	1.06	1.26	1.38	0.13	8.41	0.0
15	204	40.66	209	25.06	1.02	305	14.31	1.50	215	1.04	1.16	1.26	0.06	8.39	0.0
85	173	9.83	174	2.35	1.01	292	14.96	1.69	174	1.01	1.16	1.31	0.10	8.36	0.0
855	169	5.59	174	1.95	1.03	289	14.65	1.71	190	1.12	1.23	1.41	0.10	8.42	0.0
334	183	52.80	188	21.52	1.03	280	15.12	1.53	204	1.12	1.25	1.39	0.08	8.45	0.0
597	195	5.80	196	1.10	1.01	279	15.01	1.43	211	1.08	1.25	1.37	0.08	8.58	0.0
493	160	13.06	160	3.78	1.00	276	15.21	1.73	160	1.00	1.30	1.46	0.13	8.52	0.0
348	203	9.82	203	2.18	1.00	304	14.95	1.50	230	1.13	1.28	1.42	0.08	8.49	0.0
19	192	13.06	192	4.93	1.00	263	14.95	1.37	197	1.03	1.29	1.45	0.07	8.59	0.0
802	212	11.38	217	4.62	1.02	288	15.03	1.36	230	1.06	1.15	1.30	0.05	8.65	0.0
795	199	192.40	202	77.35	1.02	323	16.05	1.62	216	1.07	1.28	1.54	0.09	8.45	0.0
102	255	49.67	255	21.45	1.00	326	15.98	1.28	242	1.05	1.30	1.56	0.09	8.55	0.0
28	155	2.46	155	0.54	1.00	274	14.99	1.77	165	1.07	1.25	1.51	0.09	8.45	0.0
Mean		**26.83**		**10.52**	**1.01**		**15.09**	**1.56**		**1.07**	**1.24**	**1.42**		**8.45**	
S.D.		**41.01**		**16.97**	**0.01**		**0.42**	**0.14**		**0.04**	**0.04**	**0.07**		**0.15**	

(c) N=36, No. of Constraints=7 and P_s=400.

Seed	PLS		QE			SCS			HGA						
	Sol	Time	Sol	Time	SR	Sol	Time	SR	Best Sol.	Best SR	Avg. SR	Worst SR	S.D. SR	Avg. Time	S.D Tim
930	184	83.32	184	33.01	1.00	324	20.71	1.76	200	1.09	1.27	1.52	0.10	11.35	0.0
160	200	661.40	208	329.8	1.04	301	20.38	1.50	218	1.09	1.23	1.40	0.07	11.92	0.0
681	191	33.12	192	8.30	1.01	332	20.10	1.74	208	1.09	1.31	1.45	0.09	11.61	0.0
418	172	162.40	172	46.68	1.00	301	19.11	1.75	179	1.04	1.32	1.53	0.12	11.70	0.0
522	207	23.29	207	7.69	1.00	300	19.78	1.45	218	1.05	1.18	1.38	0.08	11.57	0.0
667	205	295.56	206	80.46	1.00	318	19.98	1.55	224	1.09	1.24	1.34	0.06	11.05	0.0
264	204	45.60	208	25.29	1.02	301	19.07	1.48	229	1.12	1.26	1.43	0.07	10.97	0.0
826	162	1058.4	169	240.22	1.04	267	20.12	1.65	187	1.15	1.29	1.50	0.08	11.21	0.0
15	188	117.60	189	58.01	1.01	280	20.65	1.49	204	1.09	1.25	1.51	0.10	11.78	0.0
85	192	12.59	194	5.29	1.01	292	20.13	1.52	204	1.08	1.24	1.44	0.09	11.12	0.0
855	176	129.38	186	46.01	1.06	305	19.65	1.73	193	1.10	1.29	1.49	0.09	11.32	0.0
334	197	20.30	197	5.03	1.00	285	19.49	1.45	207	1.05	1.19	1.38	0.06	10.97	4.0
597	174	155.28	180	48.01	1.03	290	19.98	1.67	185	1.06	1.27	1.48	0.10	12.01	2.1
493	173	47.65	173	17.06	1.00	299	19.82	1.73	189	1.09	1.27	1.47	0.13	11.50	0.0
348	200	1.26	200	0.48	1.00	306	20.13	1.53	213	1.07	1.24	1.42	0.09	11.17	0.0
19	222	77.81	228	42.21	1.03	343	20.72	1.55	259	1.17	1.28	1.42	0.06	11.06	0.0
802	163	8.15	163	3.54	1.00	307	22.13	1.88	170	1.04	1.28	1.63	0.14	10.97	0.0
795	173	32.45	173	12.00	1.00	279	22.16	1.61	187	1.08	1.24	1.59	0.10	11.05	0.0
102	203	283.82	212	190.85	1.04	317	25.36	1.56	225	1.11	1.24	1.41	0.07	11.16	0.0
28	166	15.70	174	6.23	1.05	305	20.76	1.84	192	1.16	1.37	1.53	0.09	11.06	0.0
Mean		**163.25**		**60.31**	**1.02**		**20.51**	**1.62**		**1.09**	**1.26**	**1.47**		**11.33**	
S.D.		**254.41**		**86.83**	**0.02**		**1.36**	**0.13**		**0.04**	**0.04**	**0.07**		**0.33**	

(d) N=50, No. of Constraints=10 and P_s=400.

Seed	PLS		SCS			HGA						
	Sol.	Time	Sol.	Time	SR	Best Sol.	Best SR	Avg. SR	Worst SR	S.D. SR	Avg. Time	S.D. Time
930	232*	3600.00	360	94.80	1.55	232	1.00	1.19	1.42	0.11	21.60	2.28
160	198*	3600.00	334	96.09	1.69	199	1.01	1.23	1.41	0.09	22.01	0.04
681	194	1279.20	332	96.96	1.71	241	1.24	1.37	1.57	0.10	21.95	0.66
418	191	2802.20	376	92.99	1.97	205	1.07	1.30	1.50	0.16	22.05	0.08
522	239	2737.20	376	91.97	1.57	262	1.10	1.29	1.48	0.08	22.64	0.04
667	248*	3600.00	368	93.77	1.48	248	1.00	1.12	1.32	0.09	21.97	3.09
264	240*	3600.00	377	94.11	1.57	251	1.05	1.22	1.51	0.11	23.23	3.86
826	251*	3600.00	350	97.03	1.39	251	1.00	1.14	1.33	0.10	21.06	2.31
15	222	1161.60	376	96.07	1.69	228	1.03	1.26	1.53	0.10	22.63	0.03
85	204	1190.40	327	97.52	1.60	220	1.08	1.36	1.56	0.13	22.05	0.03
855	241*	3600.00	380	95.04	1.58	247	1.03	1.24	1.54	0.11	22.93	0.08
334	205*	3600.00	363	97.97	1.77	205	1.00	1.26	1.52	0.10	21.01	2.30
597	185	1647.60	369	93.89	1.99	216	1.17	1.38	1.59	0.15	22.56	3.61
493	203*	3600.00	370	99.07	1.82	227	1.12	1.38	1.51	0.12	22.03	0.03
348	249	1128.00	415	98.01	1.67	249	1.00	1.25	1.43	0.09	21.07	2.25
19	247	1534.47	387	98.20	1.57	267	1.08	1.32	1.68	0.13	21.61	0.03
802	176	3131.88	369	99.24	2.10	191	1.09	1.40	1.71	0.16	21.82	0.14
795	218	1490.93	364	98.51	1.67	229	1.05	1.22	1.55	0.09	21.84	0.03
102	208	2433.52	362	99.01	1.74	234	1.12	1.36	1.63	0.12	21.86	0.03
28	215	2237.27	353	98.51	1.64	235	1.09	1.26	1.47	0.09	21.06	1.98
Mean		**2578.71**		**96.44**	**1.69**		**1.07**	**1.28**	**1.51**		**21.95**	
S.D.		**993.29**		**2.18**	**0.17**		**0.06**	**0.08**	**0.10**		**0.61**	

* The solution, which is obtained in 3600 seconds, is reported here.

N.B. A part of the above study has been published in the **Journal of the Operational Research Society of India** (OPSEARCH), vol. 38, No. 3, pp. 299-318, 2001.

CHAPTER VI

THE TRAVELLING SALESMAN PROBLEM WITH FIXED POSITION CONSTRAINTS

6.1 INTRODUCTION

In this chapter we will discuss the Travelling Salesman Problem with Fixed Position Constraints (TSP-FPC, for short), in which each admissible tour is to satisfy k fixed position constraints; prescribed positions p1, $p_2,\ldots,p_k$ along the tour are to be occupied by specified nodes q_1, q_2, ...,q_k respectively.

Schematically, let p_1, p_2, ...,p_k be the k fixed positions in which nodes q_1, q_2, q_k are to ~~be~~ appeare~~d~~ symbolically $\begin{pmatrix} p1, p2, \ldots\ldots, pk \\ q1, q2, \ldots\ldots, qk \end{pmatrix}$. Solution set is {1→... →q_1→... →q_2→...→q_k→1}. The problem is to find an optimal solution.

The TSP-FPC has practical applications as well explained by Scroggs and Therp (1972), and Das (1976).

6.2 LEXISEARCH APPROACH (PATH APPROACH) AND ILLUSTRATION

At first set the cost of (q_i, q_j) to as large as possible in the given cost matrix C, for all i, j=0, 1, 2,....,k; i≠j, if $|p_i - p_j| > 1$ and $p_0=q_0=1$. If, for any i,j, $|p_i - q_j|=1$, it is obvious that $q_i \rightarrow q_j$ or $q_j \rightarrow q_i$ as the case may be in every acceptable solution. Henc~~e~~ one can coalesce the nodes q_i and q_j into a single node and reduce the total number of nodes to be considered by '1'. It is to be noted that the node '1' will appear in 0th position of the tour. Furthermore, we examine at each stage of concatenation of the present leader with the node, whether it violates any of the constraints.

The algorithm is the modification of some steps in the lexisearch algorithm for the usual TSP, described in section 3.3. First we assume that the nodes to be appeared in the fixed positions are already present in the incomplete tour, though in reality they may not be present there. Then whenever a prescribed fixed position appears, we accept the node for further computation (i.e., bound calculation etc.). Later on, this assumption is assumed in the sequential constructive sampling (SCS) approach also.

Now, for the lexisearch algorithm, following steps are replaced in the lexisearch algorithm for usual TSP, described in section 3.3. Also **step 1** is replaced by **step 1(b)**.

Step 0: - Form the 'alphabet table', based on the 'reduced' (i.e., bias-removal) cost matrix **after incorporating all fixed position constraints**. Initialize the 'current trial solution value' to a large number. Since our starting node is '1', we start our computation from 1st row. Put r=1.

Step 1(a): - If any prescribed position appears, then accept the corresponding node (say, node α) and compute the cost of travelling. If the travel cost is greater than or equal to the 'current trial solution value', go to **step 8**, *else*, go to **step 3**, *else*, go to **step 1(b)**.

Step 6: - If the (Bound + Travel cost) is greater than or equal to the 'current trial solution value', drop the city added in **step 1** and increment r by 1, and then go to **step 7**; *else,* go to **step 8**.

6.2.1 ILLUSTRATION

Working of this algorithm is explained through the seven-city example; with inter city travel costs as given in **Table-3.1** with constraints $\begin{pmatrix}3\\5\end{pmatrix}$ i.e., node 5 is to be placed at 3rd position of the tour. So, we set the cost of the edges (1, 5) and (5, 1) to a large number. **Table-6.1** and **Table-6.2** give the modified cost matrix with bias and the reduced cost matrix respectively, while **Table-6.3** and **Table-6.4** give the 'alphabet table' and the logic-flow of the algorithm at various stages, which sequentially records the intermediate results, with decision taken (i.e., remarks) at these steps in every column respectively. In addition to the symbols given in section 3.2.2, the following symbol is also used therein.

FIX: Go to the prescribed city.

TABLE-6.1: - The Modified Cost Matrix with Bias

i \ j	1	2	3	4	5	6	7	Row Min
1	999	75	99	9	999	63	8	**8**
2	51	999	86	46	88	29	20	**20**
3	100	5	999	16	28	35	28	**5**
4	20	45	11	999	59	53	49	**11**
5	999	63	33	65	999	76	72	**33**
6	36	53	89	31	21	999	52	**21**
7	58	31	43	67	52	60	999	**31**
Col. Min	**9**	**0**	**0**	**1**	**0**	**9**	**0**	

Total Bias = 129 + 19 = 148.

TABLE-6.2:- The Reduced Cost Matrix

i \ j	1	2	3	4	5	6	7
1	982	67	91	0	991	46	0
2	22	979	66	25	68	0	0
3	86	0	994	10	23	21	23
4	0	34	0	987	48	33	38
5	957	30	0	31	966	34	39
6	6	32	68	9	0	969	31
7	18	0	12	35	21	29	968

TABLE-6.3: - The Alphabet Table

	N-V	N-V	N-V	N-V	N-V	N-V	N-V
1	4-0	7-0	6-46	2-67	3-91	1-982	5-991
2	6-0	7-0	1-22	4-25	3-66	5-68	2-979
3	2-0	4-10	6-21	5-23	7-23	1-86	3-994
4	1-0	3-0	6-33	2-34	7-38	5-48	4-987
5	3-0	2-30	4-31	6-34	7-39	1-957	5-966
6	5-0	1-6	4-9	7-31	2-32	3-68	6-949
7	2-0	3-12	1-18	5-21	6-29	4-35	7-968

TABLE-6.4: SEARCH TABLE

$1\to\alpha_1$	$\alpha_1\to\alpha_2$	$\alpha_2\to\alpha_3$ (=5)	(5=)$\alpha_3\to\alpha_4$	$\alpha_4\to\alpha_5$	$\alpha_5\to\alpha_6$	$\alpha_6\to1$
$1\to4_{(0)}$ (0)+0,GS	$4\to3_{(0)}$ (0)+30,FIX	$3\to5_{(23)}$ (23)+3,GS	$5\to2_{(30)}$ (53)+24,GS	$2\to6_{(0)}$ (53)+24,GS	$6\to7_{(31)}$ (84)+18,GS	$7\to1_{(18)}$ **TRVL=102 JO**
				$2\to7_{(0)}$ (53)+24,GS	$7\to6_{(29)}$ (82)+6,GS	$6\to1_{(6)}$ **TRVL=88 JO**
			$5\to6_{(34)}$ (57)+6,GS	$6\to7_{(31)}$ JO		
			$5\to7_{(39)}$ (62)+6,GS	$7\to2_{(0)}$ (62)+6,GS	$2\to6_{(0)}$ (62)+6,GS	$6\to1_{(6)}$ **TRVL=68 JO**
				$7\to6_{(29)}$ JO		
	$4\to6_{(33)}$ (33)+0,FIX	$6\to5_{(0)}$ (33)+0,GS	$5\to3_{(0)}$ (33)+0,GS	$3\to2_{(0)}$ (33)+18,GS	$2\to7_{(0)}$ (33)+18,GS	$7\to1_{(18)}$ **TRVL=51 JO**
				$3\to7_{(23)}$, JO		
			$5\to2_{(30)}$,JO			
	$4\to2_{(34)}$ (34)+33,JB					
	$4\to7_{(38)}$ (38)+0,FIX	$7\to5_{(21)}$ JO				
$1\to7_{(0)}$ (0)+0,GS	$7\to2_{(0)}$ (0)+22,FIX	$2\to5_{(68)}$ JO				
	$7\to3_{(12)}$ (12)+30,FIX	$3\to5_{(23)}$ (35)+36,JO				
	$7\to6_{(29)}$ (29)+22,JO					
$1\to6_{(46)}$ (46)+0,GS	$6\to4_{(9)}$ JO					
$1\to2_{(67)}$ STOP						

So, the optimum tour is $\{1\to4\to6\to5\to3\to2\to7\to1\}$, and optimal solution = 169 + 51 = 199.

The adjacency approach is not easy for this problem also. Hence we do not try that approach.

In order to examine the possible effect of number of constraints on the value of solution and time taken, we have considered different number of fixed position constraints for a same problem (see **Table-6.5**). Here, as the number of constraints increases, the only additional number of constraints is added to the old existing constraints, in spite of taking a new set of constraints. Hence, as **expected**, it is seen from the **Table-6.5** that as the number of constraints increases the solution value, for a same problem, also monotonically non-decreases. However, in the context of time taken to solve the problem, one can not say anything about the monotonicity in time taken to solve the problem. Of course, if one looks at the mean and standard deviation of the time taken for the TSP-FPC, one can conclude that as the number of constraints increases the mean and standard deviation of the time taken also increase monotonically. Time taken and the value of the solution depend upon particular set of constraints, not the numbers. So, as the number of the constraints increases, if one considers a new set of constraints, then one **can not expect** about the monotonicity of the solution value also.

Table-6.5: - A comparative study of PLS considering various number of constraints for TSP-FPC

N	Seed	Const=4		Const=5		Const=6		Const=7		Usual TSP	
		Sol.	Time	Sol.	Time	Sol.	Time	Sol.	Time	Sol.	Time
30	**930**	232	3.19	254	5.55	265	7.47	289	15.27	**156**	**2.36**
	160	247	3.79	271	5.05	274	4.40	316	42.62	**184**	**2.85**
	681	224	0.22	255	1.70	267	3.29	278	2.75	**166**	**2.64**
	418	231	2.36	250	4.17	259	10.82	297	17.08	**187**	**33.61**
	522	253	2.08	256	2.09	271	5.66	285	5.93	**199**	**3.63**
	Mean		**2.33**		**3.71**		**6.33**		**16.73**		**9.02**
	S.D.		**1.22**		**1.55**		**2.64**		**14.03**		**12.30**
34	**930**	243	9.50	257	18.02	268	32.02	268	33.40	**174**	**7.36**
	160	197	7.31	211	12.41	220	28.23	231	33.39	**151**	**13.73**
	681	255	8.84	265	2.64	297	14.78	311	28.67	**209**	**46.85**
	418	205	0.50	236	2.64	254	21.97	293	202.73	**153**	**8.68**
	522	228	3.78	232	4.61	251	12.14	267	34.55	**179**	**9.56**
	Mean		**5.97**		**8.06**		**21.83**		**66.548**		**17.24**
	S.D.		**3.38**		**6.15**		**7.60**		**68.121**		**14.96**

6.3 SEQUENTIAL CONSTRUCTIVE SAMPLING APPROACH AND ILLUSTRATION

At first we set the cost of some appropriate edges to large values due to the fixed position constraints. Furthermore, we examine at each stage of concatenation of the present leader with the node, whether it violates any of the constraints. In this case, k of **equation 3.1** is the number of 'legitimate' nodes in each row having value less than 999 which obey the constraints.

The following are the modification carried out in the SCS algorithms for the usual TSP discussed in section 3.4. Also, the existing **step 2** is replaced by **step 2(b),** which follows the following **step 2(a)**.

Step 0: - Form the 'alphabet table', based on 'modified' cost matrix after incorporating all fixed position constraints. Initialize the 'current trial solution value' to a large number.

Step 2(a): - If any prescribed position appears, then accept the corresponding node and compute the cost of travelling. If the travel cost is greater than or equal to the 'current trial solution value', go to **step 6**, *else,* go to **step 4**, *else*, go to **step 2(b)**.

6.3.1 ILLUSTRATION

Let us illustrate the process through the same example considered earlier, as given in section 6.2.1 in **Table-6.1**. The 'alphabet tables' for building the tour and calculating the bound are given in **Table-6.6** and **Table-6.7** respectively.

TABLE-6.6: - The Alphabet Table

	N-V	N-V	N-V	N-V	N-V	N-V	N-V
1	7-8	4-9	6-63	2-75	3-99	1-999	5-999
2	7-20	6-29	4-46	1-51	3-86	5-88	2-999
3	2-5	4-16	5-28	7-28	6-35	1-100	3-999
4	3-11	1-20	2-45	7-49	6-53	5-59	4-999
5	3-33	2-63	4-65	7-72	6-76	1-999	5-999
6	5-21	4-31	1-36	7-52	2-53	3-89	6-999
7	2-31	3-43	5-52	1-58	6-60	4-67	7-999

TABLE-6.7: - The Alphabet Table for Bound Calculation

Sl.	Value	Cum. Value	Row	Col.	Sl.	Value	Cum. Value	Row	Col.
1	5	5	3	2	**21**	51	595	2	1
2	8	13	1	7	**22**	52	647	6	7
3	9	22	1	4	**23**	52	699	7	5
4	11	33	4	3	**24**	53	752	4	6
5	16	49	3	4	**25**	53	805	6	2
6	20	69	2	7	**26**	58	863	7	1
7	20	89	4	1	**27**	59	922	4	5
8	21	110	6	5	**28**	60	982	7	6
9	28	138	3	5	**29**	63	1045	1	6
10	28	166	3	7	**30**	63	1108	5	2
11	29	195	2	6	**31**	65	1173	5	4
12	31	226	6	4	**32**	67	1240	7	4
13	31	257	7	2	**33**	72	1312	5	7
14	33	290	5	3	**34**	75	1387	1	2
15	35	325	3	6	**35**	76	1463	5	6
16	36	361	6	1	**36**	86	1549	2	3
17	43	404	7	3	**37**	88	1637	2	5
18	45	449	4	2	**38**	89	1726	6	3
19	46	495	2	4	**39**	99	1825	1	3
20	49	544	4	7	**40**	100	1925	3	1

Set the 'current trial solution value' as large as possible. Now since the starting node is '1', the number of 'legitimate' nodes in 1st row is 5.

Legitimate Nodes	Probabilities	Cumulative Probabilities	Random Number	Node to be concatenating
7	0.333	0.333	0.572	4
4	0.267	0.600		
6	0.200	0.800		
2	0.133	0.933		
3	0.067	1.000		

So, the partial tour will be (1,4) with value 9 and bound of this leader is 69. Since (bound + travel cost) is less than the 'current trial solution value', we accept the latest node and go ahead. Number of 'legitimate' nodes in 4th row is 4.

Legitimate Nodes	Probabilities	Cumulative Probabilities	Random Number	Node to be concatenating
3	0.400	0.400	0.753	7
2	0.300	0.700		
7	0.200	0.900		
6	0.100	1.000		

So, the partial tour will be (1,4,7) with value 58 and the bound of this leader is 49. Since (bound + travel cost) is less than the 'current trial solution value', we accept the latest node and go ahead. Since the next position is fixed, so we accept the node 5. So, the partial tour will be (1,4,7,5) with value 110 and the bound of this leader is 33. Since (bound + travel cost) is less than the 'current trial solution value', so we go ahead. Number of 'legitimate' nodes in 5th row is 3. So, we proceed in a similar way. The following table gives idea how the tour is built.

Row	Legitimate Nodes	Random Number	Node to be concatenating	Bound	Partial tour	Tour value
5	3, 2, 6	0.932	6	22	(1, 4, 7, 5, 6)	186
6	2, 3	0.543	2	57	(1, 4, 7, 5, 6, 2)	239
2	3	-----	3	---	(1, 4, 7, 5, 6, 2, 3)	425

Since the final tour value is less than the 'current trial solution value', so replace this 'current trial solution value' by this travel cost. This completes one trial.

Repeat the whole process $5n^3$ **times.**

6.4 HYBRID GENETIC ALGORITHM FOR THE TSP-FPC

To start with, we choose a population P_s of chromosomes, which satisfy the specified fixed position constraints, evaluate them and choose the best of them as the 'current trial solution value'. The crossover, mutation and the sequential constructive operators for the TSP-FPC are as follows.

(i) **Crossover**: - C1-Crossover operator (C.f. Reeves 1993), taking fixed position constraints into account, is used for our HGA. Let us describe the crossover rule for the problem through an example. Two parents P1 and P2 of length 7 with constraint $\begin{pmatrix}3\\5\end{pmatrix}$, at crossover point 3,

P1:	1	4	6 |	5	3	7	2
P2:	1	7	3 |	5	4	6	2

produce the following offspring O1and O2 as:

O1:	1	4	6	5	3	7	2
O2:	1	7	3	5	4	6	2

(ii) Mutation*:* The mutation operator for our problem is as follows. Select two consecutive genes of a chromosome, say i and $(i+1)^{\text{th}}$ genes, with prescribed probability of mutation (P_m) and interchange the alleles, taking the fixed position constraints into account, where i=2, 3, ...,n-1. This is done by checking whether any of the genes is in fixed position. If no, then interchange the alleles, otherwise do not interchange them and select next pair and repeat the process.

(iii) Sequential Constructive operator:- This method is essentially same as that of the usual salesman case, described in section 4.5, with additional checking for the fixed position constraints. Assume that the nodes to be appeared in the fixed positions are already present in the new partially constructed chromosome, though in reality they may not be present there. Then whenever a prescribed fixed position is appeared, we accept the node. Let us explicitly describe the algorithm.

Select a pair of chromosomes randomly.

Step 1: - Start from node '1' (i.e., current node i=1).

Step 2: - Select two 'legitimate' nodes appeared immediately after node 'i' (taking the precedence constraints into account), one from each of the chromosomes'. Between these two selected nodes, select node 'j' such that the cost to go to node 'j' from node 'i' is minimum. If in one of the parent chromosomes, no 'legitimate' node is present after node 'i', then select the first 'legitimate' node (i.e., node 'j') of the other parent chromosome, which does not violate the constraints. Also, if node 'i' is the last allele of both the chromosomes (or there is no 'legitimate' node after node 'i') and there are some nodes which are still to be visited, then select the node amongst the 'legitimate' nodes which has least cost from node 'i' and does not violate any fixed position constraints. The tie is broken randomly. That is, go to node 'j' next and then rename the node 'j' as node 'i'.

Step 3: - Repeat **Step 2** until all nodes have been visited.

The following example illustrates this method.

As earlier, we shall use the same problem as before, for illustration. Suppose a pair of selected chromosomes be A1 and A2, with values 330 and 337 respectively.

A1:	1	4	7	5	2	3	6
A2:	1	2	4	5	3	7	6

Select node 1, as the first allele and the 'legitimate' nodes after node 1 in A1 and A2 are 4 and 7 respectively with c_{14}=9 and c_{12}=75. Since $c_{14} < c_{12}$, accept node 4. So, the partially constructed chromosome will be (1,4).

The 'legitimate' nodes after node 4, in A1 and A2 are 7 and 3 respectively with c_{47}=49 and c_{43}=11. Since $c_{43} < c_{47}$, accept node 3. So, the partially constructed chromosome will be (1,4,3).

Since the next position is fixed, accept the corresponding node, i.e., node 5. So, the partially constructed chromosome will be (1,4,3,5).

The 'legitimate' nodes after node 5, in A1 and A2 are 2 and 7 respectively with c_{52}=63 and c_{57}=72. Since $c_{52} < c_{57}$, accept node 2. So, the partially constructed chromosome will be (1,4,3,5,2).

The 'legitimate' nodes after node 2 in A1 and in A2 are 6 and 7 respectively with c_{26}=29 and c_{27}=20. Since $c_{27} < c_{26}$, accept node 7. So, the partially constructed chromosome will be (1,4,3,5,2,7).

The 'legitimate' nodes after node 7, in both A1 and A2 6. So, accept the node 6, and thus, the full chromosome will be (1,4,3,5,2,7,6) with value 227.

6.5 RELATIVE EFFICIENCY OF DIFFERENT APPROACHES

Relative efficiency analysis was carried out for four sets of randomly generated problems of sizes 30, 34, 36 and 50. Each set contains 20 randomly generated problems. For each of the problems generated, the exact optimal solution obtained by PLS, and the best solution obtained by QE and SCS are tabulated. Also are tabulated the times taken for obtaining the same. In case of HGA, the best-found solution; the best solution-ratio of it to the optimal solution obtained by PLS along with the average solution-ratio and standard deviation of solution-ratios of 50 runs are reported. These are reported in **Table-6.8(a, b, c & d).**

TABLE-6.8:- Solution values and time taken (in Seconds) by different algorithms, for twenty randomly generated problems of sizes 30, 34, 36 and 50.

(a) N=30, No. of Constraints=6 and P_s=225.

Seed	PLS		QE			SCS			HGA						
	Sol	Time	Sol	Time	SR	Sol	Time	SR	Best Sol.	Best SR	Avg. SR	orst SR	S.D. SR	Avg. Time	S.D. Time
930	265	7.63	277	3.96	1.05	351	6.27	1.32	294	1.11	1.42	1.74	0.11	3.27	0.02
160	274	4.45	281	1.48	1.03	379	5.21	1.38	314	1.15	1.35	1.50	0.08	3.24	0.02
681	267	3.41	281	2.25	1.05	379	5.50	1.42	281	1.05	1.24	1.48	0.11	3.26	0.02
418	259	10.93	266	2.91	1.03	318	5.05	1.23	259	1.00	1.29	1.48	0.09	3.20	0.34
522	271	5.82	271	2.91	1.00	376	5.72	1.39	293	1.08	1.28	1.50	0.11	3.26	0.02
667	302	5.02	313	4.33	1.04	415	5.52	1.37	311	1.03	1.27	1.52	0.12	3.21	0.02
264	321	13.71	321	7.75	1.00	449	5.13	1.40	347	1.08	1.23	1.44	0.09	3.25	0.03
826	252	5.66	257	2.74	1.02	342	5.21	1.36	268	1.06	1.37	1.54	0.10	3.23	0.02
15	252	3.88	252	0.98	1.00	318	6.12	1.26	269	1.07	1.34	1.74	0.15	3.27	0.02
85	299	39.28	318	20.05	1.06	364	5.95	1.22	329	1.10	1.23	1.54	0.09	3.20	0.02
855	275	20.95	276	11.21	1.00	298	6.02	1.08	303	1.10	1.28	1.40	0.07	3.27	0.02
334	273	32.23	280	10.41	1.03	393	5.62	1.44	309	1.13	1.31	1.52	0.09	3.22	0.02
597	291	6.98	295	2.21	1.01	338	5.69	1.16	312	1.07	1.24	1.47	0.10	3.23	0.02
493	235	7.64	249	2.97	1.06	397	6.16	1.69	271	1.15	1.40	1.50	0.16	3.25	0.02
348	310	6.13	320	4.01	1.03	419	6.25	1.35	343	1.11	1.20	1.38	0.06	3.26	0.02
19	269	6.86	269	1.58	1.00	419	5.66	1.56	315	1.17	1.44	1.70	0.10	3.12	0.03
802	300	18.59	316	9.02	1.05	349	6.76	1.16	355	1.18	1.31	1.55	0.08	3.26	0.03
795	258	5.40	272	1.72	1.05	358	6.96	1.39	304	1.18	1.39	1.59	0.11	3.06	0.03
102	287	1.78	300	0.85	1.05	345	5.86	1.20	306	1.07	1.22	1.40	0.09	3.12	0.03
28	317	8.31	328	2.77	1.03	334	5.99	1.05	334	1.05	1.23	1.45	0.09	3.15	0.03
Mean		**10.73**		**4.81**	**1.03**		**5.83**	**1.32**		**1.10**	**1.30**	**1.52**		**3.22**	
S.D.		**9.64**		**4.62**	**0.02**		**0.50**	**0.15**		**0.05**	**0.07**	**0.10**		**0.06**	

(b) N=34, No. of Constraints=6 and P_s=400.

Seed	PLS		QE			SCS			HGA						
	Sol	Time	Sol	Time	SR	Sol	Time	SR	Best Sol.	Best SR	Avg. SR	Worst SR	S.D. SR	Avg. Time	S.D. Time
930	268	32.90	280	14.56	1.05	414	9.45	1.54	296	1.10	1.32	1.50	0.10	7.74	0.03
160	220	28.89	233	14.39	1.06	319	7.74	1.45	258	1.17	1.58	1.73	0.14	7.69	0.03
681	297	15.11	317	12.85	1.07	417	9.45	1.40	326	1.10	1.24	1.41	0.06	7.66	0.04
418	254	22.35	272	10.71	1.07	415	9.72	1.63	285	1.12	1.42	1.67	0.10	7.69	0.03
522	251	12.41	251	4.67	1.00	375	8.85	1.49	275	1.10	1.26	1.47	0.09	7.66	0.03
667	305	17.53	305	4.78	1.00	374	8.95	1.23	334	1.10	1.25	1.52	0.09	7.68	0.04
264	292	5.59	293	1.21	1.00	392	8.62	1.34	293	1.00	1.14	1.35	0.08	9.61	2.48
826	211	15.09	228	9.52	1.08	316	9.01	1.50	251	1.19	1.37	1.57	0.10	7.63	0.03
15	279	44.76	279	28.06	1.00	413	9.62	1.48	309	1.11	1.32	1.53	0.08	7.66	0.03
85	260	17.92	260	18.59	1.00	409	9.11	1.57	285	1.10	1.33	1.54	0.11	7.67	0.02
855	262	38.28	267	25.01	1.02	430	8.92	1.64	313	1.20	1.47	1.71	0.13	7.67	0.02
334	259	13.78	278	14.21	1.07	363	7.97	1.40	297	1.15	1.35	1.55	0.09	7.68	0.04
597	278	9.04	298	3.52	1.07	417	7.99	1.50	322	1.16	1.35	1.56	0.09	7.69	0.03
493	232	21.35	235	8.21	1.01	405	8.92	1.75	265	1.14	1.47	1.72	0.11	7.71	0.03
348	258	9.43	258	2.10	1.00	418	9.16	1.62	304	1.18	1.35	1.70	0.13	7.65	0.04
19	265	35.53	269	14.69	1.02	433	10.51	1.63	292	1.10	1.33	1.74	0.14	7.69	0.03
802	298	27.42	307	8.70	1.03	438	9.46	1.47	311	1.04	1.23	1.48	0.11	7.69	0.03
795	276	31.31	285	15.04	1.03	432	10.35	1.57	316	1.15	1.30	1.50	0.08	7.68	0.04
102	284	7.38	300	4.21	1.06	427	9.65	1.50	315	1.11	1.31	1.59	0.11	7.68	0.03
28	226	7.57	236	3.43	1.04	358	8.98	1.58	236	1.04	1.28	1.48	0.10	7.67	0.03
Mean		**20.68**		**10.92**	**1.03**		**9.12**	**1.51**		**1.12**	**1.33**	**1.57**		**7.78**	
S.D.		**11.23**		**7.22**	**0.03**		**0.69**	**0.12**		**0.05**	**0.10**	**0.11**		**0.42**	

(c) N=36, No. of Constraints=7 and P_s=450.

Seed	PLS		QE			SCS			HGA						
	Sol	Time	Sol	Time	SR	Sol	Time	SR	Best Sol.	Best SR	Avg. SR	orst SR	S.D. SR	Avg. Time	S.D Time
930	291	200.92	291	48.88	1.00	436	13.07	1.50	339	1.17	1.47	1.64	0.10	10.24	0.03
160	267	98.28	278	27.46	1.04	471	13.13	1.76	335	1.26	1.53	1.82	0.12	10.11	0.03
681	287	121.55	308	57.57	1.07	427	14.06	1.49	331	1.15	1.41	1.56	0.10	10.18	0.03
418	264	128.69	288	56.57	1.09	458	13.84	1.73	321	1.22	1.50	1.69	0.11	10.12	0.03
522	296	486.97	313	146.21	1.06	439	12.75	1.48	344	1.16	1.36	1.62	0.09	10.40	0.03
667	325	322.87	325	55.05	1.00	509	14.01	1.57	359	1.11	1.25	1.38	0.07	10.01	0.35
264	314	292.77	334	145.32	1.06	455	14.56	1.45	346	1.10	1.33	1.52	0.08	10.12	0.04
826	266	176.45	286	81.02	1.08	337	14.01	1.27	333	1.25	1.43	1.63	0.08	10.41	0.02
15	274	115.23	274	31.01	1.00	472	12.95	1.72	357	1.23	1.55	1.71	0.12	10.12	0.05
85	306	367.33	310	65.12	1.01	452	12.65	1.48	341	1.11	1.28	1.42	0.08	10.13	0.04
855	254	140.42	273	55.87	1.07	404	13.06	1.59	284	1.12	1.42	1.59	0.14	10.16	0.03
334	266	11.59	274	3.12	1.03	404	14.32	1.52	306	1.15	1.32	1.51	0.11	10.12	0.03
597	243	21.43	243	6.16	1.00	385	13.67	1.58	309	1.27	1.47	1.61	0.11	10.29	0.12
493	223	174.23	223	57.11	1.00	433	13.95	1.94	268	1.20	1.52	1.70	0.18	10.23	0.03
348	310	105.66	319	25.13	1.03	451	14.01	1.45	336	1.08	1.20	1.32	0.08	10.18	0.03
19	320	232.14	342	157.93	1.07	485	14.54	1.52	380	1.19	1.34	1.56	0.07	10.25	0.02
802	272	100.64	281	33.61	1.03	410	14.65	1.51	308	1.13	1.40	1.70	0.13	10.27	0.03
795	305	152.33	305	40.20	1.00	422	14.06	1.38	367	1.20	1.35	1.51	0.06	10.12	0.02
102	269	631.68	269	186.34	1.00	438	14.25	1.63	355	1.32	1.55	1.78	0.13	10.29	2.15
28	291	1716.4	291	577.10	1.00	475	14.35	1.63	372	1.28	1.50	1.66	0.08	10.21	1.35
Mean		**279.88**		**92.84**	**1.03**		**13.79**	**1.56**		**1.18**	**1.41**	**1.60**		**10.20**	
S.D.		**361.61**		**121.88**	**0.03**		**0.62**	**0.14**		**0.07**	**0.10**	**0.12**		**0.10**	

(d) N=50, No. of Constraints=10 and P_s=400.

Seed	PLS		SCS			HGA						
	Sol.	Time	Sol.	Time	SR	Best Sol.	Best SR	Avg. SR	Worst SR	S.D. SR	Avg. Time	S.D. Time
930	343	3020.08	582	66.20	1.70	425	1.24	1.52	1.75	0.11	20.40	0.03
160	371*	3600.00	483	65.12	1.30	371	1.00	1.25	1.51	0.10	21.03	2.07
681	420*	3600.00	556	67.03	1.32	420	1.00	1.23	1.47	0.08	20.03	2.87
418	410*	3600.00	480	65.01	1.17	419	1.02	1.23	1.46	0. 11	19.67	1.87
522	375	2422.40	581	63.98	1.55	431	1.15	1.34	1.49	0.16	20.01	1.10
667	504*	3600.00	653	67.01	1.30	505	1.00	1.13	1.38	0.08	20.67	0.03
264	422	2557.80	540	65.09	1.28	463	1.10	1.30	1.54	0.08	21.01	0.08
826	379*	3600.00	562	66.12	1.48	379	1.00	1.23	1.59	0.11	20.56	0.08
15	403	2126.60	558	66.05	1.38	404	1.00	1.21	1.49	0.11	20.37	0.04
85	373	1325.80	532	63.09	1.43	394	1.06	1.28	1.57	0.11	20.97	0.03
855	369*	3600.00	537	67.20	1.46	409	1.11	1.32	1.66	0.10	20.53	0.08
334	429*	3600.00	550	67.45	1.28	429	1.00	1.16	1.34	0.07	19.09	5.54
597	418*	3600.00	564	66.09	1.35	418	1.00	1.21	1.46	0.08	21.27	0.03
493	377*	3600.00	557	64.01	1.48	377	1.00	1.28	1.60	0.09	19.97	0.05
348	422*	3600.00	523	67.98	1.24	422	1.00	1.19	1.42	0.07	20.01	2.07
19	337*	3600.00	604	65.47	1.79	403	1.20	1.43	1.64	0.13	19.66	0.50
802	395*	3600.00	569	67.56	1.44	395	1.00	1.31	1.54	0.27	19.76	0.13
795	409*	3600.00	442	66.06	1.08	409	1.00	1.18	1.49	0.16	19.78	0.03
102	384*	3600.00	537	67.75	1.40	384	1.00	1.19	1.58	0.16	19.77	0.03
28	364*	3600.00	486	66.21	1.34	364	1.00	1.16	1.48	0.17	19.76	0.03
Mean		**3272.63**		**66.02**	**1.39**		**1.04**	**1.26**	**1.52**		**20.22**	
S.D.		**632.80**		**1.31**	**0.16**		**0.07**	**0.09**	**0.10**		**0.56**	

* The solution, which is obtained in 3600 seconds, is reported here.

CHAPTER VII

THE TRAVELLING SALESMAN PROBLEM WITH FIXED POSITION AND PRECEDENCE CONSTRAINTS

7.1 INTRODUCTION

In this chapter we will discuss the usual TSP with Fixed Position and Precedence Constraints (TSP-FPPC, for short). The TSP-FPPC in an appropriate union of the problems discussed in chapters V and VI. Hence the algorithms developed for the problem are the appropriate union of algorithms developed for the TSP-PC and TSP-FPC discussed in chapters V and VI respectively.

The TSP-FPPC has practical applications (cf, Scroggs and Therp 1972, and Das 1976). Despite its applicability, few papers exist about this problem in the literature. Some of them refer to the TSP-PC and some other deal with the TSP-FPC, as mentioned in chapter V and VI respectively, but very few refer to the TSP-FPPC (cf, Das 1976).

Das (1976) proposed a lexisearch algorithm for obtaining exact optimal solution to the TSP-FPPC. However the lower bound calculation method adopted therein is not efficient and hence the procedure is not practicable, when the problem sizes are even moderately large.

7.2 LEXISEARCH APPROACH (PATH APPROACH) AND ILLUSTRATION

Here all the assumptions made in the context of precedence and fixed position constraints are considered.

Now, for the lexisearch algorithm of the TSP-FPPC, following steps are replaced in the lexisearch algorithm for usual TSP, described in section 3.3. Also

step 1 is replaced by **step 1(b)**.

Step 0: - Form the 'alphabet table', based on the 'reduced' (i.e., bias-removal) cost matrix after incorporating all fixed position and precedence constraints. Initialize the 'current trial solution value' to a large number. Since our starting node is '1', we start our computation from 1st row. Put r=1.

Step 1(a): - If any prescribed position appears, then accept the corresponding node (say, node α) and compute the cost of travelling. If the travel cost is greater than or equal to the 'current trial solution value', go to **step 9**, *else,* go to **step 2**, *else*, go to **step 1(b)**.

Step 2: - If the (incomplete) word forms a sub-tour or if any prescribed constraints is violated, drop the city added in **step 1** and increment r by 1, and then go to **step 6**; *else,* go to **step 3**.

Step 6: - If the (Bound + Travel Cost) is greater than or equal to the 'current trial solution value', check whether the latest node is one of the prescribed fixed node; *else* go to **step 8**. If the latest node is one of the prescribed fixed nodes, then drop the city added in **step 1** and go to **step 9; *else*** drop the city added in **step 2,** increment l by 1, and then go to **step 7.**

7.2.1 ILLUSTRATION

Working of this algorithm is explained through the same problem as earlier, as given in **Table-3.1** with the precedence and fixed positions constraints considered in the section 5.2.1 and 6.2.1 respectively and accordingly. **Table-7.1, Table-7.2, Table-7.3** and **Table-7.4** give the modified cost matrix with bias, the reduced cost matrix, the 'alphabet table' and the 'search table' respectively.

TABLE-7.1: - The Modified Cost Matrix with Bias

i \ j	1	2	3	4	5	6	7	Row Min
1	999	999	99	9	999	63	8	8
2	51	999	86	46	999	999	20	20
3	100	5	999	16	28	35	28	5
4	20	45	11	999	59	53	49	11
5	999	63	33	65	999	999	72	33
6	999	53	89	31	21	999	52	21
7	58	31	43	67	52	60	999	31
Col. Min	9	0	0	1	0	29	0	

Total Bias = 129 + 39 = 168.

TABLE-7.2: - The Reduced Cost Matrix

i \ j	1	2	3	4	5	6	7
1	982	991	91	0	991	26	0
2	22	979	66	25	979	950	0
3	86	0	994	10	23	1	23
4	0	34	0	987	48	13	38
5	957	30	0	31	966	937	39
6	6	32	68	9	0	949	31
7	18	0	12	35	21	0	968

TABLE-7.3: - The Alphabet Table

	N-V	N-V	N-V	N-V	N-V	N-V	N-V
1	4-0	7-0	6-26	3-91	1-982	2-991	5-991
2	7-0	1-22	4-25	3-66	6-950	2-979	5-979
3	2-0	6-1	4-10	5-23	7-23	1-86	3-994
4	1-0	3-0	6-13	2-34	7-38	5-48	4-987
5	3-0	2-30	4-31	7-39	6-937	2-957	5-966
6	5-0	4-9	7-31	2-32	3-68	6-949	1-969
7	2-0	6-0	3-12	1-18	5-21	4-35	7-968

TABLE-7.4: SEARCH TABLE

$1\to\alpha_1$	$\alpha_1\to\alpha_2$	$\alpha_2\to\alpha_3(=5)$	$(5=)\alpha_3\to\alpha_4$	$\alpha_4\to\alpha_5$	$\alpha_5\to\alpha_6$	$\alpha_6\to 1$
$1\to 4_{(0)}$ (0)+0,GS	$4\to 3_{(0)}$ (0)+30,FIX	$3\to 5_{(23)}$ RV, JO				
	$4\to 6_{(13)}$ (13)+0,FIX	$6\to 5_{(0)}$ (13)+0,GS	$5\to 3_{(0)}$ (13)+0,GS	$3\to 2_{(0)}$ (13)+18,GS	$2\to 7_{(0)}$ (13)+18,GS	$7\to 1_{(18)}$ **TRVL=31 JO**
				$3\to 7_{(23)}$,JO		
			$5\to 2_{(30)}$,JO			
	$4\to 2_{(34)}$,JO					
$1\to 7_{(0)}$ (0)+22,GS	$7\to 2_{(0)}$ RV					
	$7\to 6_{(0)}$ (0)+22,FIX	$6\to 5_{(0)}$ (0)+31,JO				
	$7\to 3_{(12)}$ (12)+52, JB					
	$7\to 4_{(35)}$,JO					
$1\to 6_{(26)}$ (26)+0,GS	$6\to 4_{(9)}$, JO					
$1\to 2_{(91)}$ STOP						

So, the optimum tour is $\{1\to4\to6\to5\to3\to2\to7\to1\}$, and optimal solution = 168 + 31 = 199.

Let us now examine the possible effect of the number of constraints on the solution value and the time taken for solving the same. **Table-7.5** shows a computational report for some problems with different number of constraints. Here, the constraints are the combination of same constraints considered in the TSP-PC and TSP-FPC reported in the **Tables 5.6 and 6.5** respectively. As **expected** here also, it is seen from the **Table-7.5** that as the number of constraints increases the solution value, for a same problem, also monotonically non-decreases. In this case also, as the number of constraints increases, one **can not** say about the monotonicity of the time taken for solving the same.

Table-7.5: - A comparative study of PLS for TSP-PC, TSP-FPC and TSP-FPPC for different number of constraints.

N	Seed	No. of constraints=4						No. of constraints=5					
		TSP-PC		TSP-FPC		TSP-FPPC		TSP-PC		TSP-FPC		TSP-FPPC	
		Sol.	Time	Sol.	Time	Sol.	Time	Sol.	Time	Sol.	Time	Sol.	Time
	930	169	7.52	232	3.19	239	1.65	176	11.48	254	5.55	262	2.20
	160	191	6.87	247	3.79	259	3.90	192	6.98	271	5.05	276	2.96
	681	186	5.82	224	0.22	251	3.90	188	25.38	255	1.70	277	4.62
	418	193	2.15	231	2.36	232	3.73	196	3.46	250	4.17	250	2.03
	522	200	3.51	253	2.08	253	1.70	200	2.85	256	2.09	256	1.70
	Mean		**5.20**		**2.33**		**2.98**		**10.03**		**3.71**		**2.70**
	S.D.		**2.04**		**1.22**		**1.06**		**8.27**		**1.55**		**1.04**
30		No. of constraints=6						No. of constraints=7					
	930	176	9.45	265	7.47	286	3.08	176	8.07	289	15.27	306	4.78
	160	195	12.03	274	4.40	281	1.97	195	13.73	316	42.62	331	26.69
	681	188	21.26	267	3.29	279	5.83	188	21.97	278	2.75	338	14.33
	418	203	11.59	259	10.82	312	3.95	203	9.23	297	17.08	342	5.99
	522	200	1.15	271	5.66	281	4.23	208	1.70	285	5.93	302	4.23
	Mean		**11.10**		**6.33**		**3.81**		**10.94**		**16.73**		**11.20**
	S.D.		**6.42**		**2.64**		**1.28**		**6.73**		**14.03**		**8.56**
N	**Seed**	No. of constraints=4						No. of constraints=5					
	930	178	32.68	243	9.50	264	27.03	179	34.50	257	18.02	283	28.23
	160	154	28.67	197	7.31	208	7.96	154	19.77	211	12.41	223	9.34
	681	210	13.13	255	8.84	255	4.83	221	45.15	265	2.64	265	0.82
	418	158	4.62	205	0.50	236	3.96	160	8.29	236	2.64	258	4.12
	522	186	30.97	228	3.78	228	7.85	186	22.68	232	4.61	242	12.20
	Mean		**22.01**		**5.97**		**10.33**		**27.38**		**8.06**		**10.94**
	S.D.		**11.13**		**3.38**		**8.50**		**13.23**		**6.15**		**9.51**
34		No. of constraints=6						No. of constraints=7					
	930	180	34.44	268	32.02	294	33.12	180	26.15	268	33.40	311	49.38
	160	154	18.23	220	28.23	244	27.63	154	14.17	231	33.39	271	86.67
	681	221	35.32	297	14.78	312	13.23	223	42.02	311	28.67	337	31.80
	418	178	10.76	254	21.97	280	24.77	178	7.52	293	202.73	334	102.49
	522	186	19.39	251	12.14	294	62.01	186	23.73	267	34.55	306	124.24
	Mean		**23.63**		**21.83**		**32.15**		**22.72**		**66.548**		**78.92**
	S.D.		**9.66**		**7.60**		**16.28**		**11.74**		**68.121**		**33.95**

7.3 SEQUENTIAL CONSTRUCTIVE SAMPLING APPROACH AND ILLUSTRATION

Here all the assumptions made in the context of precedence and fixed position constraints are considered. In this case, k of **equation 3.1** is the number of ‘legitimate’ nodes in each row having value less than 999 which obey the constraints. In this case, while considering the 'legitimate' nodes, we check only whether any precedence constraint is violated. If it is violated, then we select next 'legitimate' node.

Now, for the algorithm, following steps are replaced in the sequential constructive sampling for usual TSP, described in section 3.4. Also, **step 2** is replaced by **step 2(c)**.

Step 0: - Form the ‘alphabet table’, based on ‘modified’ cost matrix **after incorporating all fixed position and precedence constraints.** Initialize the ‘current trial solution value’ to a large number.

Step 2(a): - If any prescribed position appears, consider the corresponding node and compute the cost of travelling and then go to **step 2(b)**, *else,* go to **step 2(b)**.

Step 2(b): - If the travel cost is greater than or equal to the ‘current trial solution value’, go to **step 1**, *else,* check for the precedence constraints. If one of the precedence constraints is violated, go to the previous position of the tour and the go to **step 2(a)**, *else*, go to **step 4**.

7.3.1 ILLUSTRATION

Let us illustrate the process through the same example considered earlier, as given in section 7.2.1 (**Table-7.1**). The ‘alphabet tables’ for building the tour and calculating the bound are given in **Table-7.6** and **Table-7.7** respectively.

TABLE-7.6: - The Alphabet Table

	N-V	N-V	N-V	N-V	N-V	N-V	N-V
1	7-8	4-9	6-63	3-99	1-999	2-999	5-999
2	7-20	4-46	1-51	3-86	2-999	5-999	6-999
3	2-5	4-16	5-28	7-28	6-35	1-100	3-999
4	3-11	1-20	2-45	7-49	6-53	5-59	4-999
5	3-33	2-63	4-65	7-72	1-999	5-999	6-999
6	5-21	4-31	7-52	2-53	3-89	1-999	6-999
7	2-31	3-43	5-52	1-58	6-60	4-67	7-999

TABLE-7.7: - The Alphabet Table for Bound Calculation

Sl.	Value	Cum. Value	Row	Col.	Sl.	Value	Cum. Value	Row	Col.
1	5	5	3	2	**19**	51	530	2	1
2	8	13	1	7	**20**	52	582	6	7
3	9	22	1	4	**21**	52	634	7	5
4	11	33	4	3	**22**	53	687	4	6
5	16	49	3	4	**23**	53	740	6	2
6	20	69	2	7	**24**	58	798	7	1
7	20	89	4	1	**25**	59	857	4	5
8	21	110	6	5	**26**	60	917	7	6
9	28	138	3	5	**27**	63	980	1	6
10	28	166	3	7	**28**	63	1043	5	2
11	31	197	6	4	**29**	65	1108	5	4
12	31	228	7	2	**30**	67	1175	7	4
13	33	261	5	3	**31**	72	1247	5	7
14	35	296	3	6	**32**	86	1333	2	3
15	43	339	7	3	**33**	89	1422	6	3
16	45	384	4	2	**34**	99	1521	1	3
17	46	430	2	4	**35**	100	1621	3	1
18	49	479	4	7	**36**	--	---	---	---

Set the 'current trial solution value' as large as possible. Now since the starting node is '1', the number of 'legitimate' nodes in 1st row, taking the precedence constraints in to account, is 4.

Legitimate Nodes	Probabilities	Cumulative Probabilities	Random Number	Node to be concatenating
7	0.400	0.400	0.572	4
4	0.300	0.700		
6	0.200	0.900		
3	0.100	1.000		

So, the partial tour will be (1,4) with value 9 and bound of this leader is 69. Since (bound + travel cost) is less than the 'current trial solution value', we accept the latest node and go ahead. Number of 'legitimate' nodes in 4th row is 5. But taking the precedence constraints in to account, the number of 'legitimate' nodes is 3.

Legitimate Nodes	Probabilities	Cumulative Probabilities	Random Number	Node to be concatenated
3	0.500	0.500	0.893	6
7	0.333	0.833		
6	0.167	1.000		

So, the partial tour will be (1,4,6) with value 62 and the bound of this leader is 49. Since (bound + travel cost) is less than the 'current trial solution value', we accept the latest node and go ahead. Since the next position is one of the fixed positions, so accept the prescribed node, i.e., node 5. So, the partial tour will be (1,4,6,5) with value 83 and the bound of this leader is 33. Since (bound + travel cost) is less than the 'current trial solution value', we go ahead. Number of 'legitimate' nodes in 5th row, taking the precedence constraints in to account, is 3. So, we proceed in a similar way. The following table gives idea how the tour is built.

Row	Legitimate Nodes	Random Number	Node to be concatenating	Bound	Partial tour	Tour value
5	3, 2, 7	0.335	3	22	(1, 4, 6, 5, 3)	116
3	2, 7	0.743	7	59	(1, 4, 6, 5, 3, 7)	144
7	2	----	2	---	(1, 4, 6, 5, 3, 7, 2)	226

Since the final tour value is less than the 'current trial solution value', so replace this 'current trial solution value' by this travel cost. This completes one trial.

Repeat the whole process $5n^3$ **times.**

7.4 HYBRID GENETIC ALGORITHM FOR THE TSP-FPPC

To start with, we choose a population P_s of chromosomes, which satisfy the specified fixed position and precedence constraints, evaluate them and choose the best of them as the 'current trial solution value'. The crossover, mutation and the sequential constructive operators for the TSP-FPPC are as follows.

(i) Crossover: - C1-Crossover operator (C.f. Reeves 1993), taking fixed position and precedence constraints into account, is used for our HGA. Let us describe the crossover rule for the problem through an example. Two parents P1 and P2 of length 7 with constraint node 6 < node 5, node 5 < node 2 and $\binom{3}{5}$, at crossover point 3,

P1:	1	7	6 |	5	4	3	2
P2:	1	6	3 |	5	7	2	4

produce the following offspring O1and O2 as:

O1:	1	7	6	5	3	2	4
O2:	1	6	3	5	7	4	2

(ii) Mutation: The mutation operator for our problem is as follows. Select two consecutive genes of a chromosome, say i and $(i+1)^{th}$ genes, with prescribed probability of mutation (P_m) and interchange the alleles, taking the fixed position and precedence constraints into account, where i=2, 3, ...,n-1. This is done by checking whether any of the genes is in fixed position or both i and $(i+1)^{th}$ alleles are in precedence constraints. If no, then interchange the alleles, otherwise do not interchange them and select next pair and repeat the process.

(iii) Sequential Constructive operator: This method is the appropriate union of the sequential constructive operator described in section 5.4 and 6.4. Let us explicitly describe the algorithm. So, whatever assumptions are made there, we consider here also.

Now, for the algorithm, select a pair of chromosomes randomly.

Step 1*:* - Start from node '1' (i.e., current node i=1).

Step 2: - If the fixed position is reached, go to the prescribed node 'j', taking the precedence constraints into account, *else,* select two 'legitimate' nodes appeared immediately after node 'i', taking the precedence constraints into account, one from each of the chromosomes. Between these two selected nodes, select node 'j' such that the cost to go to node 'j' from node 'i' is minimum. If in any one of the parent chromosomes, no any 'legitimate' node is present after node 'i', then select the first 'legitimate' node (i.e., node 'j') of the other parent chromosome, taking the constraints into account. Also, if node 'i' is the last allele (or there is no any 'legitimate' node after node 'i' which does not violate the constraints) of both the chromosomes and there are some nodes which are still to be visited, then select the node amongst the 'legitimate' nodes which does not violate the constraints and which has least cost from node 'i'. The tie is broken randomly. That is go to node 'j' next and then rename the node 'j' as node 'i'.

Step 3*:* - Repeat **Step 2** until all nodes have been visited.

The following example illustrates this method.

For illustration, we shall use the same problem as given earlier in section

7.2.1. Suppose a pair of selected chromosomes be A1 and A2, with values 343 and 226 respectively.

A1:	1	3	6	5	4	2	7
A2:	1	4	6	5	3	7	2

Select node 1, as the 1st allele and the 'legitimate' nodes after node 1 in A1 and A2 are 3 and 4 respectively with $c_{13}=99$ and $c_{14}=9$. Since $c_{14} < c_{13}$, accept node 4. So, the partially constructed chromosome will be (1,4).

The 'legitimate' nodes after node 4, in A1 and A2 are 2 and 6 respectively. But node 2 leads to a violation of a precedence constraint: node 5 < node 2. So, we consider the next 'legitimate' node in A1, i.e., node 7, with $c_{47}=49$ and $c_{46}=53$. Since $c_{47} < c_{46}$, accept node 7. So, the partially constructed chromosome will be (1,4,7).

Since the next position is fixed, accept the corresponding node, i.e., node 5, which violates the constraint: node 6 < node 5 and there is no any option of leaving node 5. So, we reject the node 5 at this stage and go back to the previous node in the partially constructed chromosome and replace that by node 6 and then visit node 5. So, the partially constructed chromosome will be (1,4,6,5).

The 'legitimate' nodes after node 5, in A1 and A2 are 2 and 3 respectively with $c_{52}=63$ and $c_{53}=33$. Since $c_{53} < c_{52}$, accept node 3. So, the partially constructed chromosome will be (1,4,6,5,3).

The 'legitimate' nodes after node 3 in A1 and in A2 are 2 and 7 respectively with $c_{32}=5$ and $c_{37}=28$. Since $c_{32} < c_{37}$, accept node 2. So, the partially constructed chromosome will be (1,4,6,5,3,2).

The 'legitimate' node after node 2, in A1 is 7, but in A2 is not available. So, accept the node 7, and thus, the full chromosome will be (1,4,6,5,3,2,7) with value 199.

7.5 RELATIVE EFFICIENCY OF DIFFERENT APPROACHES

Relative efficiency analysis was carried out for four sets of randomly generated problems of sizes 30, 34, 36 and 50. Each set contains 20 randomly generated problems. For each of the problems generated, the exact optimal solution obtained by PLS, and the best solution obtained by QE and SCS are tabulated. Also are tabulated the times taken for obtaining the same. In case of HGA, the best-found solution; the solution-ratio of it to the optimal solution obtained by PLS along with the average solution-ratio, worst solution-ratio and standard deviation of solution ratios of 50 runs are reported. These are reported in **Table-7.8(a, b, c & d).**

TABLE-7.8: - Solution values and time taken (in Seconds) by different algorithms, for twenty randomly generated problems of sizes 30, 34, 36 and 50.

(a) N=30, No. of Constraints (in each case) =6 and P_s=200.

Seed	PLS		QE			SCS			HGA						
	Sol	Time	Sol	Time	SR	Sol	Time	SR	Best Sol.	Best SR	Avg. SR	Worst SR	S.D. SR	Avg. Time	S.D. Time
930	289	4.67	312	3.57	1.08	393	5.33	1.36	431	1.18	1.36	1.54	0.12	4.38	0.03
160	281	2.41	281	0.66	1.00	386	5.60	1.37	323	1.15	1.37	1.55	0.09	4.16	0.02
681	324	21.21	350	16.58	1.08	387	5.38	1.19	363	1.12	1.29	1.66	0.15	4.18	0.02
418	289	1.20	293	0.55	1.01	385	5.66	1.33	318	1.10	1.33	1.54	0.13	4.38	0.02
522	291	4.84	301	2.36	1.03	405	6.04	1.39	314	1.08	1.29	1.55	0.12	4.34	0.02
667	349	20.83	363	6.06	1.04	448	5.95	1.28	394	1.13	1.28	1.58	0.11	4.13	0.04
264	362	42.50	376	16.23	1.04	433	5.62	1.20	391	1.08	1.20	1.40	0.10	4.32	0.02
826	286	10.66	296	3.21	1.03	386	5.31	1.35	320	1.12	1.35	1.53	0.11	4.31	0.03
15	289	25.43	299	5.56	1.03	348	5.45	1.20	306	1.06	1.21	1.49	0.11	4.39	0.03
85	334	44.40	342	11.32	1.02	407	6.06	1.22	384	1.15	1.22	1.54	0.10	4.18	0.02
855	309	38.23	309	12.10	1.00	400	5.64	1.29	331	1.07	1.29	1.44	0.09	4.21	0.02
334	299	19.64	307	4.56	1.03	363	5.95	1.21	308	1.03	1.17	1.38	0.11	4.41	0.02
597	294	3.63	294	0.78	1.00	414	6.32	1.41	315	1.07	1.31	1.53	0.10	4.01	1.24
493	301	39.6	316	12.16	1.05	382	5.62	1.27	346	1.15	1.25	1.63	0.13	4.09	1.19
348	360	24.67	366	8.98	1.02	470	5.89	1.31	400	1.11	1.29	1.63	0.07	4.32	0.03
19	307	38.77	316	13.37	1.03	446	5.75	1.45	347	1.13	1.36	1.54	0.12	4.37	0.03
802	321	138.87	338	65.67	1.05	410	5.86	1.28	372	1.16	1.29	1.49	0.09	4.16	0.02
795	275	15.08	279	5.30	1.02	410	5.09	1.49	316	1.15	1.35	1.59	0.18	4.23	0.04
102	293	3.00	293	0.77	1.00	330	5.01	1.13	319	1.09	1.24	1.42	0.37	4.05	1.37
28	344	35.69	375	32.45	1.09	421	5.06	1.22	368	1.07	1.20	1.38	0.08	4.33	0.03
Mean		**26.77**		**11.11**	**1.03**		**5.63**	**1.30**		**1.11**	**1.28**	**1.52**		**4.25**	
S.D.		**29.56**		**14.60**	**0.03**		**0.35**	**0.09**		**0.04**	**0.06**	**0.08**		**0.12**	

(b) N=34, No. of Constraints (in each case) =6 and P_s=400.

Seed	PLS		QE			SCS			HGA						
	Sol	Time	Sol	Time	SR	Sol	Time	SR	Best Sol.	Best SR	Avg. SR	Worst SR	S.D. SR	Avg. Time	S.D. Time
930	268	13.73	268	3.96	1.00	428	10.17	1.60	316	1.18	1.46	1.64	0.12	11.08	0.03
160	220	17.14	234	9.44	1.06	355	9.77	1.61	257	1.17	1.41	1.56	0.08	10.91	0.03
681	320	21.42	337	8.19	1.05	368	8.79	1.15	352	1.10	1.38	1.59	0.14	11.19	0.04
418	268	25.71	268	7.25	1.00	432	9.28	1.61	316	1.18	1.35	1.51	0.17	11.50	0.04
522	273	17.73	276	5.99	1.01	345	8.57	1.26	300	1.10	1.26	1.52	0.13	11.34	0.02
667	355	66.63	371	20.64	1.05	479	8.95	1.35	281	1.10	1.24	1.49	0.17	11.01	0.03
264	319	22.81	322	4.82	1.01	421	8.62	1.32	367	1.15	1.31	1.60	0.11	11.00	0.02
826	228	43.21	249	17.07	1.09	384	8.31	1.68	271	1.19	1.45	1.69	0.10	10.14	1.23
15	292	56.45	311	18.53	1.07	438	8.99	1.50	345	1.18	1.48	1.71	0.11	11.56	0.06
85	286	67.02	307	13.55	1.07	405	9.01	1.42	323	1.13	1.32	1.61	0.13	11.54	0.02
855	267	24.36	267	6.94	1.00	393	9.21	1.47	320	1.20	1.43	1.67	0.19	11.17	0.02
334	272	18.87	272	3.32	1.00	395	9.06	1.45	313	1.15	1.29	1.49	0.11	10.29	1.02
597	310	24.13	313	4.98	1.01	444	8.66	1.43	360	1.16	1.27	1.49	0.10	11.01	0.04
493	255	42.63	272	20.82	1.07	437	8.99	1.71	291	1.14	1.31	1.56	0.13	11.09	0.09
348	288	34.80	288	5.86	1.00	454	9.01	1.58	340	1.18	1.29	1.61	0.17	11.34	0.03
19	278	33.53	300	20.52	1.08	359	8.98	1.29	311	1.12	1.28	1.56	0.08	11.08	0.03
802	317	76.82	318	13.23	1.00	452	9.13	1.43	377	1.19	1.39	1.61	0.12	11.12	0.04
795	301	46.98	314	21.91	1.04	453	9.39	1.50	361	1.20	1.42	1.59	0.13	11.01	0.02
102	314	58.75	324	20.69	1.03	429	8.75	1.37	355	1.13	1.39	1.67	0.09	11.59	0.09
28	247	23.60	249	5.77	1.01	352	8.65	1.43	289	1.17	1.36	1.69	0.19	11.38	0.16
Mean		**36.82**		**11.67**	**1.03**		**9.01**	**1.46**		**1.16**	**1.35**	**1.59**		**11.12**	
S.D.		**18.86**		**6.68**	**0.03**		**0.41**	**0.14**		**0.03**	**0.07**	**0.07**		**0.036**	

(c) N=36, No. of Constraints (in each case) =7 and P_s=450.

Seed	PLS		QE			SCS			HGA						
	Sol	Time	Sol	Time	SR	Sol	Time	SR	Best Sol.	Best SR	Avg. SR	Worst SR	S.D. SR	Avg. Time	S.D. Time
930	348	741.17	364	138.41	1.05	538	16.15	1.55	442	1.27	1.57	1.79	0.10	16.03	0.03
160	306	87.28	315	22.30	1.03	492	15.77	1.61	386	1.26	1.53	1.72	0.12	15.87	0.05
681	329	412.21	356	250.68	1.08	571	15.43	1.74	411	1.25	1.54	1.76	0.10	16.07	0.04
418	307	223.22	319	70.58	1.04	498	14.12	1.62	375	1.22	1.50	1.76	0.11	15.90	0.06
522	306	143.52	306	60.86	1.00	506	14.66	1.65	361	1.18	1.36	1.62	0.09	16.44	0.02
667	394	1353.6	405	273.23	1.03	569	16.01	1.44	437	1.11	1.25	1.68	0.07	16.33	0.05
264	339	355.61	354	106.51	1.04	522	16.06	1.54	383	1.13	1.39	1.56	0.08	15.77	0.08
826	310	180.51	311	64.00	1.00	440	15.92	1.42	388	1.25	1.53	1.73	0.08	16.23	0.03
15	318	39.66	329	13.33	1.03	536	15.32	1.69	391	1.23	1.55	1.71	0.12	15.89	0.06
85	308	126.30	308	48.25	1.00	465	15.89	1.51	354	1.15	1.38	1.52	0.08	16.14	0.02
855	290	121.25	316	56.63	1.09	532	14.12	1.83	354	1.22	1.39	1.59	0.14	16.03	0.02
334	356	334.80	356	80.21	1.00	549	14.97	1.54	409	1.15	1.42	1.61	0.11	15.87	0.05
597	357	295.20	359	78.03	1.01	560	14.65	1.57	453	1.27	1.47	1.61	0.11	16.44	0.04
493	289	75.65	294	20.65	1.02	445	16.11	1.54	347	1.20	1.45	1.78	0.18	15.97	0.07
348	379	254.45	393	81.36	1.04	550	15.97	1.45	447	1.18	1.46	1.73	0.08	16.34	0.09
19	376	2651.4	399	653.84	1.06	452	14.02	1.20	421	1.12	1.25	1.50	0.08	16.39	0.03
802	297	151.72	319	53.98	1.07	471	15.52	1.59	371	1.25	1.41	1.65	0.11	16.06	0.04
795	320	65.81	320	21.76	1.00	458	14.98	1.43	394	1.23	1.46	1.72	0.12	15.97	0.12
102	318	493.60	342	151.65	1.08	469	14.92	1.47	369	1.16	1.32	1.69	0.16	16.12	0.06
28	337	1866.6	339	602.78	1.01	485	15.52	1.44	401	1.19	1.39	1.71	0.09	16.25	0.04
Mean		**498.68**		**142.45**	**1.03**		**15.31**	**1.54**		**1.20**	**1.43**	**1.67**		**16.11**	
S.D.		**667.16**		**175.78**	**0.03**		**0.69**	**0.13**		**0.05**	**0.09**	**0.08**		**0.20**	

(d) N=50, No. of Constraints (in each case) =10 and P_s=400.

Seed	PLS		SCS			HGA						
	Sol.	Time	Sol.	Time	SR	Best Sol.	Best SR	Avg. SR	Worst SR	S.D. SR	Avg. Time	S.D. Time
930	343	4114.60	616	66.09	1.80	459	1.34	1.61	1.91	0.08	25.09	0.03
160	378*	5400.00	526	63.95	1.39	445	1.18	1.51	1.80	0.11	23.06	0.06
681	550*	5400.00	617	63.06	1.12	545	1.00	1.29	1.50	0.13	23.25	0.08
418	410*	5400.00	623	63.56	1.52	557	1.36	1.64	1.83	0.09	24.01	0.11
522	375*	5400.00	600	59.16	1.60	461	1.23	1.54	1.75	0.06	24.32	0.16
667	504*	5400.00	725	71.06	1.44	634	1.26	1.41	1.89	0.15	22.98	0.12
264	309*	5400.00	596	68.06	1.93	402	1.30	1.61	1.93	0.08	24.12	0.09
826	422*	5400.00	695	64.98	1.65	536	1.27	1.63	1.90	0.011	24.03	0.05
15	422	4263.00	653	68.56	1.55	519	1.23	1.50	1.75	0.08	23.56	0.03
85	353	2405.00	558	65.04	1.58	417	1.18	1.46	1.69	0.07	22.76	0.04
855	369*	5400.00	551	63.79	1.49	494	1.34	1.61	1.95	0.14	23.06	0.06
334	452*	5400.00	668	68.21	1.48	589	1.30	1.54	1.79	0.07	23.97	0.26
597	621*	5400.00	658	69.65	1.05	621	1.00	1.18	1.29	0.15	22.07	1.26
493	493*	5400.00	653	71.03	1.32	534	1.08	1.27	1.45	0.16	24.23	0.36
348	442*	5400.00	673	75.32	1.52	526	1.19	1.49	1.76	0.09	23.75	0.52
19	422*	5400.00	669	75.96	1.59	516	1.22	1.51	1.78	0.06	23.65	0.03
802	502*	5400.00	508	76.05	1.01	534	1.06	1.29	1.57	0.12	23.16	0.04
795	402*	5400.00	586	66.50	1.46	554	1.38	1.64	1.97	0.13	23.95	0.11
102	460*	5400.00	627	66.50	1.36	460	1.00	1.19	1.41	0.16	23.07	0.13
28	493	3204.98	533	63.16	1.08	517	1.05	1.26	1.43	0.11	23.06	0.04
Mean		**5019.38**		**67.48**	**1.45**		**1.20**	**1.46**	**1.72**		**23.56**	
S.D.		**831.74**		**4.49**	**0.23**		**0.12**	**0.15**	**0.20**		**0.67**	

* The solution, which is obtained in 5400 seconds, is reported here.

CHAPTER VIII

THE TRAVELLING SALESMAN PROBLEM WITH BACKHAULS

8.1 INTRODUCTION

In this chapter we will discuss the Travelling Salesman Problem with Backhauls (TSPB, for short), in which the nodes of the network (except the starting node) are divided into linehaul and backhaul nodes. The problem is to find a least cost Hamiltonian tour such that all the linehaul nodes are visited contiguously after the starting node, followed by all the backhaul nodes.

Let us formally define the problem.

A network with n nodes, with node 1 as 'headquarters' and a travel cost (or distance, or travel time etc.) matrix $C=[c_{ij}]$ of order n associated with ordered node pairs (i,j) is given. Also is given a set of linehaul nodes (L), and a set of backhaul nodes (B) such that $|L|=l$, $|B|=m$ and $n=l+m+1$. The problem is to obtain the tours

$$\{1=\alpha_0, \alpha_1, \alpha_2, \ldots, \alpha_{n-1}, \alpha_n=1\} \equiv \{1\rightarrow\alpha_1\rightarrow\alpha_2\rightarrow\ldots\rightarrow\alpha_{n-1}\rightarrow 1\}$$

representing irreducible permutations interpreted as simple cycles in which for all $I=1, 2, \ldots, l$, $\alpha_i \in L$ and $\alpha_j \in B$, for all $j=l+1, l+2, \ldots, n-1$; for which the total travel cost

$$C\,(1=\alpha_0, \alpha_1, \alpha_2, \ldots, \alpha_{n-1}, \alpha_n=1) \triangleq \sum_{i=0}^{n-1} c\,(\alpha_i, \alpha_{i+1})$$

is minimum.

Precedence

The TSPB can be viewed as a special case of the $\wedge$ Clustered Travelling Salesman Problem (CTSP) studied by Chisman (1975), Lokin (1978), Jongens and Volgenant (1985), and Kalantari et al. (1984). In the CTSP, the set of nodes is partitioned into k clustered $B_1, B_2, \ldots\ldots\ldots, B_k$, and it is required that the nodes of B_i be visited contiguously.

As shown by Chisman (1975), it can be transformed into the usual TSP by adding / subtracting an arbitrarily large constant M to/ from the edges linking any two of the sets {1}, L, B. Chisman proposed to solve the problem by branch and bound approach of Little et al. (1963). But Lokin (1978) reported that Little's algorithm in combination with Chisman's transformation is very ineffective. Therefore, Lokin modified this algorithm by introducing a new bound technique and got better results. However, precedence relations on the clusters are also imposed by Lokin. But, Jongens and Volgenant (1985) adapted an algorithm for the TSP based on Lagrangean approach. In the Lagrangean approach a new multiplier is introduced to improve the lower bounds. Furthermore, a heuristic to find upper bounds is also described. This algorithm is not based on Chiman's transformation. Also according to Kalantari *et al.* (1985), better exact algorithms can be derived by suitably modifying existing TSP algorithms to account for clusters.

Several heuristics having good empirical behaviour, but no bounded worst-case ratio, are described by Gendreau *et al.* (1996) for the TSPB. These methods are based on the GENIUS algorithm developed for the usual TSP by Gendreau *et al.* (1992) and then compared the solutions with a lower bound based on 1-tree relaxation. Again, Gendreau *et al.* (1997) has described a heuristic having worst-case performance ratio of $\frac{3}{2}$ for the TSPB. But they did not report the results explicitly. As reported by them, on randomly generated problems of sizes 100, 200 and 300, their procedure produce an average deviation of 30% over a shortest spanning tree based lower bound.

The TSPB finds application in a variety of situations, naturally in a number of distribution settings. For example, in a warehouse system customer orders for goods will arrive, which will contain several commodities, each of which will call for different stock numbers. A motorized truck will be dispatched through the warehouse to pick up the commodities. the restriction is that a customer order must be completely satisfied before the next customer order picking is started. The order of picking commodities within each customer order and the sequencing in the customer orders are to be such, that the total order picking time is a minimum. The location of a commodity can be seen as a city and customer order as a linehaul or backhaul city (cf, Chisman 1975, Lokin 1978).

8.2 LEXISEARCH APPROACH (PATH APPROACH) AND ILLUSTRATION

The algorithm is the modification of some steps in the lexisearch algorithm for the usual TSP, described in section 3.2. At first set the cost of some appropriate edges to as large as possible due to the linehaul-backhaul precedence relation. That is, set the cost of (1,j), for $j=l+2, l+3, \ldots\ldots, n$; (i,1), for $i=2, 3, l+1$; and (i,j), for $i=l+2, l+3, \ldots, n$, $j=2, 3, \ldots, l+1$ to a large number. Furthermore, we examine at each stage of concatenation of the present leader with the node, whether it violates any of the linehaul-backhaul precedence relation.

Now, for the lexisearch algorithm of the TSPB, following steps are replaced in the lexisearch algorithm for usual TSP, described in section 3.3.

Step 0:- Remove the 'bias' of matrix C. Construct the alphabet table in such a way that in 2nd, 3rd,......, $(l+1)$th rows all possible linehaul nodes come contiguously before all backhaul nodes, and in $(l+2)$th, $(l+3)$th, ..., nth rows, the node '1' come after all possible backhaul nodes.

Step 1: - Go to the rth element of the row (say, node α) and compute the cost of travelling. If the travel cost is greater than or equal to the 'current trial solution value' or if any linehaul-backhaul precedence relation is violated, go to **step 8**, *else,* go to **step 2**.

Step 7: - If length of the incomplete word is less than l+1 and r < l , go to **step 1,** *else,* go to **step 9**. Also, if length of the incomplete word is greater than l+1 and r <m, go to **step 1**, *else,* go to **step 9**.

Step 8: - Go to sub-block, i.e., go to α^{th} row. If length of incomplete word is l+1, then set r=l, *else,* r=1; go to **step 1** in either case.

8.2.1 ILLUSTRATION

Working of this algorithm is explained through the same problem as given in **Table-3.1** and L={2,3,4}, B={5,6,7}. So, set $c_{15}=c_{16}=c_{17}=c_{21}=c_{31}=c_{41}=c_{52} = c_{53} =c_{54}=c_{62} =c_{63}=c_{64}=c_{72}=c_{73}=c_{74}=999$. **Table-8.1** and **Table-8.2** give the modified cost matrix with bias and reduced cost matrix respectively. The **Table-8.3** give the 'alphabet table', while **Table-8.4** indicates the logic-flow of the algorithm at various stages, which sequentially records the intermediate results, with decision taken (i.e., remarks) at these steps in every column.

TABLE-8.1: - The Modified Cost Matrix with bias.

	1	2	3	4	5	6	7	Row Min
1	999	75	99	9	999	999	999	9
2	999	999	86	46	88	29	20	20
3	999	5	999	16	28	35	28	5
4	999	45	11	999	59	53	49	11
5	86	999	999	999	999	76	72	72
6	36	999	999	999	21	999	52	21
7	58	999	999	999	52	60	999	52
Col. Min	6	0	0	0	0	4	0	

Total Bias = row minima + column minima = 190 + 10 = 200.

TABLE-8.2: - The Reduced Cost Matrix

$i \backslash j$	1	2	3	4	5	6	7
1	984	66	90	0	990	986	990
2	973	979	66	26	68	5	0
3	988	0	994	11	23	26	23
4	982	34	0	988	48	38	38
5	8	927	927	927	927	0	0
6	9	978	978	978	0	974	31
7	0	947	947	947	0	4	947

TABLE-8.3: - The Alphabet Table

	N-V	N-V	N-V	N-V	N-V	N-V	N-V
1	4-0	2-66	3-90	1-984	6-986	5-990	7-990
2	4-26	3-66	7-0	6-5	5-68	1-973	2-979
3	2-0	4-11	5-23	7-23	6-26	1-988	3-994
4	3-0	2-34	6-38	7-38	5-48	1-982	4-988
5	6-0	7-0	1-8	2-927	3-927	4-927	5-927
6	5-0	7-31	1-9	6-974	2-978	3-978	4-974
7	5-0	6-4	1-0	2-947	3-947	4-947	7-947

TABLE-8.4: SEARCH TABLE

$1 \to \alpha_1$	$\alpha_1 \to \alpha_2$	$\alpha_2 \to \alpha_3$	$\alpha_3 \to \alpha_4$	$\alpha_4 \to \alpha_5$	$\alpha_5 \to \alpha_6$	$\alpha_6 \to 1$
$1 \to 4_{(0)}$ (0)+0,GS	$4 \to 3_{(0)}$ (0)+0,GS	$3 \to 2_{(0)}$ (0)+0,GS	$2 \to 7_{(0)}$ (0)+0,GS	$7 \to 5_{(0)}$ (0)+9,GS	$5 \to 6_{(0)}$ (0)+9,GS	$6 \to 1_{(9)}$ **TRVL=9 JO**
				$7 \to 6_{(4)}$ (4)+8,JO		
			$2 \to 6_{(5)}$ (5)+0,GS	$6 \to 5_{(0)}$ (5)+0,GS	$5 \to 7_{(0)}$ (5)+0,GS	$7 \to 1_{(0)}$ **TRVL=5 JO**
				$6 \to 7_{(31)}$ JO		
			$2 \to 5_{(68)}$ JO			
	$4 \to 2_{(34)}$ JO					
$1 \to 2_{(66)}$ STOP						

So, the optimum tour is {1→4→3→2→6→5→7→1}, and optimal solution = bias + trial value = 200 + 5 = 205.

Here, the Lexisearch approach in the context of path representation is considered, the other approach is not easy. To examine the effect of the number of backhaul nodes we have considered different number of backhaul nodes for a same problem. Since the number of nodes in a problem is fixed, so as the number of backhaul nodes increases the number of linehaul nodes decreases. **Table-8.5** shows a comparative study of the path approach considering different number of backhaul nodes. As the number of backhaul nodes increases the only additional number of the nodes (from old set of linehaul nodes) are added to the old existing set of the backhaul nodes.

Table-8.5: - A comparative study of PLS considering various number of Backhaul nodes for the TSPB.

N	Seed	$\lvert B\rvert=\lfloor 3n/6 \rfloor$		$\lvert B\rvert=\lfloor 4n/6 \rfloor$		$\lvert B\rvert=\lfloor 5n/6 \rfloor$		Usual TSP	
		Sol.	Time	Sol.	Time	Sol.	Time	Sol.	Time
	930	208	0.11	293	9.51	286	2.64	**156**	**2.36**
	160	242	0.28	278	1.53	265	1.21	**184**	**2.85**
	681	319	0.22	268	0.22	289	1.42	**166**	**2.64**
30	**418**	326	0.66	328	0.61	221	0.99	**187**	**33.61**
	522	345	1.09	344	1.37	234	2.31	**199**	**3.63**
	Mean		**0.47**		**2.65**		**1.71**		**9.02**
	S.D.		**0.36**		**3.46**		**0.64**		**12.30**
	930	234	2.64	251	1.65	316	38.99	**168**	**30.65**
	160	310	14.12	308	18.01	357	158.90	**186**	**1233.96**
	681	320	31.69	316	4.01	216	11.70	**168**	**4.34**
	418	270	5.93	297	7.58	250	19.61	**156**	**97.65**
36	**522**	298	5.77	296	60.80	301	645.54	**205**	**65.64**
	Mean		**12.03**		**18.41**		**174.95**		**286.45**
	S.D.		**10.54**		**21.92**		**241.24**		**474.81**
	930	298	154.56	331	860.68	263	440.83	**188**	**495.93**
	160	293	441.06	263	21.70	224	99.97	**150**	**39.49**
	681	338	189.76	319	28.73	276	8.12	**158**	**121.16**
	418	259	9.78	306	5.49	267	53.12	**157**	**161.76**
39	**522**	352	43.06	315	98.48	253	162.25	**185**	**20.10**
	Mean		**167.64**		**203.02**		**152.86**		**167.69**
	S.D.		**152.22**		**330.37**		**152.78**		**172.14**

As the number of the backhaul nodes increases one can't say, from **Table-8.5,** that the solution value as well as the time taken for solving the same will also increase or decrease. Of course, if one compares with the same problem with out the linehaul-backhaul precedence constraints (i.e., same usual TSP), then one will notice that the solution values will be more than that of the solution value to the usual TSP, which is **expected** also. It is seen for most of the problems that the time taken for a TSPB is less than that of the same usual TSP. Of course, there are some problems for which the time taken for obtaining the solution values are more than that of the same usual TSP. On the average one should expect to find an exact optimal solution for TSPB faster than for a 'usual' TSP. The number of admissible solutions decreases from (n-1)! for the TSP to ($l!m!$) for the TSPB, where $l=|L|$ and $m=|B|$. The computational results support the expectation for most of the problems. For rest of the problems, while $m >> l$, the factors, e.g. the structure of the cost matrix, may have an impact on the computational effect

8.3 SEQUENTIAL CONSTRUCTIVE SAMPLING APPROACH

This algorithm is basically same as the algorithm for the usual TSP described in section 3.4. The only difference here is that the nodes are partitioned mainly into two subsets and first we have to visit probabilistically the nodes of the first subset and then visit the nodes of other subset. For that, we set the cost of some appropriate edges to a large number due to linehaul-backhaul precedence constraints. Now the 'legitimate' nodes in each row are only the nodes which do not violate any linehaul-backhaul relations and which are not present in the incomplete tour and this number is assumed to be as k of **equation 3.1**. Then we visit probabilistically as described in section 3.4 for the usual TSP.

8.4 HYBRID GENETIC ALGORITHM FOR THE TSPB

For the TSPB, the chromosome is a combination of two ordered substrings- one is the irreducible permutations of nodes 1, 2, ..., l+1 and another is the irreducible permutations of the nodes $l+2$, $l+3$,......, n. So, for a 7-city problem with L={2,3,4} and B={5,6,7}, one of the chromosomes may be (1,2,4,3,6,7,5) representing the salesman's path $\{1\to2\to4\to3\to6\to7\to5\to1\}$. Each 'chromosome' so generated being a salesman's tour has a corresponding cost, which is evaluated, and the best among them is treated as 'current trial solution value'. The crossover, mutation and the sequential constructive operators for the TSP-FPPC are as follows.

(i) Crossover: - C1-Crossover operator (C.f. Reeves 1993) is used for our HGA. The crossover operator is first applied to each of the substrings, then the resultant substrings are combined to have full offspring. For two parents P1 & P2 with L={2,3,4}, B={5,6,7},

```
P1:   1   3 | 2   4   6 | 5   7
P2:   1   2 | 4   3   7 | 6   5
```

at two crossover sites 2 and 5 respectively for the two substrings, the offspring O1 and O2 will be

```
O1:   1   3   2   4   6   7   5
O2:   1   2   3   4   7   6   5
```

(ii) Mutation:- The mutation operator for our problem is as follows. Select randomly a pair of genes (except node 1) randomly within a sub string and interchange the alleles with the probability of mutation (P_m). Repeat the process for the first sub string $l/2$ times and for the other $m/2$ times, where l and m and the numbers of linehaul and backhaul nodes respectively.

(iv) Sequential Constructive operator:- This sequential constructive search approach is same as that of the usual salesman case described in section 4.5. Only the difference is that this approach is applied first for first sub string starting from node '1' and for the second sub string starting from the last node of the first new sub string, and then the resultant substrings are combined in order to get completely new chromosome.

8.5 RELATIVE EFFICIENCY OF DIFFERENT APPROACHES

Relative efficiency analysis was carried out for four sets of randomly generated problems of sizes 30, 36, 39 and 50 having three different number of backhaul nodes each. Each set contains 5 randomly generated problems. For each of the problems generated, the exact optimal solution obtained by PLS, and the best solution obtained by QE (which is not reported for the problem of size 50) and SCS are tabulated. Also are tabulated the times taken for obtaining the same. In case of HGA, the best-found solution; the solution-ratio of it to the optimal solution obtained by PLS along with the average solution-ratio and standard deviation of solution-ratios of 50 runs are reported. These are reported in **Table-8(a, b, c, &d).**

TABLE-8.6: Solution values and time taken (in Seconds) by different algorithms, for five randomly generated problems of sizes 30, 36 and 39.

(a) N=34 and P_s=75.

|B|	Seed	PLS		QE			SCS			HGA					
		Sol	Time	Sol	Time	SR	Sol	Time	SR	Best Sol.	Best SR	Avg. SR	S.D. SR	Avg. Time	S.D. Time
	930	208	0.11	208	0.05	1.00	338	6.54	1.62	208	1.00	1.12	0.05	1.11	0.05
	160	242	0.28	242	0.11	1.00	431	7.03	1.78	242	1.00	1.10	0.08	1.14	0.09
	681	319	0.22	325	0.11	1.02	438	7.31	1.37	319	1.00	1.08	0.01	1.03	0.11
15	**418**	326	0.66	337	0.27	1.03	502	6.97	1.54	334	1.03	1.10	0.09	1.18	0.12
	522	345	1.09	356	0.66	1.03	508	7.69	1.47	345	1.00	1.08	0.04	1.12	0.05
	Mean		**0.47**		**0.23**	**1.02**		**7.11**	**1.56**		**1.01**	**1.10**		**1.12**	
	S.D.		**0.36**		**0.20**	**0.02**		**0.38**	**0.14**		**0.01**	**0.01**		**0.05**	
	930	293	9.51	293	5.16	1.00	419	6.65	1.43	322	1.10	1.17	0.10	1.18	0.11
	160	278	1.53	290	0.61	1.04	412	5.93	1.48	279	1.00	1.14	0.09	1.18	0.10
	681	268	0.22	276	0.11	1.03	346	6.65	1.29	268	1.00	1.08	0.02	1.15	0.06
20	**418**	328	0.61	330	0.11	1.01	424	5.87	1.29	328	1.00	1.08	0.03	1.14	0.08
	522	344	1.37	348	0.11	1.01	467	7.42	1.36	344	1.00	1.06	0.05	1.16	0.06
	Mean		**2.65**		**1.22**	**1.02**		**6.50**	**1.37**		**1.02**	**1.11**		**1.16**	
	S.D.		**3.46**		**1.98**	**0.02**		**0.57**	**0.08**		**0.04**	**0.04**		**0.02**	
	930	286	2.64	286	0.38	0	400	7.96	1.40	286	1.00	1.12	0.04	1.18	0.02
	160	265	1.21	270	0.61	1.9	389	7.25	1.47	278	1.05	1.13	0.10	1.21	0.04
	681	289	1.42	294	0.22	1.7	390	7.53	1.35	289	1.00	1.13	0.07	1.18	0.08
25	**418**	221	0.99	226	0.60	2.3	300	7.25	1.36	230	1.04	1.18	0.03	1.20	0.14
	522	234	2.31	240	1.21	2.6	312	8.18	1.33	239	1.02	1.13	0.06	1.20	0.07
	Mean		**1.71**		**0.60**	**1.7**		**7.63**	**1.38**		**1.02**	**1.14**		**1.19**	
	S.D.		**0.64**		**0.34**	**0.01**		**0.38**	**0.05**		**0.02**	**0.02**		**0.01**	

(b) N=36 and P_s=125.

|B|	Seed	PLS		QE			SCS			HGA					
		Sol	Time	Sol	Time	SR	Sol	Time	SR	Best Sol.	Best SR	Avg. SR	S.D. SR	Avg. Time	S.D. Time
	930	234	2.64	234	0.93	1.00	443	15.93	1.89	234	1.00	1.14	0.08	2.81	0.10
	160	310	14.12	310	7.47	1.01	449	16.70	1.61	310	1.00	1.08	0.07	2.82	0.10
	681	320	31.69	332	16.92	1.04	463	17.63	1.45	320	1.00	1.10	0.06	2.82	0.13
18	418	270	5.93	272	2.19	1.01	471	16.70	1.74	270	1.00	1.09	0.09	2.81	0.16
	522	298	5.77	309	4.62	1.04	436	16.53	1.46	300	1.01	1.06	0.03	2.86	0.25
	Mean		12.03		6.43	1.02		16.70	1.63		1.00	1.09		2.82	
	S.D.		10.54		5.70	0.02		0.55	0.17		0.00	0.03		0.02	
	930	251	1.65	251	0.38	1.00	404	13.40	1.61	251	1.00	1.21	0.10	2.81	0.11
	160	308	18.01	313	7.09	1.02	422	14.12	1.37	317	1.03	1.17	0.09	2.95	0.10
	681	316	4.01	317	1.76	1.00	506	17.52	1.60	318	1.01	1.08	0.02	2.87	0.16
24	418	297	7.58	311	2.47	1.05	506	15.38	1.70	298	1.00	1.15	0.03	2.88	0.10
	522	296	60.80	298	15.54	1.01	469	14.06	1.58	301	1.02	1.11	0.05	2.85	0.09
	Mean		18.41		5.45	1.02		14.90	1.57		1.01	1.14		2.87	
	S.D.		21.92		5.53	0.02		1.46	0.11		0.01	0.05		0.05	
	930	316	38.99	319	5.66	1.01	443	18.84	1.40	324	1.03	1.10	0.04	2.99	0.07
	160	357	158.90	368	69.09	1.03	518	18.12	1.45	367	1.03	1.10	0.10	2.95	0.06
	681	216	11.70	224	4.01	1.04	336	18.12	1.56	224	1.04	1.22	0.07	2.96	0.18
30	418	250	19.61	255	9.23	1.03	368	15.33	1.47	255	1.02	1.16	0.07	3.03	0.14
	522	301	645.54	305	252.82	1.01	422	20.04	1.40	316	1.05	1.15	0.09	3.28	0.09
	Mean		174.95		68.16	1.02		18.09	1.46		1.03	1.15		3.04	
	S.D.		241.24		95.49	0.01		1.55	0.06		0.01	0.04		0.12	

(c) N=39 and P_s=250.

|B|	Seed	PLS		QE			SCS			HGA					
		Sol	Time	Sol	Time	SR	Sol	Time	SR	Best Sol.	Best SR	Avg. SR	S.D. SR	Avg. Time	S.D. Time
	930	298	154.56	305	47.73	1.02	497	24.55	1.67	298	1.00	1.08	0.04	6.96	0.17
	160	293	441.06	295	154.61	1.01	565	25.76	1.93	297	1.01	1.08	0.03	7.06	0.13
	681	338	189.76	348	85.30	1.03	573	24.55	1.70	339	1.00	1.06	0.06	6.81	0.10
19	418	259	9.78	268	5.55	1.04	491	23.01	1.90	259	1.00	1.06	0.07	6.56	0.26
	522	352	43.06	356	16.81	1.01	623	26.04	1.77	357	1.01	1.05	0.03	6.85	0.15
	Mean		167.64		62.00	1.01		24.78	1.79		1.00	1.07		6.85	
	S.D.		152.22		53.95	0.01		1.08	0.10		0.00	0.01		0.17	
	930	331	860.68	333	220.69	1.01	536	22.96	1.62	341	1.03	1.11	0.10	6.83	0.12
	160	263	21.70	271	4.50	1.03	463	22.24	1.76	271	1.03	1.13	0.09	6.93	0.16
	681	319	28.73	320	4.73	1.00	506	21.53	1.59	322	1.01	1.09	0.05	7.49	0.09
26	418	306	5.49	306	1.04	1.00	496	16.69	1.62	327	1.07	1.13	0.10	7.36	0.10
	522	315	98.48	331	29.82	1.05	516	23.68	1.64	326	1.04	1.09	0.09	7.05	0.19
	Mean		203.02		52.17	1.02		21.42	1.65		1.04	1.11		7.13	
	S.D.		330.37		84.90	0.02		2.47	0.06		0.02	0.02		0.25	
	930	263	440.83	270	128.31	1.03	432	21.97	1.64	270	1.03	1.09	0.05	7.31	0.17
	160	224	99.97	230	27.52	1.03	406	23.18	1.81	231	1.03	1.17	0.10	7.60	0.06
	681	276	8.12	276	1.43	1.00	431	25.37	1.56	283	1.03	1.11	0.09	7.30	0.10
32	418	267	53.12	272	17.30	1.02	466	25.32	1.75	293	1.10	1.21	0.12	7.23	0.14
	522	253	162.25	255	20.27	1.01	387	26.48	1.53	266	1.05	1.13	0.09	7.23	0.19
	Mean		152.86		38.97	1.02		24.46	1.66		1.05	1.14		7.33	
	S.D.		152.78		45.48	0.01		1.64	0.11		0.03	0.04		0.14	

(d) N=50 and P_s=400.

|B|	Seed	PLS		SCS			HGA					
		Sol	Time	Sol	Time	SR	Best Sol.	Best SR	Avg. SR	S.D. SR	Avg. Time	S.D. Time
25	**930**	237	168.11	557	78.05	2.35	253	1.07	1.16	0.04	12.72	0.07
	160	250	2692.65	608	79.69	2.43	275	1.10	1.20	0.07	12.14	0.03
	681	375*	3600.00	719	71.03	1.92	377	1.01	1.13	0.06	12.05	0.10
	418	284	3570.98	633	75.05	2.23	308	1.08	1.24	0.10	12.62	0.06
	522	338*	3600.00	651	68.17	1.93	345	1.02	1.12	0.13	12.69	0.05
	Mean		**2726.35**		**74.40**	**2.17**		**1.06**	**1.17**		**12.44**	
	S.D.		**1325.57**		**4.29**	**0.21**		**0.03**	**0.04**		**0.29**	
33	**930**	293	3448.01	619	68.06	2.11	303	1.03	1.21	0.12	12.96	0.02
	160	235	530.70	570	60.78	2.43	255	1.08	1.19	0.23	12.92	0.06
	681	379*	3600.00	615	72.36	1.62	384	1.01	1.08	0.15	13.06	0.09
	418	325	429.19	698	78.97	2.15	337	1.04	1.15	0.17	12.98	0.10
	522	419*	3600.00	722	84.38	1.72	419	1.00	1.09	0.10	13.06	0.09
	Mean		**2321.58**		**72.91**	**2.01**		**1.03**	**1.14**		**13.00**	
	S.D.		**1505.05**		**8.24**	**0.30**		**0.03**	**0.05**		**0.06**	
41	**930**	261	324.51	504	82.17	1.93	277	1.06	1.16	0.12	13.92	0.07
	160	301	3572.97	492	76.98	1.63	322	1.07	1.19	0.17	13.98	0.06
	681	321*	3600.00	552	89.54	1.72	321	1.00	1.15	0.10	13.92	0.10
	418	426*	3600.00	634	91.32	1.49	426	1.00	1.09	0.10	14.04	0.04
	522	319*	3600.00	516	84.26	1.62	319	1.00	1.12	0.13	14.03	0.09
	Mean		**2939.50**		**84.85**	**1.68**		**1.03**	**1.14**		**13.98**	
	S.D.		**1307.54**		**5.16**	**0.15**		**0.03**	**0.03**		**0.05**	

*Note: The solution, which is obtained in 3600 seconds, is reported here

CHAPTER IX

THE MIN-MAX TRAVELLING SALESMAN PROBLEM

9.1 INTRODUCTION

In this chapter we will discuss a new variation of the usual TSP, called Min-Max Travelling Salesman Problem (MMTSP, for short), in which the objective is to minimize the maximum arc length in the tour.

Let us formally define the problem.

A network with n nodes, with node 1 as 'headquarters' and a travel cost (or distance, or travel time etc.) matrix $C=[c_{ij}]$ of order n associated with ordered node pairs (i,j) is given. Let $\{1=\alpha_0, \alpha_1, \alpha_2, \ldots, \alpha_{n-1}, \alpha_n=1\} \equiv \{1 \to \alpha_1 \to \alpha_2 \to \ldots \to \alpha_{n-1} \to 1\}$ be a tour, representing irreducible permutations interpreted as simple cycles. The tour value is now defined as max. $\{c_{\alpha i, \alpha i+1} : i=0,1,2,\ldots,n-1\}$. The objective is to choose a tour which has minimum tour value.

The MMTSP is first proposed by Ramesh (1997); a tour may have a total length relatively small, but may have one or more of the laps rather very large, while other tours, with larger total lengths may have all the laps of the tour small values. Hence the MMTSP has been considered, in which we are interested to minimize the largest arc length in a tour, instead of minimizing the total tour length.

For this problem both adjacency and path approaches are possible. A Lexi search approach (i.e., path approach) was developed to solve the MMTSP by Ramesh (1997). We study his algorithm and compare with our algorithms in the following sections.

9.2 THE ADJACENCY APPROACH AND ILLUSTRAION

The adjacency approach for the MMTSP is same as the adjacency approach for the usual TSP described in section 3.2, with only modification in the objective function and in the bound. For this algorithm (ALS) following steps are replaced in the algorithm for the usual TSP, described in the section 3.2. Here 'bias' removal of the given cost matrix is not justified and hence, should not be carried out.

Step 0: - Form the 'alphabet table' as follow. Sort in ascending order, this matrix row-wise and store the corresponding column indices of each row of the matrix. Initialize the 'current trial solution value' to a large number.

Step 1: - With the partial word of length (l-1), take as leader (for the partial word to be filled) the first unchecked or free letter. Compute the bound by taking the maximum among the leader value, the costs of the remaining (n-l) successive letters after ensuring that the column repetition with each of the (l-1) letters of the partial word is avoided and the present word value.

Step 6: - Current word gives the optimum tour sequence, with 'current trial solution value' as the optimum cost, and **stop**.

9.2.1 ILLUSTRATION

Working of this algorithm is explained through the same example as given earlier in **Table-3.1**. **Table-9.1** and **Table-9.2** give 'the alphabet table' and the 'search table' respectively.

Table-9.1: - Alphabet table

	N-V	N-V	N-V	N-V	N-V	N-V	N-V
1	7-8	4-9	5-35	6-63	2-75	3-99	1-999
2	7-20	6-29	4-46	1-51	3-86	5-88	2-999
3	2-5	4-16	5-28	7-28	6-35	1-100	3-999
4	3-11	1-20	2-45	7-49	6-53	5-59	4-999
5	3-33	2-63	4-65	7-72	6-76	1-86	5-999
6	5-21	4-31	1-36	7-52	2-53	3-89	6-999
7	2-31	3-43	5-52	1-58	6-60	4-67	7-999

Table-9.2: - Search Table

Leaders							Bounds L-1 , N-L values	Trial Value	Remarks
1	2	3	4	5	6	7			
7-8							0, 33	9999	GS
	6-29						8, 33	9999	GS
		2-5					29, 33	9999	GS
			3-11				29, 65	9999	GS
				4-65			29, 58	9999	GS
					5-21		65, 58	9999	CR
					1-36		65, 52	9999	GS
						5-52	65, 0	9999	GS
							TRVL=	65	JB, JO
					7-52		65, 52	65	JO
				7-72			29, 52	65	JO
			1-20				29, 43	65	GS
				3-33			29, 52	65	GS
					5-21		33, 67	65	CR
					4-31		33, 52	65	GS
						5-52	33, 0	65	GS
							TRVL=	52	JB, JO
					1-36		33, 52	52	CR
					7-52		33, 52	52	CR, JO
				2-63			29, 43	52	JO
			5-59				29, 43	52	JO
		4-16					29, 33	52	GS
			3-11				29, 63	52	CR
			1-20				29, 33	52	GS
				3-33			29, 31	52	GS
					5-21		33, 31	52	GS
						2-31	33, 0	52	GS
							TRVL=	33	JB, JO
					2-53		33, 52	33	JO
				2-63			29, 52	33	JO
			2-45				29, 43	33	JO
		5-28					29, 33	33	JB
		7-28					29, 33	33	CR
		6-35					29, 33	33	JO
	4-46						8, 33	33	JO
4-9							0, 33	33	JB
5-35							0, 33	33	STOP

*Note: - The overall bound value is the maximum among the leader value, bound value of (*l*-1) and bound value of (n-*l*) letters.

As seen from the above search table, the optimal solution is given by the permutation $\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 7 & 6 & 4 & 1 & 3 & 5 & 2 \end{pmatrix}$ or equivalently the tour is $\{1 \to 7 \to 2 \to 6 \to 5 \to 3 \to 4 \to 1\}$, and optimal solution = trial value = 33.

9.3 LEXISEARCH APPROACH (PATH APPROACH) AND ILLUSTRATION

The Lexisearch approach for the MMTSP is same as the lexisearch approach for the usual TSP, described in section 3.3, with only modification in the objective function and in the bound (cf, Ramesh 1997).

Now, for the lexisearch algorithm (PLS) of the MMTSP, following steps are replaced in the lexisearch algorithm for usual TSP, described in section 3.3.

Step 0:- Form the 'alphabet table', based on the 'given cost matrix'. Initialize the 'current trial solution value' to a large number. Since our starting node is '1', we start our computation from 1st row. Put r=1.

Step 6:- If the Bound is greater than or equal to the 'current trial solution value', drop the city added in **step 1** and increment r by 1, and then go to **step 7**; *else*, go to **step 8**.

Step 10:- Current word gives the optimal tour sequence, with 'current trial solution value' as the optimum cost.

9.3.1 BOUND CALCULATION

Bound is the maximum value of the costs among the first reachable cities (excluding latest city) from each city within the first (β-1) cities if any; otherwise take the cost of travelling the β^{th} city in that row. The value of $\beta(\leq n)$ we have considered in this present study as $\left\lfloor \frac{n}{5} \right\rfloor$.

9.3.2 ILLUSTRATION

As illustration, we consider the same problem described in section 9.2. **Table-9.1** gives the 'alphabet table'. The logic-flow of the algorithm at various stages is indicated in **Table-9.3,** which sequentially records the intermediate results, with decision taken (i.e., remarks) at these steps in every column.

TABLE-9.3: SEARCH TABLE

$1\to\alpha_1$	$\alpha_1\to\alpha_2$	$\alpha_2\to\alpha_3$	$\alpha_3\to\alpha_4$	$\alpha_4\to\alpha_5$	$\alpha_5\to\alpha_6$	$\alpha_6\to 1$
$1\to 7_{(8)}$ 33,GS	$7\to 2_{(31)}$ 33,GS	$2\to 6_{(29)}$ 33,GS	$6\to 5_{(21)}$ 33,GS	$5\to 3_{(33)}$ 20,GS	$3\to 4_{(16)}$ 20,GS	$4\to 1_{(20)}$ **TRVL=33 JO**
				$5\to 4_{(65)}$ JO		
			$6\to 4_{(31)}$ 33,JB			
			$6\to 3_{(89)}$ JO			
		$2\to 4_{(46)}$ JO				
	$7\to 3_{(43)}$ JO					
$1\to 4_{(9)}$ 33,JB						
$1\to 5_{(35)}$ STOP						

So, the optimum tour is $\{1\to 7\to 2\to 6\to 5\to 3\to 4\to 1\}$, and optimal solution = trial value = 33.

To see which of the algorithm between PLS and ALS is good for the MMTSP, we make a comparison between them. So, a comparative study is carried out for three sets problems of sizes 20, 25 and 30, and is shown in **Table-9.4**.

Table-9.4: Comparative study of PLS and ALS for the Min-Max TSP.

Seed	N=20			N=25			N=30		
	Sol.	Time		Sol.	Time		Sol.	Time	
		PLS	ALS		PLS	ALS		PLS	ALS
930	13	0.00	0.00	12	0.05	0.00	12	1.43	1.04
160	26	6.10	2.58	20	3.57	19.77	14	1.87	0.11
681	19	0.05	0.00	23	202.90	66.30	14	9.22	3.79
418	23	0.22	0.00	22	322.90	75.58	19	600	600
522	19	0.00	0.06	14	0.17	0.05	15	89.80	68.60
Mean		**1.27**	**0.53**		**105.92**	**32.34**		**140.46**	**134.71**
S.D.		**2.41**	**1.03**		**133.68**	**32.46**		**232.17**	**234.09**

It is seen from **Table-9.4** that the ALS algorithm is better than the PLS algorithm. So, the ALS approach is used for obtaining exact optimal solution and then the quasi-exact optimal solution for the MMTSP.

9.4. SEQUENTIAL CONSTRUCTIVE SAMPLING APPROACH

Our Sequential Constructive Sampling (SCS) approach is same as the Sequential Constructive Sampling Approach developed in the context of usual TSP, described in section 3.4. The only difference is in the objective function value calculation and the bound calculation method. We consider the bound as calculated in the Lexi search approach, described in section 9.3.1. Of course, we will discuss the modified 2-Opt move for the MMTSP. The modified 2-Opt move is as follows.

As it is already discussed in the 2-Opt Move, for the usual TSP, that given a tour

$\{1\to\alpha_1\to\alpha_2\to\ldots\to\alpha_i\to\alpha_{i+1}\to\alpha_{i+2}\to\ldots\to\alpha_{j-1}\to\alpha_j\to\alpha_{j+1}\to\ldots\to\alpha_{n-1}\to 1\}$,

a neighboring solution can be easily generated at random by generating randomly 'i' and 'j'. So, the new tour will be

$\{1\to\alpha_1\to\alpha_2\to\ldots\to\alpha_i\to\alpha_j\to\alpha_{i+2}\to\ldots\to\alpha_{j-1}\to\alpha_{i+1}\to\alpha_{j+1}\to\ldots\to\alpha_{n-1}\to 1\}$.

In order to check the preferability of the new tour, just to see whether the costs

$\{c(\alpha_i, \alpha_j) < c(\alpha_i, \alpha_{i+1})\}$, $\{c(\alpha_j, \alpha_{i+2}) < c(\alpha_{i+1}, \alpha_{i+2})\}$, $\{c(\alpha_{j-1}, \alpha_{i+1}) < c(\alpha_{j-1}, \alpha_j)\}$ and $\{c(\alpha_{i+1}, \alpha_{j+1}) < c(\alpha_j, \alpha_{j+1})\}$.

If the above satisfied, then only we accept the new tour.

9.5 HYBRID GENETIC ALGORITHM FOR THE MMTSP

Our hybrid genetic algorithm (HGA) is same as the HGA developed in the context of usual TSP, described in section 4.6. The only difference is in the objective function value calculation and hence the fitness function.

9.6 RELATIVE EFFICIENCY OF DIFFERENT APPROACHES

Relative efficiency analysis was carried out for four sets of randomly generated problems of sizes 20, 25, 30 and 50. Each set contains 20 randomly generated problems. For each of the problems generated, the exact optimal solution obtained by ALS, and the best solution obtained by QE (it is not reported for the problem of size 50) and SCS are tabulated. Also are tabulated the times taken for obtaining the same. In case of HGA, the best-found solution; the solution-ratio of it to the optimal solution obtained by ALS along with the average solution-ratio, worst solution-ratio and standard deviation of solution-ratios of 50 runs are reported). These are reported in **Table-9.5(a, b, c & d).**

TABLE - 9.5: - Solution values and time taken (in Seconds) by different algorithms, for twenty randomly generated problems of sizes 20, 25, 30 and 50.

(a) N=20 and P_s=150.

Seed	ALS		QE			SCS			HGA						
	Sol	Time	Sol	Time	SR	Sol	Time	SR	Best Sol.	Best SR	Avg. SR	Worst SR	S.D. SR	Avg. Time	S.D. Time
930	13	0.00	13	0.00	1.00	16	0.61	1.23	13	1.00	1.57	2.46	0.35	0.96	0.10
160	26	2.58	28	0.00	1.08	26	1.86	1.00	26	1.00	1.09	1.35	0.11	0.68	0.32
681	19	0.00	19	0.00	1.13	19	1.16	1.00	19	1.00	1.28	1.74	0.18	0.85	0.23
418	23	0.00	26	0.00	1.13	23	1.15	1.00	23	1.00	1.13	1.57	0.10	0.95	0.14
522	19	0.06	20	0.00	1.05	19	0.71	1.00	19	1.00	1.22	1.58	0.17	0.88	0.19
667	25	0.08	26	0.00	1.04	25	1.12	1.00	25	1.00	1.22	1.28	0.08	0.79	0.07
264	18	1.47	18	0.11	1.00	19	1.03	1.06	19	1.06	1.40	1.78	0.16	0.84	0.03
826	16	1.46	16	0.06	1.00	16	1.25	1.00	17	1.06	1.44	1.64	0.21	0.89	0.02
15	21	1.51	22	0.11	1.05	21	1.09	1.00	21	1.00	1.14	1.43	0.14	0.91	0.28
85	22	0.05	23	0.00	1.05	22	0.92	1.00	22	1.00	1.08	1.45	0.12	0.85	0.32
855	18	0.05	20	0.00	1.11	18	1.12	1.00	18	1.00	1.37	1.89	0.26	0.87	0.09
334	24	35.28	24	5.98	1.00	28	1.02	1.17	24	1.00	1.12	1.42	0.10	0.94	0.28
597	29	13.23	29	0.13	1.00	29	0.86	1.00	29	1.00	1.12	1.41	0.12	0.95	0.31
493	17	0.06	19	0.00	1.12	17	0.95	1.00	17	1.00	1.25	1.88	0.20	0.86	0.04
348	24	386.72	26	292.64	1.08	24	0.77	1.00	24	1.00	1.08	1.42	0.11	0.81	0.32
19	20	0.11	21	0.05	1.05	20	1.27	1.00	20	1.00	1.14	1.55	0.15	0.86	0.51
802	16	0.05	18	0.00	1.12	17	0.75	1.06	17	1.06	1.41	1.94	0.21	1.13	0.56
795	15	0.00	15	0.00	1.00	17	1.09	1.13	17	1.13	1.72	2.47	0.40	1.36	0.54
102	18	0.00	18	0.00	1.00	18	0.42	1.00	24	1.33	1.59	1.83	0.13	1.10	0.52
28	22	0.17	22	0.06	1.00	22	0.32	1.00	22	1.00	1.19	1.36	0.12	0.76	0.46
Mean		**22.14**		**14.96**	**1.05**		**0.97**	**1.03**		**1.03**	**1.28**	**1.67**		**0.91**	
S.D.		**84.02**		**63.72**	**0.05**		**0.33**	**0.07**		**0.08**	**0.18**	**0.33**		**0.14**	

(b) N=25 and P_s=225.

Seed	ALS		QE			SCS			HGA						
	Sol	Time	Sol	Time	SR	Sol	Time	SR	Best Sol.	Best SR	Avg. SR	orst SR	S.D. SR	Avg. Time	S.D. Time
930	12	0.00	13	0.00	1.08	12	1.32	1.00	13	1.08	1.55	2.08	0.27	2.34	0.02
160	20	19.77	22	8.92	1.10	20	4.61	1.00	20	1.00	1.31	1.80	0.16	2.33	0.18
681	23	66.30	23	8.25	1.00	23	4.94	1.00	23	1.00	1.11	1.30	0.11	1.61	0.73
418	22	75.58	23	15.26	1.05	22	4.95	1.00	22	1.00	1.19	1.50	0.11	2.33	0.14
522	14	0.05	15	0.06	1.07	15	4.11	1.07	16	1.14	1.52	1.79	0.13	2.32	0.03
667	21	262.45	23	18.42	1.10	21	3.12	1.00	23	1.10	1.33	1.67	0.11	2.33	0.03
264	19	0.11	21	0.00	1.11	19	1.02	1.00	19	1.00	1.16	1.37	0.13	2.21	0.53
826	13	2.35	15	0.65	1.15	13	4.11	1.00	17	1.31	1.50	2.08	0.17	2.25	0.03
15	12	1.43	12	0.88	1.00	13	3.65	1.08	16	1.33	1.52	2.58	0.24	2.29	0.02
85	18	85.83	20	29.17	1.11	18	3.97	1.00	18	1.00	1.23	1.44	0.09	2.25	0.29
855	21	26.35	21	15.41	1.00	21	0.99	1.00	21	1.00	1.09	1.38	0.09	2.31	0.71
334	11	0.08	11	0.00	1.00	11	3.11	1.00	11	1.00	1.65	2.27	0.23	2.32	0.09
597	17	2.70	17	1.70	1.00	17	3.98	1.00	17	1.00	1.45	1.76	0.24	2.29	0.35
493	21	0.35	21	0.11	1.00	21	1.12	1.00	21	1.00	1.20	1.38	0.21	2.34	0.21
348	16	14.35	17	3.09	1.06	19	4.32	1.19	18	1.13	1.56	1.81	0.17	2.28	0.03
19	17	0.00	17	0.00	1.00	17	0.97	1.00	17	1.00	1.36	1.65	0.19	2.32	0.03
802	15	0.16	17	0.07	1.13	15	3.06	1.00	15	1.00	1.42	1.67	0.14	2.36	0.11
795	17	0.62	19	0.55	1.12	17	3.86	1.00	18	1.06	1.36	1.76	0.17	2.28	0.03
102	15	0.07	16	0.00	1.07	18	3.52	1.20	21	1.40	1.47	1.87	0.13	2.32	0.03
28	22*	600.00	22	1.23	1.00	22	4.98	1.00	22	1.00	1.10	1.32	0.08	2.36	0.02
Mean		**57.93**		**5.19**	**1.06**		**3.29**	**1.03**		**1.08**	**1.35**	**1.72**		**2.27**	
S.D.		**137.94**		**8.02**	**0.05**		**1.39**	**0.06**		**0.12**	**0.17**	**0.33**		**0.16**	

(c) N=30 and P_s=325.

Seed	ALS		QE			SCS			HGA						
	Sol	Time	Sol	Time	SR	Sol	Time	SR	Best Sol.	Best SR	Avg. SR	Worst SR	S.D. SR	Avg. Time	S.D. Time
930	12	1.04	14	0.55	1.08	12	7.20	1.00	14	1.17	1.78	2.17	0.19	5.02	0.03
160	14	0.11	15	0.06	1.07	16	11.04	1.14	15	1.07	1.65	2.21	0.20	5.06	0.04
681	14	3.79	14	1.37	1.00	14	7.74	1.00	14	1.00	1.37	1.64	0.17	5.00	0.51
418	19*	600.00	19	444.78	1.00	19	11.37	1.00	19	1.00	1.18	1.47	0.12	5.09	0.13
522	15	68.60	15	2.80	1.00	16	8.96	1.07	16	1.07	1.22	1.53	0.11	5.01	0.04
667	18	0.06	20	0.06	1.11	18	3.35	1.00	18	1.00	1.33	1.72	0.16	5.12	0.54
264	17	3.11	18	1.04	1.06	17	1.98	1.00	17	1.00	1.24	1.47	0.11	5.11	0.73
826	13	50.74	13	26.09	1.00	15	12.67	1.15	14	1.08	1.78	2.31	0.38	5.12	0.04
15	11	594.72	12	47.07	1.09	11	4.13	1.00	13	1.18	1.53	2.09	0.23	5.02	0.05
85	18*	600.00	20	243.87	1.10	18	11.24	1.00	18	1.00	1.08	1.62	0.16	5.09	1.42
855	13	0.52	13	0.16	1.00	13	8.01	1.00	15	1.15	1.60	2.08	0.25	5.08	0.04
334	11	0.19	11	0.11	1.00	11	10.32	1.00	14	1.27	1.64	2.27	0.21	5.07	0.05
597	20*	600.00	20	324.98	1.00	20	1.56	1.00	20	1.00	1.19	1.45	0.17	5.11	1.13
493	12	3.23	12	1.85	1.00	16	14.12	1.33	18	1.50	1.81	2.33	0.22	5.09	0.04
348	15	66.08	15	31.25	1.00	20	9.85	1.33	20	1.33	1.55	1.73	0.09	5.08	0.4
19	15*	600.00	15	600.00	1.00	15	7.88	1.00	16	1.07	1.38	1.87	0.16	4.96	0.03
802	15*	600.00	15	600.00	1.00	15	9.57	1.00	16	1.07	1.40	1.80	0.21	4.98	0.03
795	13	4.66	14	1.58	1.08	13	9.06	1.00	14	1.08	1.81	2.38	0.25	4.92	0.04
102	31*	600.00	32	600.00	1.02	31	14.98	1.00	31	1.00	1.00	1.00	0.00	0.76	0.27
28	15	1.98	16	0.06	1.07	20	8.97	1.33	17	1.13	1.53	1.93	0.16	4.99	0.04
Mean		**219.94**		**146.38**	**1.03**		**8.70**	**1.07**		**1.11**	**1.45**	**1.85**		**4.83**	
S.D.		**279.10**		**225.64**	**0.04**		**3.60**	**0.12**		**0.13**	**0.25**	**0.36**		**0.94**	

*Note: The solution, which is obtained in 600 seconds, is reported here

(d) N=50 and P_s=400.

Seed	ALS		SCS			HGA						
	Sol.	Time	Sol.	Time	SR	Best Sol.	Best SR	Avg. SR	Worst SR	S.D. SR	Avg. Time	S.D. Time
930	10	5.44	13	141.06	1.30	13	1.30	1.64	2.10	0.16	21.03	4.09
160	12	0.21	12	145.46	1.00	12	1.00	1.55	2.33	0.30	20.97	1.80
681	13*	3600.00	13	158.54	1.00	13	1.00	1.31	2.10	0.25	16.50	5.86
418	11*	3600.00	13	170.06	1.18	14	1.27	1.69	2.61	0.20	19.58	1.11
522	13*	3600.00	13	56.08	1.00	15	1.15	1.41	2.27	0.18	19.97	0.08
667	11*	3600.00	12	123.81	1.09	11	1.00	1.51	2.00	0.15	19.01	1.70
264	13*	3600.00	13	149.53	1.00	13	1.00	1.34	2.18	0.12	18.59	3.05
826	10*	3600.00	13	149.98	1.30	13	1.30	1.94	1.61	0.25	19.87	0.06
15	9	50.40	11	128.26	1.22	11	1.22	1.83	2.56	0.26	19.89	0.08
85	8	2811.60	11	142.65	1.38	12	1.50	2.12	2.88	0.29	21.56	1.83
855	17*	3600.00	17	165.13	1.00	17	1.00	1.13	1.36	0.09	18.86	3.48
334	12*	3600.00	12	124.26	1.00	14	1.17	1.23	1.71	0.16	18.86	2.10
597	13*	3600.00	13	140.35	1.00	14	1.08	1.26	1.57	0.10	20.31	4.50
493	10	3.10	15	157.98	1.50	14	1.40	1.93	2.30	0.21	20.03	0.16
348	12*	3600.00	13	153.92	1.08	16	1.33	1.63	2.00	0.12	21.12	1.24
19	12*	3600.00	12	179.12	1.00	14	1.17	1.58	1.75	0.13	19.27	1.80
802	12*	3600.00	13	208.05	1.08	12	1.00	1.31	1.92	0.15	19.11	5.86
795	10*	3600.00	10	187.08	1.00	11	1.10	1.65	2.20	0.25	18.89	1.11
102	13*	3600.00	13	232.34	1.00	13	1.00	1.63	1.85	0.21	19.77	0.08
28	7	1241.72	11	215.95	1.57	11	1.57	1.35	3.14	0.19	19.79	1.70
Mean		**2725.62**		**156.48**	**1.13**		**1.18**	**1.55**	**2.12**		**19.65**	
S.D.		**1454.46**		**37.12**	**0.18**		**0.17**	**0.26**	**0.43**		**1.10**	

*Note: The solution, which is obtained in 3600 seconds, is reported here.

CHAPTER X

DISCUSSIONS AND COMMENTS

We have discussed exact and heuristic algorithms for the usual TSP and some of its variations. Lexisearch approach have been developed to obtain exact optimal solutions to the problems and based on the lexisearch approach, quasi-exact method have been developed. Then a sequential constructive sampling and hybrid genetic algorithm have been developed to obtain heuristically optimal solutions to the problems. Finally the relative efficiency have been carried out for randomly generated test problems of different sizes.

In the context of lexisearch, two methods have been discussed – adjacency approach and path approach. For the usual TSP, in context of lexisearch, there are two ways of sub-tour checking – (i) during formation of permutation and (ii) at the end of full permutation. One may say that the second method is better than the first one. But it is seen from the **Table-3.5** that the first one is better than the second one. It is due to the fact that there are so many full permutations generated which are not feasible tours, so one has to throw them away, which involves too much time. It is also seen that adjacency approach shows large variations in the context of time taken for solving the problems of same sizes. Hence, the data-guided lexisearch approach has been developed to minimize the variations in times. But the data-guided lexisearch approach still shows large variations in times and the problems seem to fall into two distinct groups, so far as time requirement is concerned. One group requires significantly less time than the average while another takes significantly more time than the average, with a big 'gap' between the two groups. Also, it is seen from the **Table-3.10** that for some of the problems ALS is better than DGLS in the context of time requirement. The DGLS is the modification of ALS, and it comprises of two stages– preliminary scrutiny and preprocessing. Though the data-guided algorithm is presented by Srinivas (1989), but the actual building of full-fledged data-guided is not presented and in some cases, preliminary scrutiny and pre-processing take too much time.

In path approach, explicit testing for cycle formation is avoided, but bound setting is not efficient. Also in the context of lexisearch approach, the 'bias' removal plays an important role in time requirement for solving the problems. In case of PLS also, a large variation in times is seen. In the adjacency approach, the data-guided module is not easily obtained and we could not find it. So, one might try to get data-guided module for the path approach. A comparative study has been carried out for the two approaches along with the data-guided lexisearch. It is seen from the **Table-3.10** that DGLS is the best exact method.

In the context of sequential constructive sampling approach for the TSP, the restricting choice of chance selection to only the first few of the 'legitimate nodes' in the alphabet table, is better than any systematic biasing in node selection– either by rank order or by the actual cost factors involved. In case of hybrid genetic algorithm, our proposed sequential constructive operator seems to be better than some existing good crossover operators, e.g., ERX, C1X and GNX. To see the robustness of our HGA, several runs were performed and the solution values for some of the benchmark problems given in TSPLIB are reported in **Table-4.2**. From that table it is seen that our HGA is efficient for the usual TSP. Now let us summarize the **Table-4.3** as shown in **Table-10.1**in the context of time taken.

Table-10.1- Time taken for solving usual TSP by different approaches.

N		DGLS	QE	SCS	HGA	N	DGLS	QE	SCS	HGA
34	**Mean**	8.13	4.09	14.16	3.64	39	547.77	228.9	26.43	7.91
	S.D.	11.52	5.44	0.39	0.02		1262.1	539.9	0.55	0.09
	Median	1.51	1.04	14.11	3.64		53.31	13.69	26.36	7.89
	Max Gap	30.31	9.19	0.83	0.02		2829.70	1411.30	0.64	0.15
	Max-Min	49.70	21.84	2.12	0.09		5399.68	2345.87	2.20	0.46
36	**Mean**	40.02	14.85	17.64	6.30					
	S.D.	67.52	25.48	1.49	0.20					
	Median	4.70	1.98	18.02	6.25		-----	----	-----	-----
	Max Gap	140.36	48.81	1.12	0.61					
	Max-Min	205.92	82.66	5.64	0.99					

Here, for calculating Max Gap, the time taken for solving the problems of same size are arranged in ascending order and the difference between two consecutive timings are calculated. Then the maximum of these differences is retained as the Max Gap and Max-Min is the difference between maximum and minimum timings. It is seen from the **Table-10.1** that as the size of the problem increases the mean, standard deviation, median, Max Gap and max-min increase rapidly. Also it is seen that at least half of the problems of same size take very less time and rest halves take very large time. To reduce the computational times and variation in times, the quasi-exact method based on lexisearch approach have been developed. Though the time taken by the quasi-exact method is reduced by about 50% of by lexisearch, still it shows a large variation in times. Hence a usual question may be arisen– what type of problem takes less time and large time? Or, can we say by analysing a problem before going to solve the problem, whether it takes less or large time? To answer the question, we carried out the analysis of variance of the problem, but we could not come to any conclusion. So, one might try to analyse this fact. In case of sequential constructive sampling and hybrid genetic algorithm, the time taken are stable.

However, when it comes to the most important criterion of performance, viz. **nearness** to the optimal solution value, the quasi-exact method is better than sequential constructive sampling approach as well as hybrid genetic algorithm. For HGA, let us have a look on the solution quality as shown in **Table-10.2**.

Table-10.2- Performance of our HGA for the usual TSP.

N	Problem	Best SR	Avg. SR	Worst SR	Avg. (Time)
34	Ftv33	1.00	1.05	1.08	3.77
	Random*	1.02	1.14	1.32	3.64
36	Ftv35	1.00	1.01	1.04	4.42
	Random*	1.02	1.13	1.30	6.30
39	Ftv38	1.00	1.02	1.07	5.21
	Random*	1.03	1.13	1.30	7.91

*Note: - mean of the 20 randomly generated problems, as reported in table-4.3, is reported here.

It is seen from the **Table-10.2** that our HGA is good for the randomly generated test problems and better for the benchmark, structured problems. Hence, if we consider the solution quality as well as the time taken, then it can be concluded that HGA is the best for the usual TSP.

We have seen in case of usual TSP that the adjacency approach is better than the path approach in the context of the lexisearch. But, for the TSP-PC as well as TSP-FPC, TSP-FPPC and TSPB, the restriction checking is much more easier in the path approach than in the adjacency approach while forming the tour. In fact, in the adjacency approach, we can check the feasibility of a solution only after a complete permutation is obtained. It is seen from the **Table-5.5** that PLS is better than the ALS for TSP-PC, so far the time taken is concerned. Hence, one can't say that a method which is good for the usual TSP will also good for the TSP with restrictions, in the context of time requirement. The PLS is considered for obtaining exact optimal solutions to TSP-PC, TSP-FPC, TSP-FPPC and TSPB, and the quasi-exact methods are developed on the basis of path approaches. Let us summarize the time taken by various methods for the TSP-PC, TSP-FPC and TSP-FPPC as shown in **Table-10.3**.

It is seen from the **Table-10.3** that as the size of the problem increases, the mean, standard deviation, median, Max Gap and Max-Min of times also increase. Here also, at least half of the problems take very less time and the other half takes very large time, while the time taken by sequential constructive sampling and hybrid genetic algorithm are stable. For these problems also, though the time taken by the Quasi Exact method is reduced by about 50% of by PLS, still it shows a large variation in times and the problems seem to fall into two distinct groups, in so far as time required is concerned. This finding is similar to the situations found in quite a few other combinatorial programming problems as well. (cf, Pandit and Ravikumar 1993, Ravikumar 1994) suggesting that a preprocessing of particular problem data is very desirable, to evolve a data-guided sequencing of modules in optimization algorithm so as to obtain solution much quicker than otherwise, by using the same algorithm with arbitrary, rigid, sequencing of modules. Hence, in case of the TSP with different constraints considered here, it is perhaps desirable

Table-10.3- Time taken for solving TSP-PC, TSP-FPC and TSP-FPPC by different approaches.

	N		PLS	QE	SCS	HGA	N	PLS	QE	SCS	HGA
TSP-PC	30	Mean	16.38	6.20	8.29	5.47	36	163.25	60.31	20.51	11.32
		S.D.	22.26	8.46	0.78	0.21		254.41	86.83	1.36	0.33
		Median	5.92	3.37	8.12	5.57		47.65	25.29	20.12	11.17
		Max Gap	34.65	9.74	0.84	0.33		397.00	110.39	3.20	0.15
		Max-Min	83.78	31.84	3.34	0.64		1057.14	329.32	6.29	1.04
	34	Mean	26.83	10.52	15.09	8.45	50	2578.71	-----	96.44	21.95
		S.D.	41.01	16.97	0.42	0.15		993.29		2.18	0.61
		Median	11.38	4.45	15.01	8.45		2737.20		96.96	21.95
		Max Gap	139.60	52.29	0.50	0.12		589.67		1.03	0.53
		Max-Min	190.95	76.91	1.74	0.60		2472.00		7.27	2.22
TSP-FPC	30	Mean	10.73	4.81	5.83	3.22	36	279.88	92.84	13.79	10.20
		S.D.	9.64	4.62	0.50	0.06		361.61	121.88	0.62	0.10
		Median	6.86	2.91	5.72	3.23		152.33	55.87	14.01	10.18
		Max Gap	11.28	8.84	0.49	0.06		1084.72	390.76	0.54	0.11
		Max-Min	37.50	19.20	1.91	0.21		1704.81	573.98	2.00	0.40
	34	Mean	20.68	10.92	9.12	7.78	50	3272.63	------	66.02	20.22
		S.D.	11.23	7.22	0.69	0.42		632.80		1.31	0.56
		Median	17.53	9.52	9.01	7.68		3600.00		66.09	20.01
		Max Gap	6.48	6.42	0.63	1.87		800.80		1.00	0.57
		Max-Min	39.17	26.85	2.77	1.98		2274.20		4.89	2.18
TSP-FPPC	30	Mean	26.77	11.11	5.63	4.25	36	498.68	142.45	15.31	16.11
		S.D.	29.56	14.60	0.35	0.12		667.16	175.78	0.69	0.20
		Median	20.83	5.56	5.62	4.23		223.22	70.58	15.43	16.06
		Max Gap	94.47	33.22	0.26	0.08		784.80	329.55	0.53	0.10
		Max-Min	137.67	65.12	1.31	0.40		2611.74	640.51	2.13	0.67
	34	Mean	36.82	11.67	9.01	11.12	50	5019.38	------	67.48	23.56
		S.D.	18.86	6.68	0.41	0.036		831.74		4.49	0.67
		Median	25.71	8.19	8.99	11.09		5400.00		66.50	23.56
		Max Gap	9.80	3.79	0.40	0.62		1137.00		4.26	0.77
		Max-Min	63.09	18.59	1.86	1.45		2995.00		16.89	3.02

to investigate the possibility of using a data-guided module-sequencing approach to optimization, but we could not do so. It is also observed that during the process of obtaining the optimum solution by the Lexisearch approach, the confirmation of a solution to be 'optimal' takes a lion's share in the total computational time. But the heuristics give most of the times "near optimal" (if not optimal) solutions very quickly. There are some problems of size 50, which can't be solved optimally by

PLS within one hour, but HGA gives a good result in a reasonable amount of time. For all of the problems, if one considers the solution quality as well as the time taken for solving them, then one can conclude that HGA is the best.

In the context of lexisearch approach for the TSPB, as the number of the backhaul nodes increases one can't say from **Table-8.5** whether the solution value will also increase or decrease. On the average, one should expect to find an exact optimal solution for TSPB faster than that of a 'usual' TSP. The number of admissible solutions decreases from (n-1)! for the TSP to ($l!m!$) for the TSPB, where $l=|L|$ and $m=|B|$. The computational results support the expectation for most of the problems. For the rest of the problems, the factors, e.g. the structure of the cost matrix, may have an impact on the computational effect while $m >> l$. For this problem also in the context of time requirement, lexisearch shows a large variation while the sequential constructive sampling and hybrid genetic algorithm are stable, and HGA is the best among the heuristic algorithms.

The bound calculation method plays a vital role in the lexisearch approach. But no good bound is found for the TSP with different constraints. So, we have used the method whatever used for the usual TSP without taking the constraints into account. Hence, a good (efficient) bound may reduce the time taken for obtaining the exact optimal solution.

For MMTSP, both the adjacency and path approaches are possible. Ramesh (1997) proposed the path approach for solving the same. So, we discussed both the approaches. As shown in **Table-9.4,** it is seen that adjacency approach is better than the path approach in the context of time taken. For this problem also, adjacency approach as well as quasi-exact method show a large variation in times, while the sequential constructive sampling and hybrid genetic algorithm are stable (see also **Table-10.4**). From **Table-9.5,** if one compares the solution quality by SCS with the best as well as average solution quality by HGA, then one can conclude that the sequential constructive sampling approach is the best among the heuristic approaches. Also there are some problems of size 50, which can't be solved optimally by ALS within one hour, but SCS gives a good result with a reasonable amount of time.

Table-10.4- Time taken for solving MMTSP by different approaches.

N		ALS	QE	SCS	HGA	N	ALS	QE	SCS	HGA
20	**Mean**	22.14	14.96	0.97	0.91	30	219.94	146.38	8.70	4.83
	S.D.	84.02	63.72	0.33	0.14		279.10	225.64	3.60	0.94
	Median	0.06	0.00	1.02	0.87		4.66	1.85	8.97	5.06
	Max Gap	351.44	286.66	0.59	0.23		526.12	196.80	3.07	4.16
	Max-Min	386.72	292.64	1.54	0.68		599.94	599.94	13.42	4.36
25	**Mean**	57.93	5.19	3.29	2.27	50	2725.62	-----	156.48	19.65
	S.D.	137.94	8.02	1.39	0.16		1454.46		37.12	1.10
	Median	1.43	0.65	3.65	2.32		3600.00		149.98	19.77
	Max Gap	337.55	10.75	1.74	0.60		1569.88		67.73	2.09
	Max-Min	600.00	29.17	4.01	0.75		3599.79		176.26	5.06

In this present study, it is very difficult to say how big n before the lexisearch becomes impracticable. It certainly depends upon the structure of the problem. That is why some of the problems, say, of size 50, give solution very quickly and some other take large amount of time. One important point, ignored in the present study, needs special mention. In real-life situation, one expects some sort of structure to be present in the cost matrix C. One does not expect that every node is statistically equivalent to every other node in terms of cost of travel between them; cost of travelling only a few nodes will be near enough to a particular node, many more nodes being quite far from it. This situation could act very favourable with respect to lexisearch and could possibly profitably utilized in sequential constructive sampling approach by making the probability of selection of nodes for augmentation to be not equal (as was done in the present study) but dependent on this prior information about the structure of the C matrix. We investigated in this direction, but we could not come to any conclusion in this regard.

In case of HGA, though we have tried to fine-tune the parameters for better performance, actually the setting of parameters is not a simple task. Carefully choosing the parameters may lead to a better performance of the HGA for the problems.

APPENDIX

```
// PROGRAM 1.1
// USUAL TRAVELLING SALESMAN PROBLEM USING LEXI-SEARCH
// ADJACENCY APPROACH

#include<stdio.h>
#include<stdlib.h>
#include<dos.h>

int n,bias,d[41][41];
void bias_removal();


void main()
  {
    struct time t;
    float t1,t2;
    int i,j,k,l,i1,i2,i3,seed,temp,index,dis,temp1,check,elt,bound;
    int min,ass[46],str[46],y[46],p[46],vv[46],dist[46][46];
    int x[46][46],z[46][46];

    printf(" ENTER THE No. OF CITIES AND A SEED :");
     scanf("%d%d",&n,&seed);

//  RANDOM MATRIX GENERATION
    srand((unsigned)seed);
    for(i=1;i<=n;i++)
     for(j=1;j<=n;j++)
      d[i][j]=(rand()%100)+1;
    for(i=1;i<=n;i++)
     d[i][i]=999;

    gettime(&t);
    t1= 3600*t.ti_hour + 60*t.ti_min + t.ti_sec + 0.01*t.ti_hund;

    bias_removal();
    printf("   BIAS=%d",bias);

//  ALPHABET TABLE
    for(i=1;i<=n;i++)
     { for(j=1;j<=n;j++)
      vv[j]=0;
       for(j=1;j<=n;j++)
      { min=999;
        for(k=1;k<=n;k++)
         if((vv[k]==0)&&(d[i][k]<min))
          { index=k;
            min=d[i][k];
          }
         z[i][j]=index;
         vv[index]=1;
      }
     }

    for(i=1;i<=n;i++)
     for(j=1;j<n;j++)
      for(k=j+1;k<=n;k++)
       if(d[i][j]>d[i][k])
```

```
        { temp=d[i][j];
          d[i][j]=d[i][k];
          d[i][k]=temp;
        }

      for(i=1;i<=n;i++)
       for(j=1;j<=n;j++)
        { dist[j][i]=d[i][j];
        x[j][i]=z[i][j];
        }

//  MAIN PROGRAM STARTS HERE
       min=900;
       for(i=1;i<=n;i++)
        { vv[i]=0;
        y[i]=0;
        p[i]=0;
        }
       dis=j=i=0;
 GS:   j=j+1;
       i=y[j]+1;
 NC:   temp=p[j];
       temp1=x[i][j];
       check=dis+dist[i][j];
       if(check>=min)
        goto JO;
       if(vv[temp1]==1)
 JB:    { i=i+1;
        goto NC;
        }

//     CYCLE CHECKING
       if(temp1<j && j!=n)
        { k=temp1;
        for(i1=1;i1<j;i1++)
         { if(p[k]<j)
             k=p[k];
           else
             if(p[k]==j)
              goto JB;
             else
              goto BD;
         }
        }
    BD:

//     BOUND CALCULATION
       bound=0;
       for(i1=j+1;i1<=n;i1++)
        { for(i2=1;i2<n;i2++)
         { elt=x[i2][i1];
           if((elt!=temp1) && (vv[elt]==0))
            { bound+=dist[i2][i1];
              goto next_col;
            }
         }
        next_col:
      }
     bound+=check;
     if(bound>=min)
      goto JB;
     p[j]=temp1;
     if(temp!=0)
      y[j+1]=0;
     y[j]=i;
     vv[temp1]=1;
     dis=check;
     if(dis>min)
      goto JO;
     if(j==n)
      {
      for(i1=1;i1<=n;i1++)
       ass[i1]=p[i1];
      min=dis;
 PREV:      dis-=dist[i][j];
      vv[p[j]]=0;
 JO:    j=j-1;
      if(j<1)
       goto STOP;
      vv[p[j]]=0;
      dis-=dist[y[j]][j];
      i=y[j]+1;
      goto NC;
      }
     goto GS;

  STOP:

      str[1]=1;
      for(i=2;i<=n;i++)
       { k=str[i-1];
       str[i]=ass[k];
       }

      gettime(&t);
      t2= 3600*t.ti_hour + 60*t.ti_min + t.ti_sec + 0.01*t.ti_hund;
      min+=bias;

      printf("\n  N=%d SEED=%d SOL=%d TIME=%.2f ",n,seed,min,t2-t1);
}
```

// REMOVING BIAS

```
void bias_removal()
  { int i,j,rmin[41],cmin[41];
    for(i=1;i<=n;i++)
     { rmin[i]=9999;
       for(j=1;j<=n;j++)
      if(d[i][j]<rmin[i])
       rmin[i]=d[i][j];
     }
    for(i=1;i<=n;i++)
     for(j=1;j<=n;j++)
      d[i][j]-=rmin[i];
    for(j=1;j<=n;j++)
```

```
   { cmin[j]=9999;
      for(i=1;i<=n;i++)
    if(d[i][j]<cmin[j])
      cmin[j]=d[i][j];
   }
  for(j=1;j<=n;j++)
   for(i=1;i<=n;i++)
    d[i][j]-=cmin[j];
  bias=0;
  for(i=1;i<=n;i++)
   bias+=rmin[i]+cmin[i];
}
```

```
// PROGRAM 1.2
// USUAL TSP USING DATA-GIUDED LEXI-SEARCH APPROACH

#include<stdio.h>
#include<stdlib.h>
#include<dos.h>

void main()
  {
    struct time t;
    float t1,t2;
    int i,j,k,l,l1,l2,m,m1,j2,n,min,seed,temp,bias,big,sml,remb;
    int bond,king,k1,d[46][46],x[46][46],ind[46][46],dest[46];
    int digit[46],seq[46],flag[46],p[46],mrow[46],ord[46],nzr[46];
    int index[46],rmin[46],cmin[46],post[46],bnd[46],vv[46];

    printf(" ENTER THE No. OF CITIES AND A SEED:");
     scanf("%d%d",&n,&seed);

//  RANDOM MATRIX GENERATION
    srand((unsigned)seed);
    for(i=1;i<=n;i++)
     for(j=1;j<=n;j++)
      d[i][j]=(rand()%100)+1;
    for(i=1;i<=n;i++)
     d[i][i]=999;

    gettime(&t);
    t1= 3600*t.ti_hour + 60*t.ti_min + t.ti_sec + 0.01*t.ti_hund;

//  CALCULATING THE MINIMUM ELEMENTS OF ROWS
    for(i=1;i<=n;i++)
     { rmin[i]=999999;
       for(j=1;j<=n;j++)
      if(d[i][j]<rmin[i])
       rmin[i]=d[i][j];
     }
    for(i=1;i<=n;i++)
     for(j=1;j<=n;j++)
      d[i][j]-=rmin[i];

//  CALCULATING THE MINIMUM ELEMENTS OF COLUMNS
    for(j=1;j<=n;j++)
     { cmin[j]=999999;
       for(i=1;i<=n;i++)
      if(d[i][j]<cmin[j])
       cmin[j]=d[i][j];
     }
    for(j=1;j<=n;j++)
     { mrow[j]=999999;
       index[j]=j;
       nzr[j]=0;
     }

//  REMOVING BIAS
    bias=0;
    for(i=1;i<=n;i++)
```

```
        { bias+=rmin[i]+cmin[i];
          for(j=1;j<=n;j++)
         { d[i][j]-=cmin[j];
           if(d[i][j]==0)
            { nzr[j]+=1;
              goto next1;
            }
           if(mrow[i]>d[i][j])
            mrow[i]=d[i][j];
           next1:
         }
        }

//    ALPHABET TABLE
      for(i=1;i<=n;i++)
        flag[i]=0;
      for(i=1;i<=n;i++)
        { big=0;
          sml=999999;
          for(j=1;j<=n;j++)
         { if(flag[j]==1)
             goto next2;
           if(nzr[j]<sml)
             sml=nzr[j];
           next2:
         }
          for(j=1;j<=n;j++)
         { if((flag[j]==1)||(nzr[j]!=sml))
             goto next3;
           if(mrow[j]>big)
            { big=mrow[j];
              remb=j;
            }
           next3:
         }
          flag[remb]=1;
          ord[i]=remb;
          index[remb]=i;
        }

      for(i=1;i<=n;i++)
        { k=ord[i];
          for(j=1;j<=n;j++)
         { x[i][j]=d[k][j];
           ind[k][j]=index[j];
         }
        }

      for(l=1;l<=n;l++)
        for(i=1;i<n;i++)
         for(j=i+1;j<=n;j++)
          if(x[l][i]>=x[l][j])
         { temp=x[l][i];
           x[l][i]=x[l][j];
           x[l][j]=temp;
           temp=ind[l][i];
           ind[l][i]=ind[l][j];
```

```
            ind[l][j]=temp;
          }
        for(i=1;i<=n;i++)
         for(j=1;j<=n;j++)
          if(j>i)
           { temp=x[i][j];
           x[i][j]=x[j][i];
           x[j][i]=temp;
           temp=ind[i][j];
           ind[i][j]=ind[j][i];
           ind[j][i]=temp;
           }

//    MAIN PROGRAM STARTS HERE
      for(i=1;i<=n;i++)
       dest[i]=digit[i]=post[i]=bnd[i]=flag[i]=0;
      min=900;
      post[1]=0;
 N1:  post[1]+=1;
      k=post[1];
      j2=ind[k][1];
      m=2;
      m1=1;
      bnd[1]=x[k][1];
      dest[1]=j2;
      digit[1]=j2;
      flag[j2]=1;
      if(post[1]==n)
       goto N10;
 N2:  for(l=m;l<=n;l++)
       { bond=0;
       for(l1=l+1;l1<=n;l1++)
        { for(l2=1;l2<n;l2++)
           { j2=ind[l2][l1];
             if(flag[j2]==1)
              goto next4;
             bond+=x[l2][l1];
              goto next5;
             next4:
           }
          next5:
        }
       for(k=m1;k<=n;k++)
        { if((bnd[l-1]+bond+x[k][l])>=min)
           goto N11;
          j2=ind[k][l];
          if(flag[j2]==1)
           goto next6;
          king=j2;
          if(l==n)
           goto N5;
        N4: if(dest[king]==1)
           goto next6;
          if(dest[king]==0)
           goto N5;
          king=dest[king];
           goto N4;
          next6:
         }
     N5: post[l]=k;
        dest[l]=j2;
        digit[l]=j2;
        flag[j2]=1;
        bnd[l]=bnd[l-1]+x[k][l];
        m1=1;
        if(l<n)
         goto next7;
        min=bnd[n];
        for(i=1;i<=n;i++)
         seq[i]=digit[i];
        goto N11;
        next7:
        }
N11:    k1=post[l];
        if(k1==0)
         goto N15;
        j2=ind[k1][l];
        post[l]=0;
        bnd[l]=0;
        dest[l]=0;
        flag[j2]=0;
N15:    l--;
        k1=post[l];
        j2=ind[k1][l];
        flag[j2]=0;
        bnd[l]=0;
        dest[l]=0;
        if(l==1)
         goto N1;
        post[l]=0;
        m=l;
        m1=k1+1;
        if(m1<n)
         goto N2;
        goto N11;
N10:    for(i=1;i<=n;i++)
         k=seq[i];
         p[1]=1;
         for(i=1;i<=n;i++)
        { k=seq[i];
          post[i]=ord[k];
        }
         for(i=1;i<=n;i++)
         for(j=1;j<=n;j++)
          if(p[i]==ord[j])
           p[i+1]=post[j];
        gettime(&t);
        t2= 3600*t.ti_hour + 60*t.ti_min + t.ti_sec + 0.01*t.ti_hund;
        min+=bias;
        printf("\n  N=%d SEED=%d SOL=%d TIME=%.2f ",n,seed,min,t2-t1);

}
```

```
// PROGRAM 1.3
// USUAL TRAVELLING SALESMAN PROBLEM USING LEXI-SEARCH
// PATH APPROACH

#include<stdio.h>
#include<stdlib.h>
#include<dos.h>

int n,bias,d[41][41];
void bias_removal();

void main()
  {
    struct time t;
    float t1,t2;
    int i,j,k,l,i1,i2,i3,seed,temp,index,dis,temp1,check,elt,bound,min;
    int str[41],y[41],p[41],vv[41],x[41][41];
    printf(" ENTER THE No. OF CITIES AND A SEED :");
     scanf("%d%d",&n,&seed);

//  RANDOM MATRIX GENERATION
    srand((unsigned)seed);
    for(i=1;i<=n;i++)
     for(j=1;j<=n;j++)
      d[i][j]=(rand()%100)+1;
    for(i=1;i<=n;i++)
     d[i][i]=999;

    gettime(&t);
    t1= 3600*t.ti_hour + 60*t.ti_min + t.ti_sec + 0.01*t.ti_hund;

//  CALLING BIAS REMOVAL FUNCTION
    bias_removal();

//  ALPHABET TABLE
    for(i=1;i<=n;i++)
     { for(j=1;j<=n;j++)
      vv[j]=0;
       for(j=1;j<=n;j++)
      { min=999;
        for(k=1;k<=n;k++)
         if((vv[k]==0)&&(d[i][k]<min))
          { index=k;
            min=d[i][k];
          }
         x[i][j]=index;
         vv[index]=1;
      }
     }

//  MAIN PROGRAM STARTS HERE
     min=900;
     for(j=1;j<=n;j++)
      { vv[j]=0;
      y[j]=0;
      p[j]=0;
      }
     dis=0;
     i=k=p[1]=vv[1]=1;
 GS: k=k+1;
     j=y[k]+1;
 NC: temp=p[k];
     temp1=x[i][j];
     check=dis+d[p[k-1]][temp1];
     if(check>=min)
      goto JO;
     if(vv[temp1]==1)
 JB:  { j=j+1;
      goto NC;
      }

//   BOUND CALCULATION
     bound=0;
     for(i1=1;i1<=n;i1++)
      { if(vv[i1]==0)
       { for(i2=1;i2<n/5;i2++)
          { elt=x[i1][i2];
            if((elt!=temp1) && ((vv[elt]==0)||(elt==1)))
             { bound+=d[i1][elt];
             goto next_row;
             }
          }
         elt=x[i1][n/5];
         bound+=d[i1][elt];
       }
      next_row:
      }
     bound+=check;
     if(bound>=min)
      goto JB;
     p[k]=temp1;
     if(temp!=0)
      y[k+1]=0;
     y[k]=j;
     vv[temp1]=1;
     dis=check;
     i=p[k];
     if(k==n)
      { dis+=d[temp1][1];
      if(dis>min)
       goto PREV;
      for(j=1;j<=n;j++)
       str[j]=p[j];
      min=dis;
 PREV:  dis-=d[temp1][1];
      dis-=d[p[k-1]][temp1];
      vv[temp1]=0;
 JO:    k=k-1;
      if(k<=1)
       goto STOP;
      vv[p[k]]=0;
      dis-=d[p[k-1]][p[k]];
      j=y[k]+1;
      i=p[k-1];
      goto NC;
      }
     goto GS;

  STOP:
      gettime(&t);
      t2= 3600*t.ti_hour + 60*t.ti_min + t.ti_sec + 0.01*t.ti_hund;
      min+=bias;

      printf("\n  N=%d SEED=%d SOL=%d TIME=%.2f",n,seed,min,t2-t1);
}

//  REMOVING BIAS
void bias_removal()
  { int i,j,rmin[41],cmin[41];
    for(i=1;i<=n;i++)
     { rmin[i]=9999;
       for(j=1;j<=n;j++)
      if(d[i][j]<rmin[i])
       rmin[i]=d[i][j];
     }
    for(i=1;i<=n;i++)
     for(j=1;j<=n;j++)
      d[i][j]-=rmin[i];
    for(j=1;j<=n;j++)
     { cmin[j]=9999;
       for(i=1;i<=n;i++)
      if(d[i][j]<cmin[j])
       cmin[j]=d[i][j];
     }
    for(j=1;j<=n;j++)
     for(i=1;i<=n;i++)
      d[i][j]-=cmin[j];
    bias=0;
    for(i=1;i<=n;i++)
     bias+=rmin[i]+cmin[i];
  }
```

```
// PROGRAM 1.4
// USUAL TSP USING SEQUENTIAL CONSTRUCTIVE SAMPLING APPROACH

#include<stdio.h>
#include<stdlib.h>
#include<dos.h>

void main()
 {
   struct time t;
   int n,iter,i,j,k,l,i1,i2,i3,dis,min,idx1,idx2,seed,lex,temp;
   int elt,bound,c[1601],vv[41],v[41][41],d[41][41],x[41][41];
   int pop,value1,value2,str[41],p[41],y[41],temp1,row[1601],col[1601];
   float t1,t2,sum,rnd,py[41];
   printf(" ENTER THE No. OF CITIES, SEED AND No. OF SAMPLES :");
    scanf("%d%d%d",&n,&seed,&pop);

//  RANDOM MATRIX GENERATION
    srand((unsigned)seed);
    for(i=1;i<=n;i++)
     for(j=1;j<=n;j++)
      d[i][j]=(rand()%100)+1;
    for(i=1;i<=n;i++)
     d[i][i]=999;
    gettime(&t);
    t1= 3600*t.ti_hour + 60*t.ti_min + t.ti_sec + 0.01*t.ti_hund;

//  ALPHABET TABLE
    for(i=1;i<=n;i++)
     { for(j=1;j<=n;j++)
      vv[j]=0;
       for(j=1;j<=n;j++)
      { min=9999;
        for(k=1;k<=n;k++)
         if((vv[k]==0)&&(d[i][k]<min))
          { idx1=k;
            min=d[i][k];
          }
         x[i][j]=idx1;
         vv[idx1]=1;
      }
     }

// ALPHABET TABLE FOR BOUND CALCULATION
   for(i=1;i<=n;i++)
    for(j=1;j<=n;j++)
     v[i][j]=0;
      for(i=1;i<=n*n;i++)
      { min=9999;
        for(j=1;j<=n;j++)
         for(k=1;k<=n;k++)
          if((v[j][k]==0)&&(d[j][k]<min))
          { idx1=j;
            idx2=k;
            min=d[j][k];
          }
         row[i]=idx1;
```

```
          col[i]=idx2;
          v[idx1][idx2]=1;
        }
    c[0]=0;
    for(i=1;i<=n*n;i++)
     { elt=d[row[i]][col[i]];
       if(elt<999)
        c[i]=c[i-1]+elt;
     }

// MAIN PROGRAM
    min=9999;
    for(i=1;i<=pop;i++)
     { for(j=1;j<=n;j++)
        { vv[j]=0;
        str[j]=0;
        }
       dis=0;
       str[1]=vv[1]=1;
       for(j=2;j<=n;j++)
        { l=0;
        for(k=1;k<=n;k++)
         y[k]=0;
        i1=n-j+1;
        temp=str[j-1];
        for(k=1;k<=n;k++)
         { elt=x[temp][k];
           if((vv[elt]==0) && (d[temp][elt]<999))
            { l=l+1;
              y[l]=elt;
              if(l>i1/10.0+1)
             goto NEXT;
            }
         }
        NEXT:
        py[0]=0;
        sum=l*(l+1)/2.0;
        for(k=1;k<=l;k++)
         py[k]=py[k-1]+ (l-k+1)/sum;
        i3=0;
       REPT: rnd=(rand()%100)*0.01;
        i3++;
        for(k=1;k<=l;k++)
         if(rnd>=py[k-1] && rnd<py[k])
          { str[j]=y[k];
            goto EXIT;
          }
       EXIT:
        dis+=d[temp][str[j]];
        if(dis>=min)
         goto OUT;
        temp1=str[j];

// BOUND CALCULATION
        bound=0;
        for(i1=1;i1<=10;i1++)
         { if(vv[row[i1]]==0)
```

```
          { if(vv[col[i1]]==0 || col[i1]==1)
           bound=c[j+i1-1]-c[i1-1];
             goto next_row;
            }
        }
       bound=c[j+10]-c[10];
   next_row:
       bound+=dis;
       if(bound>=min && i3<1)
        goto REPT;
       vv[str[j]]=1;
       }
       dis+=d[str[n]][1];
       if(dis<min)
        { min=dis;
          for(j=1;j<=n;j++)
           p[j]=str[j];
        }
      OUT:
       }

// MODIFIED 2-OPT MOVE
p[n+1]=1;
for(j=2;j<n-1;j++)
for(k=j+2;k<=n;k++)
{value1=d[p[j-1]][p[j]]+d[p[j]][p[j+1]]+d[p[k-1]][p[k]]+d[p[k]][p[k+1]];
 value2=d[p[j-1]][p[k]]+d[p[k]][p[j+1]]+d[p[k-1]][p[j]]+d[p[j]][p[k+1]];
  if(value2<value1)
   { temp=p[k];
     p[k]=p[j];
     p[j]=temp;
    }
}
     min=0;
     for(j=1;j<=n;j++)
      min+=d[p[j]][p[j+1]];

      gettime(&t);
      t2= 3600*t.ti_hour + 60*t.ti_min + t.ti_sec + 0.01*t.ti_hund;

      printf("\n  N=%d SEED=%d SOL=%d TIME=%.2f ",n,seed,min,t2-t1);

}
```

```
//  PROGRAM 1.5
//  USUAL TSP USING HYBRID GENETIC ALGORITHM

#include<stdio.h>
#include<stdlib.h>
#include<dos.h>
#include<conio.h>

int n,pop,bias,dis,chr[601][41],d[41][41];

void bias_removal();
void create();
void objective(int);
void repro_duction();
void cross_over(float);
void mutation(float);
void seq_search();

void main()
  {
    struct time t;
    float pcr,pmt,t1,t2;
    int i,j,min,str[41],vv[41],seed,gen;

    printf("ENTER THE No. OF CITY and A SEED RESPECTIVELY: ");
     scanf("%d%d",&n,&seed);
    printf("ENTER THE POPULATION SIZE : ");
     scanf("%d",&pop);
    printf("ENTER THE PROB. OF CROSSOVER AND MUTATION RESPECTIVELY: ");
     scanf("%f%f",&pcr,&pmt);

//  RANDOM MATRIX GENERATION
    srand((unsigned)seed);
    for(i=1;i<=n;i++)
     for(j=1;j<=n;j++)
      d[i][j]=(rand()%100)+1;
    for(i=1;i<=n;i++)
     d[i][i]=999;

    gettime(&t);
    t1= 3600*t.ti_hour + 60*t.ti_min + t.ti_sec + 0.01*t.ti_hund;

//  CALLING BIAS REMOVAL FUNCTION
    bias_removal();

//  TERMINATING CRITERION
    min=d[1][1];
    create();
    for(i=1;i<=pop;i++)
     { objective(i);
       if(dis<min)
      { min=dis;
        for(j=1;j<=n;j++)
         str[j]=chr[i][j];
      }
     }
    for(gen=1;gen<=4*n;gen++)
```

```
      { repro_duction();
        cross_over(pcr);
        mutation(pmt);
        seq_search();
        for(i=1;i<=pop;i++)
         { objective(i);
           if(dis<min)
            min=dis;
         }
      }
     min+=bias;
     gettime(&t);
     t2= 3600*t.ti_hour + 60*t.ti_min + t.ti_sec + 0.01*t.ti_hund;

     printf("\n N=%d Best Solution=%d Time=%.2f \n",n,min,t2-t1);
   }

// REMOVING BIAS
void bias_removal()
   { int i,j,rmin[41],cmin[41];
     for(i=1;i<=n;i++)
      { rmin[i]=9999;
        for(j=1;j<=n;j++)
       if(d[i][j]<rmin[i])
        rmin[i]=d[i][j];
      }
     for(i=1;i<=n;i++)
      for(j=1;j<=n;j++)
       d[i][j]-=rmin[i];
     for(j=1;j<=n;j++)
      { cmin[j]=9999;
        for(i=1;i<=n;i++)
       if(d[i][j]<cmin[j])
        cmin[j]=d[i][j];
      }
     for(j=1;j<=n;j++)
      for(i=1;i<=n;i++)
       d[i][j]-=cmin[j];
     bias=0;
     for(i=1;i<=n;i++)
      bias+=rmin[i]+cmin[i];
   }

// INITIALIZE THE POPULATION
void create()
  { int m,i,j,i1,elt,count,vv[41];
    for(i=1;i<=pop;i++)
     { for(j=1;j<=n;j++)
        { vv[j]=0;
        chr[i][j]=0;
        }
       chr[i][1]=1;
       vv[1]=1;
       for(j=2;j<=n;j++)
        { elt=rand() % (n-j+1) +1;
        count=0;
        for(i1=2;i1<=n;i1++)
```

```
          if(vv[i1]==0)
           { count++;
             if(count==elt)
              { chr[i][j]=i1;
              goto rxx;
              }
           }
         rxx:
         vv[i1]=1;
         }
     }
 }

// CALCULATE THE VALUE OF THE OBJECTIVE FUNCTION
void objective(int i)
  { int j;
    dis=0;
    for(j=1;j<n;j++)
     dis+=d[chr[i][j]][chr[i][j+1]];
    dis+=d[chr[i][n]][1];
  }

// STOCHASTIC REMAINDER SELECTION METHOD
void repro_duction()
 {  float fit[601],random,avg,expect,sum=0,frac[601];
    int i,j,k,assign,select,choice[601];
    for(i=1;i<=pop;i++)
     { objective(i);
       fit[i]=1.0/(float)(dis+1);
       sum+=fit[i];
     }
    avg=sum/pop;
    k=0;
    for(i=1;i<=pop;i++)
     { expect=fit[i]/avg;
       assign=expect;
       frac[i]=expect-assign;
       while(assign>0)
      { k=k+1;
        assign-=1;
        choice[k]=i;
      }
     }
    i=0;
    while(k<pop)
     { i=i+1;
       if(i>pop)
      i=1;
       if(frac[i]>0.0)
      { random=(rand()%1000)*0.001;
        if(frac[i]>random)
         { k=k+1;
           choice[k]=i;
           frac[i]-=1.0;
         }
      }
     }
    for(i=1;i<=pop;i++)
     { select=choice[i];
       for(j=1;j<=n;j++)
      chr[i][j]=chr[select][j];
     }
 }
```

```
// C1-CROSSOVER OPERATION
 void cross_over(float pc)
  { int i,j,k,m,cross,min,cost,v1[41],v2[41],x,y,nwchr[21][41];
    float random;
    for(i=1;i<pop;i++)
     { random=(rand()%100)*0.01;
       if(random<pc)
      { for(m=1;m<=20;m=m+2)
         { for(j=1;j<=n;j++)
            { v1[j]=0;
            v2[j]=0;
            }
           cross=(rand()%(n-2))+2;
           for(j=1;j<cross;j++)
            { x=chr[i][j];
            y=chr[i+1][j];
            nwchr[m][j]=x;
            nwchr[m+1][j]=y;
            v1[x]=1;
            v2[y]=1;
            }
           k=0;
           for(j=cross;j<=n;j++)
    read1:     { k=k+1;
            x=chr[i+1][k];
            if(v1[x]==1)
             goto read1;
            v1[x]=1;
            nwchr[m][j]=x;
            }
           k=0;
           for(j=cross;j<=n;j++)
    read2:     { k=k+1;
            y=chr[i][k];
            if(v2[y]==1)
             goto read2;
            v2[y]=1;
            nwchr[m+1][j]=y;
            }
         }
        min=999;
        for(m=1;m<=20;m++)
         { cost=0;
           for(j=1;j<n;j++)
            cost+=d[nwchr[m][j]][nwchr[m][j+1]];
           cost+=d[nwchr[m][n]][1];
           if(cost<min)
            { min=cost;
            for(j=1;j<=n;j++)
             chr[i][j]=nwchr[m][j];
```

```
              }
           }
        }
      }
    }

// BITWISE MUTATION OPERATION
  void mutation(float pm)
   { int temp,m,i,j,r1,r2;
     float random;
      for(i=1;i<=pop;i++)
       { m=rand()%n+1;
         for(j=1;j<=m;j=j+2)
          { random=(rand()%100)*0.01;
            if(random<pm)
             { r1=rand()%(n-1)+2;
             r2=rand()%(n-1)+2;
             temp=chr[i][r1];
             chr[i][r1]=chr[i][r2];
             chr[i][r2]=temp;
             }
          }
       }
    }

// SEQUENTIAL CONSTRAUCTIVE SEARCH APPROACH
void seq_search()
  { int i,j,k,l,m,min,temp,index,x[3],cost[3],v[51],nwchr[51];
    for(i=1;i<pop;i++)
     { for(j=1;j<=n;j++)
      { v[j]=0;
        nwchr[j]=0;
      }
       nwchr[1]=v[1]=1;
       for(k=2;k<=n;k++)
      { for(l=0;l<=1;l++)
         { x[l]=0;
           cost[l]=999;
           for(j=1;j<=n;j++)
            { if(chr[i+l][j]==nwchr[k-1])
             { for(m=1;m<=n;m++)
                { if(j+m>n)
                   goto next;
                  if(v[chr[i+l][j+m]]==0)
                   { cost[l]=d[chr[i+l][j]][chr[i+l][j+m]];
                   x[l]=chr[i+l][j+m];
                   goto next;
                   }
                }
             }
            }
           next:
         }
        if(cost[1]==999 && cost[0]==999)
         { min=999;
           for(j=1;j<=n;j++)
            if(v[j]==0)
```

```
               { temp=d[nwchr[k-1]][j];
               if(temp<min)
                { nwchr[k]=j;
                  min=temp;
                }
               }
             goto ACCEPT;
           }
          nwchr[k]=x[0];
          if(cost[1]<cost[0])
           nwchr[k]=x[1];
ACCEPT:     v[nwchr[k]]=1;
        }
        for(j=1;j<=n;j++)
         chr[i][j]=nwchr[j];
       }
 }
```

```
// SUB-ROUTINE 1.1
// EDGE-RECOMBINATION CROSSOVER OPERATOR
 void cross_over()
  {
    int i,j,k,l,m,min,count,temp,temp1,temp2,index[41];
    int x[41][6],nwchr[41],v[51],w[41],node[4];
    for(i=1;i<pop;i++)
     { chr[i][n+1]=1;
       chr[i+1][n+1]=1;
       chr[i][0]=chr[i][n];
       chr[i+1][0]=chr[i+1][n];

//  CREATING THE EDGE MAP
       for(k=1;k<=n;k++)
      { for(j=1;j<=n;j++)
         w[j]=0;
        for(j=1;j<=5;j++)
         x[k][j]=0;
        count=0;
        for(l=0;l<=1;l++)
         { for(j=1;j<=n;j++)
            { if(chr[i+l][j]==k)
             { if(w[chr[i+l][j-1]]==0)
                { count+=1;
                  temp=chr[i+l][j-1];
                  x[k][count]=temp;
                  w[temp]=1;
                }
               if(w[chr[i+l][j+1]]==0)
                { count+=1;
                  temp=chr[i+l][j+1];
                  x[k][count]=temp;
                  w[temp]=1;
                }
             }
            }
         }
      }
//  EDGE MAP IS CRAETED

//  BUILDING NEW CHROMOSOMES
       nwchr[1]=v[1]=1;  //STARTING NODE 1 //
       for(j=2;j<=n;j++)
      v[j]=0;
       for(m=2;m<=n;m++)
      { for(k=1;k<=n;k++)
         if(v[k]==0)
          { for(j=1;j<=4;j++)
             if(x[k][j]==nwchr[m-1])
            { for(l=j+1;l<=4;l++)
               x[k][l-1]=x[k][l];
              x[k][4]=0;
              goto DELETE_NEXT;
            }
            DELETE_NEXT:
          }
//  COUNTING EDGES
        for(k=1;k<=n;k++)
         { for(j=1;j<=5;j++)
            if(x[k][j]==0)
             { index[k]=j-1;
             goto NEXT_INDEX;
             }
           NEXT_INDEX:
         }

//  EDGE COUNT ENDS HERE

//  ACCEPTING THE NODE WHICH HAS MINIMUM EDGES
       temp=nwchr[m-1];
       min=5;
       if(index[temp]>0)
        for(k=1;k<=index[temp];k++)
         { temp1=x[temp][k];
           if(index[temp1]<min)
            { min=index[temp1];
            temp2=temp1;
            count=0;
            }
           if(index[temp1]==min)
            { node[count]=temp2;
            count+=1;
            }
         }
        if(count>0)
         count=rand()%count;
        temp=node[count];
        if(v[temp]==1)
         for(j=1;j<=n;j++)
          if(v[j]==0)
           { temp=j;
             goto ACCEPT;
           }
  ACCEPT: nwchr[m]=temp;
        v[temp]=1;
      }
       for(j=1;j<=n;j++)
      chr[i][j]=nwchr[j];
     }
  }
```

```
// SUB-ROUTINE 1.2
// GENERALIZED N POINT CROSSOVER OPERATOR

 void cross_over()
  {
    int i,j,k,l,nx,point,temp,start,end;
    int xpoint[41],segment[11],nwchr[41],v[51],w[41];
    for(i=1;i<pop;i++)
     {
       nx=n/5;

//  RANDOMLY GENERATING N CROSSOVER POINT
       xpoint[0]=0;
       xpoint[nx+1]=n;
       for(j=1;j<=nx;j++)
      xpoint[j]=xpoint[j-1]+rand()%3+2;

//  RANDOMLY ORDERING THE SEGMENTS TO BE TESTED
       segment[0]=0;
       for(j=1;j<=nx+1;j++)
      w[j]=0;
       for(j=1;j<=nx+1;j++)
 REPEAT:{ temp=rand()%(nx+1)+1;
        if(w[temp]==1)
         goto REPEAT;
        segment[j]=temp;
        w[temp]=1;
      }

//  CREATING THE OFFSPRING
       for(j=1;j<=n;j++)
      { v[j]=0;
        nwchr[j]=0;
      }
      for(l=1;l<=2;l++)
       for(j=1;j<=nx+1;j++)
      { point=segment[j];
        start=xpoint[point-1]+1;
        end=xpoint[point];
        if((point%2==0 && l==1) || (point%2==1 && l==2))
         { for(k=start;k<=end;k++)
            { temp=chr[i][k];
            if(v[temp]==0 && nwchr[k]==0)
             { nwchr[k]=temp;
               v[temp]=1;
             }
            }
         }
        else
         { for(k=start;k<=end;k++)
            { temp=chr[i+1][k];
            if(v[temp]==0 && nwchr[k]==0)
             { nwchr[k]=temp;
               v[temp]=1;
             }
            }
         }
        }

//  FILLING UP THE GENES WHICH ARE BLANK
         for(j=1;j<=n;j++)
        if(nwchr[j]==0)
         { for(k=1;k<=n;k++)
            if(v[k]==0)
             { nwchr[j]=k;
               v[k]=1;
               goto NEXT;
             }
           NEXT:
         }
         for(j=1;j<=n;j++)
        chr[i][j]=nwchr[j];
       }
   }
```

```
// PROGRAM 2.1
// TSP WITH PRECEDENCE CONSTRAINTS USING LEXI-SEARCH
// PATH APPROACH

#include<stdio.h>
#include<stdlib.h>
#include<dos.h>

int n,bias,d[41][41];
void bias_removal();

void main()
  {
    struct time t;
    float t1,t2;
    int i,j,k,l,i1,i2,nres,seed,temp,index,dis,temp1,check,elt;
    int a[15],b[15],idx[15],y[41],p[41],str[41],vv[41];
    int c[41][41],x[41][41],min,bound;

    printf(" ENTER THE No. OF CITIES AND A SEED :");
     scanf("%d%d",&n,&seed);
    printf(" ENTER THE No. OF CONSTRAINTS:");
     scanf("%d",&nres);
    printf(" ENTER THE CONSTRAINTS IN THE FORM A < B : ");
    for(i=1;i<=nres;i++)
     scanf("%d%d",&a[i],&b[i]);

// ARRANGE THE NODES THAT ARE INVOLVED IN THE RESTRICTIONS
   for(i=1;i<=nres;i++)
    for(j=1;j<=nres;j++)
     if(a[i]==b[j])
      { nres++;
      a[nres]=a[j];
      b[nres]=b[i];
      }

    for(i=1;i<=n;i++)
     for(j=1;j<=n;j++)
      c[i][j]=0;
    for(i=1;i<=n;i++)
     { i1=0;
       for(j=1;j<=n;j++)
      vv[j]=0;
       for(j=1;j<=nres;j++)
      if(i==b[j] && vv[a[j]]==0)
       { i1++;
         c[i][i1]=a[j];
         vv[a[j]]=1;
       }
      }

//  RANDOM MATRIX GENERATION
    srand((unsigned)seed);
    for(i=1;i<=n;i++)
     for(j=1;j<=n;j++)
      d[i][j]=(rand()%100)+1;
    for(i=1;i<=n;i++)
```

```
     d[i][i]=999;

   gettime(&t);
   t1= 3600*t.ti_hour + 60*t.ti_min + t.ti_sec + 0.01*t.ti_hund;

// MAKE THE DISTANCE BETWEEN b & a TO LARGE ENOUGH
    for(i=1;i<=nres;i++)
     { d[b[i]][a[i]]=999;
       d[1][b[i]]=999;
       d[a[i]][1]=999;
     }

// CALLING THE BIAS REMOVAL FUNCTION
    bias_removal();

// THE ALPHABET TABLE
    for(i=1;i<=n;i++)
     { for(j=1;j<=n;j++)
      vv[j]=0;
       for(j=1;j<=n;j++)
      { min=999;
        for(k=1;k<=n;k++)
         if((vv[k]==0)&&(d[i][k]<min))
          { index=k;
            min=d[i][k];
          }
         x[i][j]=index;
         vv[index]=1;
      }
     }

// MAIN PROGRAM STARTS HERE
   min=900;
   for(j=1;j<=n;j++)
    { vv[j]=0;
      y[j]=0;
      p[j]=0;
    }
   dis=0;
   p[1]=vv[1]=i=k=1;
 GS: k=k+1;
     j=y[k]+1;
 NC: temp=p[k];
     temp1=x[i][j];
     check=dis+d[p[k-1]][temp1];
     if(check>=min)
      goto JO;
     if(vv[temp1]==1)
      goto JB;
     for(i1=1;i1<=n;i1++)
      { if(temp1==i1)
       for(i2=1;i2<=n;i2++)
        { if(c[i1][i2]==0)
           goto BND;
          if(vv[c[i1][i2]]==0)
 JB:       { j=j+1;
             goto NC;
          }
       }
     }

// BOUND CALCULATION
    BND:
    bound=0;
    for(i1=1;i1<=n;i1++)
     { if(vv[i1]==0)
      { for(i2=1;i2<n/5;i2++)
         { elt=x[i1][i2];
           if((elt!=temp1)&&((vv[elt]==0)||(elt==1)))
            { bound+=d[i1][elt];
            goto next_row;
            }
         }
        elt=x[i1][n/5];
        bound+=d[i1][elt];
      }
     next_row:
     }
    bound+=check;
    if(bound>=min)
     goto JB;
    p[k]=temp1;
    if(temp!=0)
     y[k+1]=0;
    y[k]=j;
    vv[temp1]=1;
    dis=check;
    i=p[k];
    if(k==n)
     { dis+=d[temp1][1];
     if(dis>=min)
      goto PREV;
     for(j=1;j<=n;j++)
      str[j]=p[j];
     min=dis;

 PREV:  dis-=d[temp1][1];
     dis-=d[p[k-1]][temp1];
     vv[temp1]=0;
 JO:  k=k-1;
     if(k<=1)
      goto STOP;
     vv[p[k]]=0;
     dis-=d[p[k-1]][p[k]];
     j=y[k]+1;
     i=p[k-1];
     goto NC;
     }
    goto GS;

  STOP:
     gettime(&t);
     t2= 3600*t.ti_hour + 60*t.ti_min + t.ti_sec + 0.01*t.ti_hund;
     min+=bias;
```

```
      printf("\n  N=%d SEED=%d SOL=%d TIME=%.2f ",n,seed,min,t2-t1);
}

//  REMOVING BIAS
void bias_removal()
  { int i,j,rmin[41],cmin[41];
    for(i=1;i<=n;i++)
     { rmin[i]=999;
       for(j=1;j<=n;j++)
      if(d[i][j]<rmin[i])
       rmin[i]=d[i][j];
     }
    for(i=1;i<=n;i++)
     for(j=1;j<=n;j++)
      d[i][j]-=rmin[i];
    for(j=1;j<=n;j++)
     { cmin[j]=999;
       for(i=1;i<=n;i++)
      if(d[i][j]<cmin[j])
       cmin[j]=d[i][j];
     }
    for(j=1;j<=n;j++)
     for(i=1;i<=n;i++)
      d[i][j]-=cmin[j];
    bias=0;
    for(i=1;i<=n;i++)
     bias+=rmin[i]+cmin[i];
  }
```

```
// PROGRAM 2.2
// TSP WITH PRECEDENCE CONSTRAINTS USING LEXI-SEARCH
// ADJACENCY APPROACH

#include<stdio.h>
#include<stdlib.h>
#include<dos.h>

int n,bias,d[41][41];
void bias_removal();


void main()
  {
    struct time t;
    float t1,t2;
    int i,j,k,l,i1,i2,i3,seed,temp,index,dis,temp1,check,elt,bound;
    int min,iter,v[41],ass[41],str[41],y[41],p[41],vv[41],dist[41][41];
    int nres,a[15],b[15],c[41][41],x[41][41],z[41][41];

    printf(" ENTER THE No. OF CITIES AND A SEED :");
     scanf("%d%d",&n,&seed);
    printf(" ENTER THE No. OF CONSTRAINTS:");
     scanf("%d",&nres);
    printf(" ENTER THE CONSTRAINTS IN THE FORM A < B : ");
    for(i=1;i<=nres;i++)
     scanf("%d%d",&a[i],&b[i]);


// ARRANGE THE NODES THAT ARE INVOLVED IN THE RESTRICTIONS
   for(i=1;i<=nres;i++)
    for(j=1;j<=nres;j++)
     if(a[i]==b[j])
      { nres++;
      a[nres]=a[j];
      b[nres]=b[i];
      }

    for(i=1;i<=n;i++)
     for(j=1;j<=n;j++)
      c[i][j]=0;
    for(i=1;i<=n;i++)
     { i1=0;
       for(j=1;j<=n;j++)
      vv[j]=0;
       for(j=1;j<=nres;j++)
      if(i==b[j] && vv[a[j]]==0)
       { i1++;
         c[i][i1]=a[j];
         vv[a[j]]=1;
       }
      }

//  RANDOM MATRIX GENERATION
    srand((unsigned)seed);
    for(i=1;i<=n;i++)
     for(j=1;j<=n;j++)
```

```
      d[i][j]=(rand()%100)+1;
    for(i=1;i<=n;i++)
     d[i][i]=999;

  gettime(&t);
  t1= 3600*t.ti_hour + 60*t.ti_min + t.ti_sec + 0.01*t.ti_hund;

// MAKE THE DISTANCE BETWEEN b & a TO LARGE ENOUGH
    for(i=1;i<=nres;i++)
     { d[b[i]][a[i]]=999;
       d[1][b[i]]=999;
       d[a[i]][1]=999;
     }

// CALLING THE BIAS REMOVAL FUNCTION
    bias_removal();

// THE ALPHABET TABLE
    for(i=1;i<=n;i++)
     { for(j=1;j<=n;j++)
      vv[j]=0;
       for(j=1;j<=n;j++)
      { min=999;
        for(k=1;k<=n;k++)
         if((vv[k]==0)&&(d[i][k]<min))
          { index=k;
            min=d[i][k];
          }
         z[i][j]=index;
         vv[index]=1;
      }
     }

    for(i=1;i<=n;i++)
     for(j=1;j<n;j++)
      for(k=j+1;k<=n;k++)
       if(d[i][j]>d[i][k])
      { temp=d[i][j];
        d[i][j]=d[i][k];
        d[i][k]=temp;
      }

    for(i=1;i<=n;i++)
     for(j=1;j<=n;j++)
      { dist[j][i]=d[i][j];
      x[j][i]=z[i][j];
      }

//  MAIN PROGRAM STARTS HERE
     min=900;
     for(i=1;i<=n;i++)
      { vv[i]=0;
      y[i]=0;
      p[i]=0;
      }
     dis=j=i=0;
 GS: j=j+1;
```

```
    i=y[j]+1;
NC: temp=p[j];
    temp1=x[i][j];
    check=dis+dist[i][j];
    if(check>=min)
     goto JO;
    if(vv[temp1]==1)
JB:  { i=i+1;
     goto NC;
     }

//  CYCLE CHECKING
    if(temp1<j && j!=n)
     { k=temp1;
     for(i1=1;i1<j;i1++)
      { if(p[k]<j)
          k=p[k];
        else
         if(p[k]==j)
          goto JB;
         else
          goto BD;
      }
     }
  BD:

//  BOUND CALCULATION
    bound=0;
    for(i1=j+1;i1<=n;i1++)
     { for(i2=1;i2<n;i2++)
      { elt=x[i2][i1];
        if((elt!=temp1) && (vv[elt]==0))
         { bound+=dist[i2][i1];
           goto next_col;
         }
      }
     next_col:
     }
    bound+=check;
    if(bound>=min)
     goto JB;
    p[j]=temp1;
    if(temp!=0)
     y[j+1]=0;
    y[j]=i;
    vv[temp1]=1;
    dis=check;
    if(dis>min)
     goto JO;
    if(j==n)
     {
     ass[1]=1;
     for(i1=2;i1<=n;i1++)
      { k=ass[i1-1];
        ass[i1]=p[k];
        v[i1]=0;
      }
//  CHECKING FOR CONSTRAINTS
      for(i1=1;i1<=n;i1++)
       { v[ass[i1]]=1;
         for(i2=1;i2<=n;i2++)
          { if(c[ass[i1]][i2]==0)
              goto OK;
            if(v[c[ass[i1]][i2]]==0)
              goto PREV;
          }
        OK:
       }
      min=dis;
      for(i1=1;i1<=n;i1++)
       str[i1]=ass[i1];
 PREV:      dis-=dist[i][j];
      vv[p[j]]=0;
 JO:    j=j-1;
      if(j<1)
       goto STOP;
      vv[p[j]]=0;
      dis-=dist[y[j]][j];
      i=y[j]+1;
      goto NC;
      }
     goto GS;
  STOP:
      gettime(&t);
      t2= 3600*t.ti_hour + 60*t.ti_min + t.ti_sec + 0.01*t.ti_hund;
      min+=bias;
      printf("\n  N=%d SEED=%d SOL=%d TIME=%.2f ",n,seed,min,t2-t1);
}

//  REMOVING BIAS
void bias_removal()
  { int i,j,rmin[41],cmin[41];
    for(i=1;i<=n;i++)
     { rmin[i]=9999;
       for(j=1;j<=n;j++)
      if(d[i][j]<rmin[i])
       rmin[i]=d[i][j];
     }
    for(i=1;i<=n;i++)
     for(j=1;j<=n;j++)
      d[i][j]-=rmin[i];
    for(j=1;j<=n;j++)
     { cmin[j]=9999;
       for(i=1;i<=n;i++)
      if(d[i][j]<cmin[j])
       cmin[j]=d[i][j];
     }
    for(j=1;j<=n;j++)
     for(i=1;i<=n;i++)
      d[i][j]-=cmin[j];
    bias=0;
    for(i=1;i<=n;i++)
     bias+=rmin[i]+cmin[i];
  }
```

```
// PROGRAM 2.3
// TSP WITH PRECEDENCE CONSTRAINTS
// USING SEQUENTIAL CONSTRUCTIVE SAMPLING APPROACH

#include<stdio.h>
#include<stdlib.h>
#include<dos.h>

void main()
 {
  struct time t;
  int n,i,j,k,l,i1,i2,i3,i4,nres,dis,min,idx1,idx2,seed,temp,elt,bound;
  int cl[1601],vv[41],v[41][41],d[41][41],x[41][41],str[41],p[41],y[41];
  int temp1,value1,value2,pop,row[1601],col[1601],c[41][41],a[15],b[15];
   float t1,t2,sum,rnd,py[41];
   printf(" ENTER THE No. OF CITIES, A SEED AND No. OF SAMPLES :");
    scanf("%d%d%d",&n,&seed,&pop);
   printf(" ENTER THE No. OF CONSTRAINTS:");
    scanf("%d",&nres);
   printf(" ENTER THE CONSTRAINTS IN THE FORM A < B : ");
   for(i=1;i<=nres;i++)
    scanf("%d%d",&a[i],&b[i]);

// THE PREDECESSOR TABLE
   for(i=1;i<=nres;i++)
    for(j=1;j<=nres;j++)
     if(a[i]==b[j])
      { nres++;
      a[nres]=a[j];
      b[nres]=b[i];
      }

   for(i=1;i<=n;i++)
    for(j=1;j<=n;j++)
     c[i][j]=0;
   for(i=1;i<=n;i++)
    { i1=0;
      for(j=1;j<=n;j++)
       vv[j]=0;
      for(j=1;j<=nres;j++)
       if(i==b[j] && vv[a[j]]==0)
      { i1++;
        c[i][i1]=a[j];
        vv[a[j]]=1;
      }
    }

//  RANDOM MATRIX GENERATION
    srand((unsigned)seed);
    for(i=1;i<=n;i++)
     for(j=1;j<=n;j++)
      d[i][j]=(rand()%100)+1;
    for(i=1;i<=n;i++)
     d[i][i]=999;
    gettime(&t);
    t1= 3600*t.ti_hour + 60*t.ti_min + t.ti_sec + 0.01*t.ti_hund;

// MAKE THE DISTANCE BETWEEN b & a TO LARGE ENOUGH
   for(i=1;i<=nres;i++)
    { d[b[i]][a[i]]=999;
      d[1][b[i]]=999;
      d[a[i]][1]=999;
    }

// THE ALPHABET TABLE
   for(i=1;i<=n;i++)
    { for(j=1;j<=n;j++)
       vv[j]=0;
      for(j=1;j<=n;j++)
       { min=9999;
       for(k=1;k<=n;k++)
        if((vv[k]==0)&&(d[i][k]<min))
         { idx1=k;
           min=d[i][k];
         }
       x[i][j]=idx1;
       vv[idx1]=1;
       }
    }

// ALPHABET TABLE FOR BOUND CALCULATION
   for(i=1;i<=n;i++)
    for(j=1;j<=n;j++)
     v[i][j]=0;
      for(i=1;i<=n*n;i++)
      { min=9999;
        for(j=1;j<=n;j++)
         for(k=1;k<=n;k++)
          if((v[j][k]==0)&&(d[j][k]<min))
          { idx1=j;
            idx2=k;
            min=d[j][k];
          }
         row[i]=idx1;
         col[i]=idx2;
         v[idx1][idx2]=1;
      }
   c1[0]=0;
   for(i=1;i<=n*n;i++)
    { elt=d[row[i]][col[i]];
      if(elt<999)
       c1[i]=c1[i-1]+elt;
    }

// MAIN PROGRAM
   min=9999;
   for(i=1;i<=pop;i++)
    { for(j=1;j<=n;j++)
       { vv[j]=0;
       str[j]=0;
       }
      dis=0;
      str[1]=vv[1]=1;
      for(j=2;j<=n;j++)
       { l=0;
       for(k=1;k<=n;k++)
        y[k]=0;
       i4=n-j+1;
       temp=str[j-1];
       for(k=1;k<=n;k++)
        { elt=x[temp][k];
          if((vv[elt]==0) && (d[temp][elt]<999))
           { for(i2=1;i2<=n;i2++)
             { if(c[elt][i2]==0)
                 goto NEXT1;
               if(vv[c[elt][i2]]==0)
                 goto NEXT2;
             }
           }
          goto NEXT2;
 NEXT1:     l=l+1;
          y[l]=elt;
          if(l>i4/10.0+1)
           goto NEXT3;
 NEXT2:
        }
 NEXT3: py[0]=0;
       sum=l*(l+1)/2.0;
       for(k=1;k<=l;k++)
        py[k]=py[k-1]+ (l-k+1)/sum;
       i3=0;
 REPT:  rnd=(rand()%100)*0.01;
       i3++;
       if(i3>l)
        goto OUT;
       for(k=1;k<=l;k++)
        if(rnd>=py[k-1] && rnd<py[k])
         { templ=y[k];
           goto EXIT;
         }
 EXIT:  dis+=d[temp][templ];
       if(dis>=min)
        goto REPT;

// BOUND CALCULATION
       bound=0;
       for(il=1;il<=10;il++)
        { if(vv[row[il]]==0)
           { if(vv[col[il]]==0 || col[il]==1)
            bound=cl[j+il-1]-cl[il-1];
             goto next_row;
           }
        }
       bound=cl[j+10]-cl[10];
       next_row:
       bound+=dis;
       if(bound>=min)
        goto REPT;
        str[j]=temp1;
        vv[temp1]=1;
        }
       dis+=d[str[n]][1];
       if(dis<min)
        { min=dis;
        for(j=1;j<=n;j++)
         p[j]=str[j];
        }
  OUT:
    }

// MODIFIED 2-OPT MOVE
p[n+1]=1;
for(j=2;j<n-1;j++)
 for(k=j+2;k<=n;k++)
  { temp=0;
    for(i=1;i<=nres;i++)
     if(p[j]==a[i] && p[k]==b[i])
      temp=1;
    if(temp==0)
{value1=d[p[j-1]][p[j]]+d[p[j]][p[j+1]]+d[p[k-1]][p[k]]+d[p[k]][p[k+1]];
 value2=d[p[j-1]][p[k]]+d[p[k]][p[j+1]]+d[p[k-1]][p[j]]+d[p[j]][p[k+1]];
 if(value2<value1)
   { temp=p[k];
     p[k]=p[j];
     p[j]=temp;
   }
}
  }

  min=0;
  for(j=1;j<=n;j++)
   min+=d[p[j]][p[j+1]];

  gettime(&t);
  t2= 3600*t.ti_hour + 60*t.ti_min + t.ti_sec + 0.01*t.ti_hund;
  printf("\n N=%d SEED=%d SOL=%d TIME=%.2f",n,seed,min,t2-t1);
}
```

```
// PROGRAM 2.4
// TSP WITH PRECEDENCE CONSTRAINTS
// USING HYBRID GENETIC ALGORITHM

#include<stdio.h>
#include<stdlib.h>
#include<dos.h>

int n,nres,pop,bias,a[15],b[15],dis,c[41][41],chr[451][41],d[41][41];

void bias_removal();
void create();
void objective(int);
void repro_duction();
void cross_over(float);
void mutation(float);
void seq_search();


void main()
  {
    struct time t;
    float pcr,pmt,t1,t2;
    int i,j,i1,min,nres,str[72],vv[72],seed,iter,gen;

    printf("ENTER THE No. OF CITY and A SEED RESPECTIVELY: ");
     scanf("%d%d",&n,&seed);
    printf("ENTER No.OF CONSTRAINTS AND POPULATION SIZE RESPECTIVELY:");
    scanf("%d%d",&nres,&pop);
    printf("ENTER THE PROB. OF CROSSOVER AND MUTATION RESPECTIVELY: ");
     scanf("%f%f",&pcr,&pmt);
    printf(" ENTER THE CONSTRAINTS IN THE FORM A < B : ");
    for(i=1;i<=nres;i++)
     scanf("%d%d",&a[i],&b[i]);

// ARRANGE THE NODES THAT ARE INVOLVED IN THE RESTRICTIONS
   for(i=1;i<=nres;i++)
    for(j=1;j<=nres;j++)
     if(a[i]==b[j])
      { nres++;
      a[nres]=a[j];
      b[nres]=b[i];
      }

    for(i=1;i<=n;i++)
     for(j=1;j<=n;j++)
      c[i][j]=0;
    for(i=1;i<=n;i++)
     { i1=0;
       for(j=1;j<=n;j++)
      vv[j]=0;
       for(j=1;j<=nres;j++)
      if(i==b[j] && vv[a[j]]==0)
       { i1++;
         c[i][i1]=a[j];
         vv[a[j]]=1;
       }
      }

//   RANDOM MATRIX GENERATION
     srand((unsigned)seed);
     for(i=1;i<=n;i++)
      for(j=1;j<=n;j++)
       d[i][j]=(rand()%100)+1;
     for(i=1;i<=n;i++)
      d[i][i]=999;

     gettime(&t);
     t1= 3600*t.ti_hour + 60*t.ti_min + t.ti_sec + 0.01*t.ti_hund;

// MAKE THE DISTANCE BETWEEN b & a TO LARGE ENOUGH
     for(i=1;i<=nres;i++)
      d[b[i]][a[i]]=999;

//   CALLING BIAS REMOVAL FUNCTION
     bias_removal();

//   TERMINATING CRITERION
     min=d[1][1];
     create();
     for(i=1;i<=pop;i++)
      { objective(i);
        if(dis<min)
       { min=dis;
         for(j=1;j<=n;j++)
          str[j]=chr[i][j];
       }
      }
     for(gen=1;gen<=4*n;gen++)
      { repro_duction();
        cross_over(pcr);
        mutation(pmt);
        seq_search();
        for(i=1;i<=pop;i++)
       { objective(i);
         if(dis<min)
          min=dis;
       }
      }
     min+=bias;
     gettime(&t);
     t2= 3600*t.ti_hour + 60*t.ti_min + t.ti_sec + 0.01*t.ti_hund;

     printf("\n N=%d Best Solution=%d Time=%.2f \n",n,min,t2-t1);
   }

//   REMOVING BIAS
void bias_removal()
   { int i,j,rmin[41],cmin[41];
     for(i=1;i<=n;i++)
      { rmin[i]=9999;
        for(j=1;j<=n;j++)
       if(d[i][j]<rmin[i])
        rmin[i]=d[i][j];
      }
     for(i=1;i<=n;i++)
      for(j=1;j<=n;j++)
       d[i][j]-=rmin[i];
     for(j=1;j<=n;j++)
      { cmin[j]=9999;
        for(i=1;i<=n;i++)
       if(d[i][j]<cmin[j])
        cmin[j]=d[i][j];
      }
     for(j=1;j<=n;j++)
      for(i=1;i<=n;i++)
       d[i][j]-=cmin[j];
     bias=0;
     for(i=1;i<=n;i++)
      bias+=rmin[i]+cmin[i];
   }

// INITIALIZE THE POPULATION
void create()
 { int i,j,i1,i2,elt,count,temp,vv[41];
   for(i=1;i<=pop;i++)
    { for(j=1;j<=n;j++)
       { vv[j]=0;
       chr[i][j]=0;
       }
      chr[i][1]=1;
      vv[1]=1;
      for(j=2;j<=n;j++)
 RPT:  { elt=rand() % (n-j+1) +1;
       count=0;
       for(i1=2;i1<=n;i1++)
        if(vv[i1]==0)
         { count++;
           if(count==elt)
            { temp=i1;
            goto rxx;
            }
         }
   rxx:  for(i1=1;i1<=n;i1++)
        { if(temp==i1)
           for(i2=1;i2<=n;i2++)
            { if(c[i1][i2]==0)
             goto ACCEPT;
            if(vv[c[i1][i2]]==0)
             goto RPT;
            }
        }
 ACCEPT: chr[i][j]=temp;
       vv[temp]=1;
       }
    }
 }

// CALCULATE THE VALUE OF THE OBJECTIVE FUNCTION
void objective(int i)
  { int j;
```

```
    dis=0;
    for(j=1;j<n;j++)
     dis+=d[chr[i][j]][chr[i][j+1]];
    dis+=d[chr[i][n]][1];
  }

// STOCHASTIC REMAINDER SELECTION METHOD
void repro_duction()
 {  float fit[451],random,avg,expect,sum=0,frac[451];
    int i,j,k,assign,select,choice[451];
    for(i=1;i<=pop;i++)
     { objective(i);
       fit[i]=1.0/(float)(dis+1);
       sum+=fit[i];
     }
    avg=sum/pop;
    k=0;
    for(i=1;i<=pop;i++)
     { expect=fit[i]/avg;
       assign=expect;
       frac[i]=expect-assign;
       while(assign>0)
      { k=k+1;
        assign-=1;
        choice[k]=i;
      }
     }
    i=0;
    while(k<pop)
     { i=i+1;
       if(i>pop)
      i=1;
       if(frac[i]>0.0)
      { random=(rand()%1000)*0.001;
        if(frac[i]>random)
         { k=k+1;
           choice[k]=i;
           frac[i]-=1.0;
         }
      }
     }
    for(i=1;i<=pop;i++)
     { select=choice[i];
       for(j=1;j<=n;j++)
      chr[i][j]=chr[select][j];
     }
 }

// C1-CROSSOVER OPERATION
void cross_over(float pc)
  { int i,j,k,m,cross,min,cost,v1[41],v2[41],x,y,nwchr[21][41];
    float random;
    for(i=1;i<pop;i++)
     { random=(rand()%100)*0.01;
       if(random<pc)
      { for(m=1;m<=20;m=m+2)
         { for(j=1;j<=n;j++)
```

```
          { v1[j]=0;
          v2[j]=0;
          }
         cross=(rand()%(n-2))+2;
         for(j=1;j<cross;j++)
          { x=chr[i][j];
          y=chr[i+1][j];
          nwchr[m][j]=x;
          nwchr[m+1][j]=y;
          v1[x]=1;
          v2[y]=1;
          }
         k=0;
         for(j=cross;j<=n;j++)
    read1:     { k=k+1;
          x=chr[i+1][k];
          if(v1[x]==1)
           goto read1;
          v1[x]=1;
          nwchr[m][j]=x;
          }
         k=0;
         for(j=cross;j<=n;j++)
    read2:     { k=k+1;
          y=chr[i][k];
          if(v2[y]==1)
           goto read2;
          v2[y]=1;
          nwchr[m+1][j]=y;
          }
        }
       min=9999;
       for(m=1;m<=20;m++)
        { cost=0;
          for(j=1;j<n;j++)
           cost+=d[nwchr[m][j]][nwchr[m][j+1]];
          cost+=d[nwchr[m][n]][1];
          if(cost<min)
           { min=cost;
           for(j=1;j<=n;j++)
            chr[i][j]=nwchr[m][j];
           }
        }
      }
     }
    }

// BITWISE MUTATION OPERATION
void mutation(float pm)
   { int temp,i,i1,j,rv[41];
     float random;
     for(i=1;i<=pop;i++)
      for(j=2;j<=n;j++)
       { for(i1=1;i1<=nres;i1++)
        rv[b[i1]]=1;
       if(j<n)
        { random=(rand()%100)*0.01;
```

```
          if(random<pm)
           { if((rv[chr[i][j]]==1)||(rv[chr[i][j+1]]==1))
            goto next;
             temp=chr[i][j];
             chr[i][j]=chr[i][j+1];
             chr[i][j+1]=temp;
           }
        }
      next:
       }
   }

// SEQUENTIAL CONSTRUCTIVE SEARCH APPROACH
void seq_search()
  { int i,j,k,l,m,x[3],v[41],p[3][41],nwchr[41];
    int i1,i2,temp;
    long min,cost[3];
    for(i=1;i<pop;i++)
     { for(j=1;j<=n;j++)
      { p[1][j]=chr[i][j];
        p[2][j]=chr[i+1][j];
        v[j]=0;
      }
       nwchr[1]=1;
       v[1]=1;
       for(k=2;k<=n;k++)
      { for(l=1;l<=2;l++)
         { for(j=1;j<=n;j++)
            { if(p[l][j]==nwchr[k-1])
             { for(m=1;m<=n;m++)
                { if(j+m>n)
                   { cost[l]=999;
                   x[l]=p[l][j];
                   goto next;
                   }
                  if(v[p[l][j+m]]==0)
                   { cost[l]=d[p[l][j]][p[l][j+m]];
                   x[l]=p[l][j+m];
                   for(i1=1;i1<=n;i1++)
                    { if(x[l]==i1)
                       for(i2=1;i2<=n;i2++)
                        { if(c[i1][i2]==0)
                         goto next;
                        if(v[c[i1][i2]]==0)
                        goto RPT;
                        }
                    }
                   }
                  RPT:
                }
             }
            }
           next:
         }
        if((cost[1]==999) && (cost[2]==999))
         for(j=1;j<=n;j++)
          { if(v[p[1][j]]==0)
```

```
              { temp=p[1][j];
              for(i1=1;i1<=n;i1++)
               { if(temp==i1)
                  for(i2=1;i2<=n;i2++)
                   { if(c[i1][i2]==0)
                    goto ACCEPT;
                   if(v[c[i1][i2]]==0)
                    goto RPT1;
                   }
               }
              }
             RPT1:
           }
        temp=x[1];
        if(cost[2]<cost[1])
          temp=x[2];
ACCEPT:   nwchr[k]=temp;
        v[temp]=1;
      }
     for(k=1;k<=n;k++)
      chr[i][k]=nwchr[k];
     }
  }
```

```
// PROGRAM 3.1
// TSP WITH FIXED POSITION CONSTRAINTS USING LEXI-SEARCH
// PATH APPROACH

#include<stdio.h>
#include<stdlib.h>
#include<dos.h>

int n,bias,d[41][41];
void bias_removal();

void main()
  {
    struct time t;
    float t1,t2;
    int i,j,k,l,i1,i2,i3,i4,i5,nres,seed,temp,index,dis,temp1,check,elt;
    int p[21],q[21],r[41],str[41],vv[41],y[41];
    int x[41][41],bv[41],pv[41],min,bound,iter;

    printf(" ENTER THE No. OF CITIES AND A SEED RESPECTIVELY : ");
     scanf("%d%d",&n,&seed);
    printf(" ENTER THE No. OF CONSTRAINTS : ");
     scanf("%d",&nres);
    printf(" ENTER THE PRESCRIBED POSITIONS : ");
    for(i=1;i<=nres;i++)
     scanf("%d",&p[i]);
    printf(" ENTER THE CORRESPONDING NODES : ");
    for(i=1;i<=nres;i++)
     scanf("%d",&q[i]);

//  RANDOM MATRIX GENERATION
    srand((unsigned)seed);
    for(i=1;i<=n;i++)
     for(j=1;j<=n;j++)
      d[i][j]=(rand()%100)+1;
    for(i=1;i<=n;i++)
     d[i][i]=999;

// MAKE THE DISTANCE BETWEEN TWO POSSIBLE NODES TO LARGE ENOUGH
    for(i=0;i<nres;i++)
     for(j=i+1;j<=nres;j++)
      if(p[j]-p[i]>=2)
       { d[q[i]][q[j]]=999;
       if(q[j]-q[i]==n-1)
        goto DONOT;
       d[q[j]][q[i]]=999;
       DONOT:
       }

   gettime(&t);
   t1= 3600*t.ti_hour + 60*t.ti_min + t.ti_sec + 0.01*t.ti_hund;

//  CALLING BIAS REMOVAL FUNCTION
    bias_removal();

//  THE ALPHABET TABLE
    for(i=1;i<=n;i++)
```

```
     { for(j=1;j<=n;j++)
      vv[j]=0;
       for(j=1;j<=n;j++)
      { min=999;
        for(k=1;k<=n;k++)
          if((vv[k]==0)&&(d[i][k]<min))
           { index=k;
             min=d[i][k];
           }
          x[i][j]=index;
          vv[index]=1;
      }
     }

// MAIN PROGRAM STARTS HERE
   for(j=1;j<=n+1;j++)
    { vv[j]=0;
      y[j]=0;
      r[j]=0;
      bv[j]=0;
      pv[j]=0;
    }
   min=900;
   for(i=1;i<=nres;i++)
    { vv[q[i]]=1;
      pv[p[i]+1]=1;
    }
   dis=0;
   i=k=r[1]=vv[1]=bv[1]=1;
   p[0]=0;
   p[nres+1]=n;
   q[nres+1]=1;
   for(i1=0;i1<=nres;i1++)
    { for(i2=p[i1]+2;i2<=p[i1+1]+1;i2++)
       { j=y[i2]+1;
 NC:     i5=0;
       if(i2==p[i1+1]+1)
        { temp1=q[i1+1];
          temp=temp1;
          check=dis+d[r[i2-1]][temp1];
          if(check>=min)
           goto JO;
          if(i2==n+1)
           { min=check;
             for(i3=1;i3<=n;i3++)
            str[i3]=r[i3];
             goto JO;
           }
          i5=1;
          goto BOUND;
        }
       temp=r[i2];
       temp1=x[i][j];
       check=dis+d[r[i2-1]][temp1];
       if(check>=min)
 JO:      { i2--;
          if(i2<=p[i1]+1)
```

```
            i1--;
           if(i2<=1)
            goto STOP;
           dis-=d[r[i2-1]][r[i2]];
           bv[r[i2]]=0;
           vv[r[i2]]=0;
           if(pv[i2]==1)
            { vv[r[i2]]=1;
              goto JO;
            }
           j=y[i2]+1;
           i=r[i2-1];
           goto NC;
        }
       if(vv[templ]==1)
JB:        { j=j+1;
           if(j>=n || i5==1)
            goto JO;
           goto NC;
        }
BOUND:          bound=0;
       for(i3=1;i3<=n;i3++)
        { if(bv[i3]==0)
           { for(i4=1;i4<6;i4++)
            { elt=x[i3][i4];
              if((elt!=templ)&&((bv[elt]==0)||(elt==1)))
               { bound+=d[i3][elt];
                 goto next_row;
               }
            }
             elt=x[i3][6];
             bound+=d[i3][elt];
           }
          next_row:
        }
       elt=bound;
       bound+=check;
       if(bound>=min)
        goto JB;
       r[i2]=templ;
       if(temp!=0)
        y[i2+1]=0;
       y[i2]=j;
       vv[r[i2]]=1;
       bv[r[i2]]=1;
       dis=check;
       i=r[i2];
       }
    }

 STOP:
      gettime(&t);
      t2= 3600*t.ti_hour + 60*t.ti_min + t.ti_sec + 0.01*t.ti_hund;
      min+=bias;
      printf("\n  N=%d SEED=%d SOL=%d TIME=%.2f",n,seed,min,t2-t1);
}
```

```
//  REMOVING BIAS
void bias_removal()
  { int i,j,rmin[41],cmin[41];
    for(i=1;i<=n;i++)
     { rmin[i]=9999;
       for(j=1;j<=n;j++)
      if(d[i][j]<rmin[i])
       rmin[i]=d[i][j];
     }
    for(i=1;i<=n;i++)
     for(j=1;j<=n;j++)
      d[i][j]-=rmin[i];
    for(j=1;j<=n;j++)
     { cmin[j]=9999;
       for(i=1;i<=n;i++)
      if(d[i][j]<cmin[j])
       cmin[j]=d[i][j];
     }
    for(j=1;j<=n;j++)
     for(i=1;i<=n;i++)
      d[i][j]-=cmin[j];
    bias=0;
    for(i=1;i<=n;i++)
     bias+=rmin[i]+cmin[i];
  }
```

```
// PROGRAM 3.2
// TSP WITH FIXED POSITION CONSTRAINTS
// USING SEQUENTIAL CONSTRUCTIVE SAMPLING APPROACH

#include<stdio.h>
#include<stdlib.h>
#include<dos.h>

void main()
 {
   struct time t;
   int nl,n,iter,i,j,k,l,i1,i2,i3,i4,dis,min,idx1,idx2,seed,lex,temp;
   int elt,bound,c1[1601],vv[41],v[41][41],d[41][41],x[41][41],str[41];
   int r[41],y[41],temp1,row[1601],col[1601];
   int nres,c[41][41],p[15],q[15],bv[41],pop,value1,value2;
   float t1,t2,sum,rnd,py[41];
  printf(" ENTER No.OF CITIES,A SEED AND No.OF SAMPLES RESPECTIVELY:");
     scanf("%d%d%d",&n,&seed,&pop);
    printf(" ENTER THE No. OF CONSTRAINTS : ");
     scanf("%d",&nres);
    printf(" ENTER THE PRESCRIBED POSITIONS : ");
    for(i=1;i<=nres;i++)
     scanf("%d",&p[i]);
    printf(" ENTER THE CORRESPONDING NODES : ");
    for(i=1;i<=nres;i++)
     scanf("%d",&q[i]);

//  RANDOM MATRIX GENERATION
    srand((unsigned)seed);
    for(i=1;i<=n;i++)
     for(j=1;j<=n;j++)
      d[i][j]=(rand()%100)+1;
    for(i=1;i<=n;i++)
     d[i][i]=999;


   gettime(&t);
   t1= 3600*t.ti_hour + 60*t.ti_min + t.ti_sec + 0.01*t.ti_hund;

// MAKE THE DISTANCE BETWEEN TWO POSSIBLE NODES TO LARGE ENOUGH
   for(i=0;i<nres;i++)
    for(j=i+1;j<=nres;j++)
     if(p[j]-p[i]>=2)
      { d[q[i]][q[j]]=999;
      if(q[j]-q[i]==n-1)
       goto DONOT;
      d[q[j]][q[i]]=999;
      DONOT:
      }

//  THE ALPHABET TABLE
    for(i=1;i<=n;i++)
     { for(j=1;j<=n;j++)
      vv[j]=0;
       for(j=1;j<=n;j++)
      { min=9999;
        for(k=1;k<=n;k++)
```

```
          if((vv[k]==0)&&(d[i][k]<min))
           { idx1=k;
             min=d[i][k];
           }
          x[i][j]=idx1;
          vv[idx1]=1;
      }
     }

// ALPHABET TABLE FOR BOUND CALCULATION
   for(i=1;i<=n;i++)
    for(j=1;j<=n;j++)
     v[i][j]=0;
      for(i=1;i<=n*n;i++)
      { min=9999;
        for(j=1;j<=n;j++)
         for(k=1;k<=n;k++)
          if((v[j][k]==0)&&(d[j][k]<min))
          { idx1=j;
            idx2=k;
            min=d[j][k];
          }
        row[i]=idx1;
        col[i]=idx2;
        v[idx1][idx2]=1;
      }
   cl[0]=0;
   for(i=1;i<=n*n;i++)
    { elt=d[row[i]][col[i]];
      if(elt<999)
       cl[i]=cl[i-1]+elt;
    }

// MAIN PROGRAM
     min=9999;
      for(i=1;i<=pop;i++)
       { for(j=1;j<=n;j++)
        { vv[j]=0;
          str[j]=0;
          bv[j]=0;
        }
       for(i1=1;i1<=nres;i1++)
        vv[q[i1]]=1;
       dis=0;
       str[1]=vv[1]=bv[1]=1;
       for(j=2;j<=n;j++)
        { temp=str[j-1];
          for(i1=1;i1<=nres;i1++)
           if(j==p[i1]+1)
            { temp1=q[i1];
            dis+=d[temp][temp1];
            if(dis>=min)
             goto OUT;
            i3=-2;
            goto BND;
            }
          l=0;
          for(k=1;k<=n;k++)
           y[k]=0;
          i4=n-j+1;
          for(k=1;k<=n;k++)
           { elt=x[temp][k];
             if((vv[elt]==0) && (d[temp][elt]<999))
            { l=l+1;
              y[l]=elt;
              if(l>i4/10.0+1)
               goto NEXT;
            }
           }
     NEXT:  py[0]=0;
          sum=l*(l+1)/2.0;
          for(k=1;k<=l;k++)
           py[k]=py[k-1]+ (l-k+1)/sum;
          i3=0;
      REPT: i3++;
          if(i3>l || i3<0)
           goto OUT;
          rnd=(rand()%100)*0.01;
          for(k=1;k<=l;k++)
           if(rnd>=py[k-1] && rnd<py[k])
            { temp1=y[k];
            goto EXIT;
            }
      EXIT: dis+=d[temp][temp1];
          if(dis>=min)
           goto REPT;

// BOUND CALCULATION
      BND:  bound=0;
          for(i1=1;i1<=10;i1++)
           { if(bv[row[i1]]==0)
            { if(bv[col[i1]]==0 || col[i1]==1)
               bound=cl[j+i1-1]-cl[i1-1];
              goto next_row;
            }
           }
          bound=cl[j+10]-cl[10];
          next_row:
          bound+=dis;
          if(bound>=min)
           goto REPT;
          str[j]=temp1;
          vv[temp1]=1;
          bv[temp1]=1;
        }
       dis+=d[str[n]][1];
       if(dis<min)
        { min=dis;
          for(j=1;j<=n;j++)
           r[j]=str[j];
        }
       OUT:
       }
```

```
// MODIFIED 2-OPT MOVE
r[n+1]=1;
for(j=2;j<n-1;j++)
 for(k=j+2;k<=n;k++)
  { temp=0;
    for(i=1;i<=nres;i++)
     if(j==p[i]+1 || k==p[i]+1)
      temp=1;
    if(temp==0)
{value1=d[r[j-1]][r[j]]+d[r[j]][r[j+1]]+d[r[k-1]][r[k]]+d[r[k]][r[k+1]];
 value2=d[r[j-1]][r[k]]+d[r[k]][r[j+1]]+d[r[k-1]][r[j]]+d[r[j]][r[k+1]];
 if(value2<value1)
  { temp=r[k];
    r[k]=r[j];
    r[j]=temp;
  }
}
  }

     min=0;
     for(j=1;j<=n;j++)
      min+=d[r[j]][r[j+1]];


      gettime(&t);
      t2= 3600*t.ti_hour + 60*t.ti_min + t.ti_sec + 0.01*t.ti_hund;

      printf("\n  N=%d SEED=%d SOL=%d TIME=%.2f",n,seed,min,t2-t1);

}
```

```
//   PROGRAM 3.3
//   TSP WITH POSITION CONSTRAINTS
//   USING HYBRID GENETIC ALGORITHM

#include<stdio.h>
#include<stdlib.h>
#include<dos.h>

int n,nres,pop,dis,bias,p[15],q[15],chr[501][41],d[41][41];

void bias_removal();
void create();
void objective(int);
void repro_duction();
void cross_over(float);
void mutation(float);
void seq_search();

void main()
  {

    struct time t;
    float pcr,pmt,t1,t2;
    int i,j,min,seed,gen,str[72],vv[72];

    printf(" ENTER THE No. OF CITIES AND A SEED RESPECTIVELY : ");
     scanf("%d%d",&n,&seed);
    printf(" ENTER THE No. OF CONSTRAINTS : ");
     scanf("%d",&nres);
    printf(" ENTER THE PRESCRIBED POSITIONS : ");
    for(i=1;i<=nres;i++)
     scanf("%d",&p[i]);
    printf(" ENTER THE CORRESPONDING NODES : ");
    for(i=1;i<=nres;i++)
     scanf("%d",&q[i]);
 printf(" ENTER POP_SIZE,PROB.OF CROSSOVER AND MUTATION RESPECTIVELY:");
     scanf("%d%f%f",&pop,&pcr,&pmt);

//  RANDOM MATRIX GENERATION
    srand((unsigned)seed);
    for(i=1;i<=n;i++)
     for(j=1;j<=n;j++)
      d[i][j]=(rand()%100)+1;
    for(i=1;i<=n;i++)
     d[i][i]=999;

    gettime(&t);
    t1= 3600*t.ti_hour + 60*t.ti_min + t.ti_sec + 0.01*t.ti_hund;

//  CALLING BIAS REMOVAL FUNCTION
    bias_removal();

//  TERMINATING CRITERION
    min=d[1][1];
    create();
    for(i=1;i<=pop;i++)
     { objective(i);
```

```
       if(dis<min)
      { min=dis;
        for(j=1;j<=n;j++)
         str[j]=chr[i][j];
      }
     }
    for(gen=1;gen<=4*n;gen++)
     { repro_duction();
       cross_over(pcr);
       mutation(pmt);
       seq_search();
       for(i=1;i<=pop;i++)
      { objective(i);
        if(dis<min)
         min=dis;
      }
     }

    min+=bias;
    gettime(&t);
    t2= 3600*t.ti_hour + 60*t.ti_min + t.ti_sec + 0.01*t.ti_hund;

    printf("\nN=%d Best Solution=%d Time=%.2f ",n,min,t2-t1);
  }

//  REMOVING BIAS
void bias_removal()
  { int i,j,rmin[41],cmin[41];
    for(i=1;i<=n;i++)
     { rmin[i]=9999;
       for(j=1;j<=n;j++)
      if(d[i][j]<rmin[i])
       rmin[i]=d[i][j];
     }
    for(i=1;i<=n;i++)
     for(j=1;j<=n;j++)
      d[i][j]-=rmin[i];
    for(j=1;j<=n;j++)
     { cmin[j]=9999;
       for(i=1;i<=n;i++)
      if(d[i][j]<cmin[j])
       cmin[j]=d[i][j];
     }
    for(j=1;j<=n;j++)
     for(i=1;i<=n;i++)
      d[i][j]-=cmin[j];
    bias=0;
    for(i=1;i<=n;i++)
     bias+=rmin[i]+cmin[i];
  }

// INITIALIZE THE POPULATION
void create()
 { int m,i,j,k,elt,count,vv[41];
   for(i=1;i<=pop;i++)
    { for(j=1;j<=n;j++)
        vv[j]=0;
       for(j=1;j<=nres;j++)
        vv[q[j]]=1;
       chr[i][1]=1;
       vv[1]=1;
       m=n-nres-1;
       for(j=2;j<=n;j++)
        { for(k=1;k<=nres;k++)
         if(j==p[k]+1)
          { chr[i][j]=q[k];
            goto next2;
          }
        elt=(rand()%m)+1;
        count=0;
        for(k=2;k<=n;k++)
         if(vv[k]==0)
          { count++;
            if(count==elt)
             { chr[i][j]=k;
             goto next1;
             }
          }
        next1:
        vv[chr[i][j]]=1;
        m--;
        next2:
        }
     }
  }

// CALCULATE THE VALUE OF THE OBJECTIVE FUNCTION
void objective(int i)
  { int j;
    dis=0;
    for(j=1;j<n;j++)
     dis+=d[chr[i][j]][chr[i][j+1]];
    dis+=d[chr[i][n]][1];
  }

// STOCHASTIC REMAINDER SELECTION METHOD
void repro_duction()
 {  float fit[501],random,avg,expect,sum=0,frac[501];
    int i,j,k,assign,select,choice[501];
    for(i=1;i<=pop;i++)
     { fit[i]=0;
       objective(i);
       if(dis>0)
      fit[i]=1.0/(float)dis;
       sum+=fit[i];
     }
    avg=sum/pop;
    k=0;
    for(i=1;i<=pop;i++)
     { expect=fit[i]/avg;
       assign=expect;
       frac[i]=expect-assign;
       while(assign>0)
      { k=k+1;
```

```
          assign-=1;
          choice[k]=i;
        }
      }
    i=0;
    while(k<pop)
     { i=i+1;
       if(i>pop)
      i=1;
       if(frac[i]>0.0)
      { random=(rand()%1000)*0.001;
        if(frac[i]>random)
         { k=k+1;
           choice[k]=i;
           frac[i]-=1.0;
         }
      }
     }
    for(i=1;i<=pop;i++)
     { select=choice[i];
       for(j=1;j<=n;j++)
      chr[i][j]=chr[select][j];
     }
 }

// C1-CROSSOVER OPERATION
void cross_over(float pc)
  { int i,j,k,l,m,cross,min,cost,v1[41],v2[41],x,y,nwchr[21][41];
    float random;
    for(i=1;i<pop;i++)
     { random=(rand()%100)*0.01;
       if(random<pc)
      { for(m=1;m<=10;m=m+2)
         { for(j=1;j<=n;j++)
            { v1[j]=0;
            v2[j]=0;
            }
           cross=(rand()%(n-2))+2;
           for(j=1;j<cross;j++)
            { x=chr[i][j];
            y=chr[i+1][j];
            nwchr[m][j]=x;
            nwchr[m+1][j]=y;
            v1[x]=1;
            v2[y]=1;
            }
           k=0;
           for(j=cross;j<=n;j++)
    read1:     { for(l=1;l<=nres;l++)
              if(j==p[l]+1)
               { x=q[l];
                 goto next1;
               }
            k=k+1;
            x=chr[i+1][k];
            if(v1[x]==1)
             goto read1;
        next1:     v1[x]=1;
             nwchr[m][j]=x;
             }
            k=0;
            for(j=cross;j<=n;j++)
     read2:     { for(l=1;l<=nres;l++)
              if(j==p[l]+1)
               { y=q[l];
                 goto next2;
               }
             k=k+1;
             y=chr[i][k];
             if(v2[y]==1)
              goto read2;
        next2:     v2[y]=1;
             nwchr[m+1][j]=y;
             }
          }
         min=9999;
         for(m=1;m<=10;m++)
          { cost=0;
            for(j=1;j<n;j++)
             cost+=d[nwchr[m][j]][nwchr[m][j+1]];
            cost+=d[nwchr[m][n]][1];
            if(cost<min)
             { min=cost;
             for(j=1;j<=n;j++)
              chr[i][j]=nwchr[m][j];
             }
          }
      }
     }
   }

// BITWISE MUTATION OPERATION
void mutation(float pm)
   { int temp,i,i1,j,rv[41];
     float random;
     for(i=1;i<=pop;i++)
      for(j=2;j<n;j++)
       { random=(rand()%100)*0.01;
       if(random<pm)
        { for(i1=1;i1<=nres;i1++)
           if(j==p[i1]+1 || j+1==p[i1]+1)
            goto next;
          temp=chr[i][j];
          chr[i][j]=chr[i][j+1];
          chr[i][j+1]=temp;
        }
      next:
       }
   }


// SEQUENTIAL CONSTRUCTIVE SEARCH APPROACH
void seq_search()
  { int i,j,k,l,i1,i2,m,min,x[3],cost[3],v[41],r[3][41],nwchr[41];
  for(i=1;i<pop;i++)
   { for(j=1;j<=n;j++)
    { r[1][j]=chr[i][j];
      r[2][j]=chr[i+1][j];
      v[j]=0;
    }
     for(i1=1;i1<=nres;i1++)
    v[q[i1]]=1;
     nwchr[1]=1;
     v[1]=1;
     for(k=2;k<=n;k++)
    { for(i1=1;i1<=nres;i1++)
       if(k==p[i1]+1)
        { nwchr[k]=q[i1];
          goto FIXED;
        }
      for(l=1;l<=2;l++)
       { for(j=1;j<=n;j++)
          { if(r[l][j]==nwchr[k-1])
           { for(m=1;m<n;m++)
              { if(j+m>n)
                 { cost[l]=9999;
                 x[l]=r[l][j];
                 goto next;
                 }
                if(v[r[l][j+m]]==0)
                 { cost[l]=d[r[l][j]][r[l][j+m]];
                 x[l]=r[l][j+m];
                 goto next;
                 }
              }
           }
          }
         next:
       }
      nwchr[k]=x[1];
      if(cost[2]<cost[1])
       nwchr[k]=x[2];
      if(v[nwchr[k]]==1)
       for(j=1;j<=n;j++)
        if(v[j]==0)
         { nwchr[k]=j;
           goto FIXED;
         }
FIXED:  v[nwchr[k]]=1;
    }
   for(k=1;k<=n;k++)
    chr[i][k]=nwchr[k];
   }
}
```

```
// PROGRAM 4.1
// TSP WITH FIXED POSITION & PRECEDENCE CONSTRAINTS
// USING LEXI-SEARCH PATH APPROACH

#include<stdio.h>
#include<stdlib.h>
#include<dos.h>

int n,bias,d[41][41];
void bias_removal();

void main()
  {
    struct time t;
    float t1,t2;
    int i,j,k,l,i1,i2,i3,i4,i5,mres,nres,pres,seed,temp,index,dis;
    int temp1,check,elt,min,bound,itr,a[15],b[15],idx[15],y[41],r[41];
    int str[41],vv[41],c[41][41],x[41][41],p[11],q[11],pv[41],bv[41];
    printf(" ENTER THE No. OF CITIES AND A SEED : ");
     scanf("%d%d",&n,&seed);
    printf(" ENTER No.OF FIXED POSITION AND PRECEDENCE CONSTRAINTS:");
     scanf("%d%d",&mres,&nres);
    printf(" ENTER THE PRESCRIBED POSITIONS : ");
    for(i=1;i<=mres;i++)
     scanf("%d",&p[i]);
    printf(" ENTER THE CORRESPONDING NODES : ");
    for(i=1;i<=mres;i++)
     scanf("%d",&q[i]);
    printf(" ENTER THE PRECEDENCE CONSTRAINTS IN THE FORM A < B : ");
    for(i=1;i<=nres;i++)
     scanf("%d%d",&a[i],&b[i]);

//  MAKE THE PREDECESSOR TABLE
    for(i=1;i<=nres;i++)
     for(j=1;j<=nres;j++)
      if(a[i]==b[j])
       { nres++;
       a[nres]=a[j];
       b[nres]=b[i];
       }

    for(i=1;i<=nres;i++)
     if(a[i]==b[i])
      { printf("\n Solutions are not feasible\n");
      goto OUT;
      }

    pres=nres;

//  ARRANGE THE NODES THAT ARE INVOLVED IN THE PRECEDENCE RESTRICTIONS
    for(i=1;i<=n;i++)
     for(j=1;j<=n;j++)
      c[i][j]=0;
    for(i=1;i<=n;i++)
     { i1=0;
       for(j=1;j<=n;j++)
      vv[j]=0;
```

```
        for(j=1;j<=nres;j++)
       if(i==b[j] && vv[a[j]]==0)
        { i1++;
          c[i][i1]=a[j];
          vv[a[j]]=1;
        }
      }

//   RANDOM MATRIX GENERATION
     srand((unsigned)seed);
     for(i=1;i<=n;i++)
      for(j=1;j<=n;j++)
       d[i][j]=(rand()%100)+1;
     for(i=1;i<=n;i++)
      d[i][i]=999;

// MAKE THE DISTANCE BETWEEN b & a TO LARGE ENOUGH
     for(i=1;i<=pres;i++)
      { d[b[i]][a[i]]=999;
        d[1][b[i]]=999;
        d[a[i]][1]=999;
      }

//   MAKE THE DISTANCE BETWEEN TWO POSSIBLE NODES TO LARGE ENOUGH
     p[0]=1;
     q[0]=1;
     for(i=0;i<mres;i++)
      for(j=i+1;j<=mres;j++)
       if(p[j]-p[i]>=2)
        { d[q[i]][q[j]]=999;
        if(q[j]-q[i]==n-1)
         goto DONOT;
        d[q[j]][q[i]]=999;
        DONOT:
        }

    gettime(&t);
    t1= 3600*t.ti_hour + 60*t.ti_min + t.ti_sec + 0.01*t.ti_hund;

//   CALLING BIAS REMOVAL FUNCTION
     bias_removal();

//   ARRANGING THE INDICES IN ASCENDING ORDER
     for(i=1;i<=n;i++)
      { for(j=1;j<=n;j++)
       vv[j]=0;
        for(j=1;j<=n;j++)
       { min=999;
         for(k=1;k<=n;k++)
          if((vv[k]==0)&&(d[i][k]<min))
           { index=k;
             min=d[i][k];
           }
          x[i][j]=index;
          vv[index]=1;
       }
      }
```

```
// MAIN PROGRAM STARTS HERE
   for(j=1;j<=n+1;j++)
    { vv[j]=0;
      y[j]=0;
      r[j]=0;
      bv[j]=0;
      pv[j]=0;
    }
   min=9999;
   for(i=1;i<=mres;i++)
    { vv[q[i]]=1;
      pv[p[i]+1]=1;
    }
   dis=0;
   i=k=r[1]=vv[1]=bv[1]=1;
   p[0]=0;
   p[mres+1]=n;
   q[mres+1]=1;
   for(i1=0;i1<=mres;i1++)
    { for(i2=p[i1]+2;i2<=p[i1+1]+1;i2++)
        { j=y[i2]+1;
 NC:      i5=0;
        if(i2==p[i1+1]+1)
         { temp1=q[i1+1];
           temp=temp1;
           check=dis+d[r[i2-1]][temp1];
           if(check>=min)
            goto JO;
           if(i2==n+1)
            { min=check;
              for(i3=1;i3<=n;i3++)
             str[i3]=r[i3];
              goto JO;
            }
           for(i3=1;i3<=n;i3++)
            { if(temp1==i3)
              for(i4=1;i4<=n;i4++)
             { if(c[i3][i4]==0)
                 goto ACC;
               if(bv[c[i3][i4]]==0)
                 goto JO;
             }
            }
           ACC:
           i5=1;
           goto BOUND;
         }
        temp=r[i2];
        temp1=x[i][j];
        check=dis+d[r[i2-1]][temp1];
        if(check>=min)
 JO:       { i2--;
           if(i2<=p[i1]+1)
            i1--;
           if(i2<=1)
            goto STOP;
```

```
          dis-=d[r[i2-1]][r[i2]];
          bv[r[i2]]=0;
          vv[r[i2]]=0;
          if(pv[i2]==1)
           { vv[r[i2]]=1;
             goto JO;
           }
          j=y[i2]+1;
          i=r[i2-1];
          goto NC;
        }
       if(vv[temp1]==1)
        goto JB;
       for(i3=1;i3<=n;i3++)
        { if(temp1==i3)
          for(i4=1;i4<=n;i4++)
           { if(c[i3][i4]==0)
            goto BOUND;
             if(bv[c[i3][i4]]==0)
JB:           { j=j+1;
              if(j>=n || i5==1)
               goto JO;
              goto NC;
             }
           }
        }

BOUND:      bound=0;
       for(i3=1;i3<=n;i3++)
        { if(bv[i3]==0)
           { for(i4=1;i4<6;i4++)
            { elt=x[i3][i4];
              if((elt!=temp1)&&((bv[elt]==0)||(elt==1)))
               { bound+=d[i3][elt];
                 goto next_row;
               }
            }
             elt=x[i3][6];
             bound+=d[i3][elt];
           }
          next_row:
        }
       bound+=check;
       if(bound>=min)
        goto JB;
       r[i2]=temp1;
       if(temp!=0)
        y[i2+1]=0;
       y[i2]=j;
       vv[r[i2]]=1;
       bv[r[i2]]=1;
       dis=check;
       i=r[i2];
       }
  · }

 STOP:
      gettime(&t);
      t2= 3600*t.ti_hour + 60*t.ti_min + t.ti_sec + 0.01*t.ti_hund;
      min+=bias;

      printf("\n  N=%d SEED=%d SOL=%d TIME=%.2f",n,seed,min,t2-t1);

 OUT:
}

//  REMOVING BIAS
void bias_removal()
  { int i,j,rmin[41],cmin[41];
    for(i=1;i<=n;i++)
     { rmin[i]=9999;
       for(j=1;j<=n;j++)
      if(d[i][j]<rmin[i])
       rmin[i]=d[i][j];
     }
    for(i=1;i<=n;i++)
     for(j=1;j<=n;j++)
      d[i][j]-=rmin[i];
    for(j=1;j<=n;j++)
     { cmin[j]=9999;
       for(i=1;i<=n;i++)
      if(d[i][j]<cmin[j])
       cmin[j]=d[i][j];
     }
    for(j=1;j<=n;j++)
     for(i=1;i<=n;i++)
      d[i][j]-=cmin[j];
    bias=0;
    for(i=1;i<=n;i++)
     bias+=rmin[i]+cmin[i];
  }
```

```
// PROGRAM 4.2
// TSP WITH FIXED POSITION & PRECEDENCE CONSTRAINTS
// USING SEQUENTIAL CONSTRUCTIVE SAMPLING APPROACH

#include<stdio.h>
#include<stdlib.h>
#include<dos.h>

void main()
 {
   struct time t;
   int n,i,j,k,l,i1,i2,i3,i4,i5,dis,min,idx1,idx2,seed,lex,temp;
   int elt,bound,c1[1601],vv[41],v[41][41],d[41][41],x[41][41],bv[41];
   int str[41],r[41],y[41],temp1,row[1601],col[1601], p[15],q[15];
   int mres,nres,pres,pop,flag,value1,value2,c[41][41],a[15],b[15];
   float t1,t2,rnd,py[41],sum;
   printf(" ENTER THE No. OF CITIES, A SEED AND No. OF SMAPLES : ");
    scanf("%d%d%d",&n,&seed,&pop);
   printf(" ENTER No.OF FIXED POSITION AND PRECEDENCE CONSTRAINTS:");
     scanf("%d%d",&mres,&nres);
    printf(" ENTER THE PRESCRIBED POSITIONS : ");
    for(i=1;i<=mres;i++)
     scanf("%d",&p[i]);
    printf(" ENTER THE CORRESPONDING NODES : ");
    for(i=1;i<=mres;i++)
     scanf("%d",&q[i]);
    printf(" ENTER THE PRECEDENCE CONSTRAINTS IN THE FORM A < B : ");
    for(i=1;i<=nres;i++)
     scanf("%d%d",&a[i],&b[i]);

//  CREATING THE PREDECESSOR TABLE
    for(i=1;i<=nres;i++)
     for(j=1;j<=nres;j++)
      if(a[i]==b[j])
       { nres++;
       a[nres]=a[j];
       b[nres]=b[i];
       }

    for(i=1;i<=nres;i++)
     if(a[i]==b[i])
      { printf("\n Solutions are not feasible\n");
      goto NF;
      }

    pres=nres;

//  ARRANGE THE NODES THAT ARE INVOLVED IN THE PRECEDENCE RESTRICTIONS
    for(i=1;i<=n;i++)
     for(j=1;j<=n;j++)
      c[i][j]=0;
    for(i=1;i<=n;i++)
     { i1=0;
       for(j=1;j<=n;j++)
      vv[j]=0;
       for(j=1;j<=nres;j++)
      if(i==b[j] && vv[a[j]]==0)
```

```
         { i1++;
           c[i][i1]=a[j];
           vv[a[j]]=1;
         }
       }

//   RANDOM MATRIX GENERATION
     srand((unsigned)seed);
     for(i=1;i<=n;i++)
      for(j=1;j<=n;j++)
       d[i][j]=(rand()%100)+1;
     for(i=1;i<=n;i++)
      d[i][i]=999;

// MAKE THE DISTANCE BETWEEN b & a TO LARGE ENOUGH
     for(i=1;i<=pres;i++)
      { d[b[i]][a[i]]=999;
        d[1][b[i]]=999;
        d[a[i]][1]=999;
      }

//   MAKE THE DISTANCE BETWEEN TWO POSSIBLE NODES TO LARGE ENOUGH
     p[0]=1;
     q[0]=1;
     for(i=0;i<mres;i++)
      for(j=i+1;j<=mres;j++)
       if(p[j]-p[i]>=2)
        { d[q[i]][q[j]]=999;
        if(q[j]-q[i]==n-1)
         goto DONOT;
        d[q[j]][q[i]]=999;
        DONOT:
        }

     gettime(&t);
     t1= 3600*t.ti_hour + 60*t.ti_min + t.ti_sec + 0.01*t.ti_hund;

//   ALPHABET TABLE
     for(i=1;i<=n;i++)
      { for(j=1;j<=n;j++)
       vv[j]=0;
        for(j=1;j<=n;j++)
       { min=999;
         for(k=1;k<=n;k++)
          if((vv[k]==0)&&(d[i][k]<min))
           { idx1=k;
             min=d[i][k];
           }
          x[i][j]=idx1;
          vv[idx1]=1;
       }
      }

// ALPHABET TABLE FOR BOUND CALCULATION
    for(i=1;i<=n;i++)
     for(j=1;j<=n;j++)
      v[i][j]=0;
      for(i=1;i<=n*n;i++)
      { min=999;
        for(j=1;j<=n;j++)
         for(k=1;k<=n;k++)
          if((v[j][k]==0)&&(d[j][k]<min))
          { idx1=j;
            idx2=k;
            min=d[j][k];
          }
        row[i]=idx1;
        col[i]=idx2;
        v[idx1][idx2]=1;
      }
   c1[0]=0;
   for(i=1;i<=n*n;i++)
    { elt=d[row[i]][col[i]];
      if(elt<999)
       c1[i]=c1[i-1]+elt;
    }
```

```
// MAIN PROGRAM
      min=999;
      for(i=1;i<=pop;i++)
       { for(j=1;j<=n;j++)
         { vv[j]=0;
           str[j]=0;
           bv[j]=0;
         }
       for(i1=1;i1<=mres;i1++)
         vv[q[i1]]=1;
       dis=0;
       str[1]=vv[1]=bv[1]=1;
       for(j=2;j<=n;j++)
         { temp=str[j-1];
           flag=0;
           for(i1=1;i1<=mres;i1++)
            if(j==p[i1]+1)
             { elt=q[i1];
             dis+=d[temp][elt];
             if(dis>=min)
              goto OUT;
             for(i3=1;i3<=n;i3++)
              { flag=1;
                if(c[elt][i3]==0)
                 goto BND;
                if(vv[c[elt][i3]]==0)
                 { j=j-2;
                   i4=0;
                   flag=0;
                   goto JO;
                 }
              }
             }
       JO:  l=0;
           for(k=1;k<=n;k++)
            y[k]=0;
           i5=n-j+1;
```

```
          for(k=1;k<=n;k++)
           { elt=x[temp][k];
             if((vv[elt]==0) && (d[temp][elt]<999))
            { for(i2=1;i2<=n;i2++)
               { if(c[elt][i2]==0)
                  goto NEXT1;
                 if(vv[c[elt][i2]]==0)
                  goto NEXT2;
               }
            }
             goto NEXT2;
     NEXT1:     l=l+1;
             y[l]=elt;
             if(l>i5/10.0+1)
            goto NEXT3;
             NEXT2:
           }
     NEXT3: py[0]=0;
          sum=l*(l+1)/2.0;
          for(k=1;k<=l;k++)
           py[k]=py[k-1]+ (l-k+1)/sum;
          i4=0;
      REPT: rnd=(rand()%100)*0.01;
          i4++;
          if(i4>l)
           goto OUT;
          for(k=1;k<=l;k++)
           if(rnd>=py[k-1] && rnd<py[k])
            { elt=y[k];
            goto EXIT;
            }
      EXIT: dis+=d[temp][elt];
          if(dis>=min)
           goto REPT;

// BOUND CALCULATION
      BND:  bound=0;
          for(i1=1;i1<=10;i1++)
           { if(bv[row[i1]]==0)
            { if(bv[col[i1]]==0 || col[i1]==1)
               bound=c1[j+i1-1]-c1[i1-1];
              goto next_row;
            }
           }
          bound=c1[j+10]-c1[10];
          next_row:
          bound+=dis;
          if(bound>=min)
           { if(flag==1)
            { j=j-2;
              i4=0;
              flag=0;
              goto JO;
            }
            else
             goto REPT;
           }
```

```
            str[j]=elt;
            vv[elt]=1;
            bv[elt]=1;
          }
        dis+=d[str[n]][1];
        if(dis<min)
          { min=dis;
            for(j=1;j<=n;j++)
              r[j]=str[j];
          }
        OUT:
        }

// MODIFIED 2-OPT MOVE
r[n+1]=1;
for(j=2;j<n-1;j++)
 for(k=j+2;k<=n;k++)
  { temp=0;
    for(i=1;i<=mres;i++)
      if(j==p[i]+1 || k==p[i]+1)
       temp=1;
    for(i=1;i<=pres;i++)
     if(r[j]==a[i] && r[k]==b[i])
      temp=1;
     if(temp==0)
{value1=d[r[j-1]][r[j]]+d[r[j]][r[j+1]]+d[r[k-1]][r[k]]+d[r[k]][r[k+1]];
 value2=d[r[j-1]][r[k]]+d[r[k]][r[j+1]]+d[r[k-1]][r[j]]+d[r[j]][r[k+1]];
 if(value2<value1)
  { temp=r[k];
    r[k]=r[j];
    r[j]=temp;
  }
}
  }

      min=0;
      for(j=1;j<=n;j++)
       min+=d[r[j]][r[j+1]];

       gettime(&t);
       t2= 3600*t.ti_hour + 60*t.ti_min + t.ti_sec + 0.01*t.ti_hund;

       printf("\n  N=%d SEED=%d SOL=%d TIME=%.2f",n,seed,min,t2-t1);

NF:
}
```

```
// PROGRAM 4.3
// TSP WITH FIXED POSITION & PRECEDENCE CONSTRAINTS
// USING HYBRID GENETIC ALGORITHM

#include<stdio.h>
#include<stdlib.h>
#include<dos.h>

int n,mres,nres,pop,dis,bias,p[15],q[15],a[15],b[15],idx[15];
int c[36][36],chr[601][36],d[36][36];

void bias_removal();
void create();
void objective(int);
void repro_duction();
void cross_over(float);
void mutation(float);
void seq_search();

void main()
  {
    struct time t;
    float pcr,pmt,t1,t2;
    int i,j,i1,min,seed,gen,vv[36],str[36];

    printf(" ENTER THE No. OF CITIES, A SEED AND POPULATION SIZE : ");
     scanf("%d%d%d",&n,&seed,&pop);
    printf(" ENTER THE PROB. OF CROSSOVER AND MUTATION : ");
     scanf("%f%f",&pcr,&pmt);
    printf(" ENTER No.OF FIXED POSITION AND PRECEDENCE CONSTRAINTS:");
     scanf("%d%d",&mres,&nres);
    printf(" ENTER THE PRESCRIBED POSITIONS : ");
    for(i=1;i<=mres;i++)
     scanf("%d",&p[i]);
    printf(" ENTER THE CORRESPONDING NODES : ");
    for(i=1;i<=mres;i++)
     scanf("%d",&q[i]);
    printf(" ENTER THE PRECEDENCE CONSTRAINTS IN THE FORM A < B : ");
    for(i=1;i<=nres;i++)
     scanf("%d%d",&a[i],&b[i]);

    for(i=1;i<=nres;i++)
     for(j=1;j<=nres;j++)
      if(a[i]==b[j])
       { nres++;
       a[nres]=a[j];
       b[nres]=b[i];
       }

//  ARRANGE THE NODES THAT ARE INVOLVED IN THE PRECEDENCE RESTRICTIONS
    for(i=1;i<=n;i++)
     for(j=1;j<=n;j++)
      c[i][j]=0;
    for(i=1;i<=n;i++)
     { i1=0;
       for(j=1;j<=nres;j++)
      if(i==b[j])
```

```
        { i1++;
          c[i][i1]=a[j];
        }
       else
        c[i][j]=0;
       }
     i1=0;
     for(i=1;i<=n;i++)
      { if(c[i][1]!=0)
       { i1++;
         b[i1]=i;
         for(j=1;j<=nres;j++)
          if(c[i][j]>0)
           { c[i1][j]=c[i][j];
             idx[i1]=j;
           }
          else
           goto NEXT_RES;
       }
        NEXT_RES:;
      }
     nres=i1;

//   RANDOM MATRIX GENERATION
     srand((unsigned)seed);
     for(i=1;i<=n;i++)
      for(j=1;j<=n;j++)
       d[i][j]=(rand()%100)+1;
     for(i=1;i<=n;i++)
      d[i][i]=999;

// MAKE THE DISTANCE BETWEEN b & a TO LARGE ENOUGH
     for(i=1;i<=nres;i++)
      { d[b[i]][a[i]]=999;
        d[1][b[i]]=999;
        d[a[i]][1]=999;
      }

//   MAKE THE DISTANCE BETWEEN TWO POSSIBLE NODES TO LARGE ENOUGH
     p[0]=1;
     q[0]=1;
     for(i=0;i<mres;i++)
      for(j=i+1;j<=mres;j++)
       if(p[j]-p[i]>=2)
        { d[q[i]][q[j]]=999;
        if(q[j]-q[i]==n-1)
         goto DONOT;
        d[q[j]][q[i]]=999;
        DONOT:
        }

     gettime(&t);
     t1= 3600*t.ti_hour + 60*t.ti_min + t.ti_sec + 0.01*t.ti_hund;

//   CALLING BIAS REMOVAL FUNCTION
     bias_removal();
```

```
//  TERMINATING CRITERION
    min=d[1][1];
    create();
    for(i=1;i<=pop;i++)
     { objective(i);
       if(dis<min)
      { min=dis;
        for(j=1;j<=n;j++)
         str[j]=chr[i][j];
      }
     }
    for(gen=1;gen<=4*n;gen++)
     { repro_duction();
       cross_over(pcr);
       mutation(pmt);
       seq_search();
       for(i=1;i<=pop;i++)
      { objective(i);
        if(dis<min)
         { min=dis;
           for(j=1;j<=n;j++)
            str[j]=chr[i][j];
         }
      }
     }
    min+=bias;
    gettime(&t);
    t2= 3600*t.ti_hour + 60*t.ti_min + t.ti_sec + 0.01*t.ti_hund;
    printf("\nN=%d Best Solution=%d Time=%.2f",n,min,t2-t1);
  }

//  REMOVING BIAS
void bias_removal()
  { int i,j,rmin[41],cmin[41];
    for(i=1;i<=n;i++)
     { rmin[i]=9999;
       for(j=1;j<=n;j++)
      if(d[i][j]<rmin[i])
       rmin[i]=d[i][j];
     }
    for(i=1;i<=n;i++)
     for(j=1;j<=n;j++)
      d[i][j]-=rmin[i];
    for(j=1;j<=n;j++)
     { cmin[j]=9999;
       for(i=1;i<=n;i++)
      if(d[i][j]<cmin[j])
       cmin[j]=d[i][j];
     }
    for(j=1;j<=n;j++)
     for(i=1;i<=n;i++)
      d[i][j]-=cmin[j];
    bias=0;
    for(i=1;i<=n;i++)
     bias+=rmin[i]+cmin[i];
  }
```

```
// INITIALIZE THE POPULATION
void create()
 { int m,i,j,k,i1,elt,temp,count,vv[41];
   for(i=1;i<=pop;i++)
    { for(j=1;j<=n;j++)
        vv[j]=0;
      for(j=1;j<=mres;j++)
        vv[q[j]]=1;
      chr[i][1]=1;
      vv[1]=1;
      m=n-mres-1;
      for(j=2;j<=n;j++)
       { for(k=1;k<=mres;k++)
        if(j==p[k]+1)
         { chr[i][j]=q[k];
           goto next2;
         }
  RPT:        elt=(rand()%m)+1;
        count=0;
        for(k=2;k<=n;k++)
         if(vv[k]==0)
          { count++;
            if(count==elt)
             { temp=k;
            goto next1;
             }
          }
 next1:    for(i1=1;i1<=n;i1++)
          { if(c[temp][i1]==0)
             goto ACCEPT;
            if(vv[c[temp][i1]]==0)
             goto RPT;
          }
 ACCEPT: chr[i][j]=temp;
        vv[chr[i][j]]=1;
        m--;
        next2:
        }
    }
 }

// CALCULATE THE VALUE OF THE OBJECTIVE FUNCTION
void objective(int i)
  { int j;
    dis=0;
    for(j=1;j<n;j++)
     dis+=d[chr[i][j]][chr[i][j+1]];
    dis+=d[chr[i][n]][1];
  }

// STOCHASTIC REMAINDER SELECTION METHOD
void repro_duction()
 {  float fit[551],random,avg,expect,sum=0,frac[551];
    int i,j,k,assign,select,choice[551];
    for(i=1;i<=pop;i++)
     { objective(i);
       fit[i]=1.0/(float)dis;
      sum+=fit[i];
     }
    avg=sum/pop;
    k=0;
    for(i=1;i<=pop;i++)
     { expect=fit[i]/avg;
       assign=expect;
       frac[i]=expect-assign;
       while(assign>0)
      { k=k+1;
        assign-=1;
        choice[k]=i;
      }
     }
    i=0;
    while(k<pop)
     { i=i+1;
       if(i>pop)
      i=1;
       if(frac[i]>0.0)
      { random=(rand()%1000)*0.001;
        if(frac[i]>random)
          { k=k+1;
            choice[k]=i;
            frac[i]-=1.0;
          }
      }
     }
    for(i=1;i<=pop;i++)
     { select=choice[i];
       for(j=1;j<=n;j++)
      chr[i][j]=chr[select][j];
     }
}
```

// C1-CROSSOVER OPERATION

```
 void cross_over(float pc)
  { int i,j,k,l,m,cross,min,cost,v1[41],v2[41],x,y,nwchr[21][41];
    float random;
    for(i=1;i<pop;i++)
     { random=(rand()%100)*0.01;
       if(random<pc)
      { for(m=1;m<=20;m=m+2)
         { for(j=1;j<=n;j++)
            { v1[j]=0;
            v2[j]=0;
            }
           cross=(rand()%(n-2))+2;
           for(j=1;j<cross;j++)
            { x=chr[i][j];
            y=chr[i+1][j];
            nwchr[m][j]=x;
            nwchr[m+1][j]=y;
            v1[x]=1;
            v2[y]=1;
            }
           k=0;
```

```
          for(j=cross;j<=n;j++)
           { for(l=1;l<=mres;l++)
            if(j==p[l]+1)
             { x=q[l];
               goto next1;
             }
      read1:      k=k+1;
           x=chr[i+1][k];
           if(v1[x]==1)
            goto read1;
      next1:      v1[x]=1;
           nwchr[m][j]=x;
           }
          k=0;
          for(j=cross;j<=n;j++)
           { for(l=1;l<=mres;l++)
            if(j==p[l]+1)
             { y=q[l];
               goto next2;
             }
      read2:      k=k+1;
           y=chr[i][k];
           if(v2[y]==1)
            goto read2;
      next2:      v2[y]=1;
           nwchr[m+1][j]=y;
           }
        }
       min=999;
       for(m=1;m<=20;m++)
        { cost=0;
          for(j=1;j<n;j++)
           cost+=d[nwchr[m][j]][nwchr[m][j+1]];
          cost+=d[nwchr[m][n]][1];
          if(cost<min)
           { min=cost;
           for(j=1;j<=n;j++)
            chr[i][j]=nwchr[m][j];
           }
        }
     }
    }
   }
```

// BITWISE MUTATION OPERATION

```
  void mutation(float pm)
   { int temp,i,i1,j,rv[41];
     float random;
     for(i=1;i<=pop;i++)
      for(j=2;j<=n;j++)
       { for(i1=1;i1<=nres;i1++)
        rv[b[i1]]=1;
       if(j<n)
        { random=(rand()%100)*0.01;
          if(random<pm)
           { for(i1=1;i1<=mres;i1++)
            if(j==p[i1]+1 || j+1==p[i1]+1)
```

```
                goto next;
                if((rv[chr[i][j]]==1)||(rv[chr[i][j+1]]==1))
               goto next;
                temp=chr[i][j];
                chr[i][j]=chr[i][j+1];
                chr[i][j+1]=temp;
              }
           }
        next:
         }
    }

// SEQUENTIAL CONSTRUCTIVE SEARCH APPROACH
void seq_search()
  { int i,j,k,l,m,min,x[3],cost[3],v[41],r[3][41],nwchr[41];
    int rv[15][15],i1,i2;
    for(i=1;i<pop;i++)
     { for(j=1;j<=n;j++)
      { r[1][j]=chr[i][j];
        r[2][j]=chr[i+1][j];
        v[j]=0;
      }
       for(i1=1;i1<=mres;i1++)
      v[q[i1]]=1;
       for(i1=1;i1<=nres;i1++)
      for(i2=1;i2<=idx[i1];i2++)
       rv[i1][i2]=0;
       nwchr[1]=1;
       v[1]=1;
       for(k=2;k<=n;k++)
      { for(i1=1;i1<=mres;i1++)
         if(k==p[i1]+1)
          { nwchr[k]=q[i1];
            goto FIXED;
          }
        for(l=1;l<=2;l++)
         { for(j=1;j<=n;j++)
            { if(r[l][j]==nwchr[k-1])
             { for(m=1;m<=n;m++)
                { if(j+m>n)
                   { cost[l]=999;
                   x[l]=r[l][j];
                   goto next;
                   }
                  if(v[r[l][j+m]]==0)
                   { cost[l]=d[r[l][j]][r[l][j+m]];
                   x[l]=r[l][j+m];
                   goto next;
                   }
                }
             }
           }
          next:
        }
        if((cost[1]==999) && (cost[2]==999))
         for(j=1;j<=n;j++)
          if(v[j]==0)
          { nwchr[k]=j;
            goto CHK;
          }
        nwchr[k]=x[1];
        if(cost[2]<cost[1])
          nwchr[k]=x[2];
   CHK: for(i1=1;i1<=nres;i1++)
          for(i2=1;i2<=idx[i1];i2++)
           if((nwchr[k]==b[i1])&&(rv[i1][i2]==0))
    RPT:    { for(j=1;j<=n;j++)
             if(v[r[1][j]]==0)
              { nwchr[k]=r[1][j];
                goto ACCEPT;
              }
            }
     ACCEPT:
       if(v[nwchr[k]]==1)
         goto RPT;
       for(i1=1;i1<=nres;i1++)
        for(i2=1;i2<=idx[i1];i2++)
         if(nwchr[k]==c[i1][i2])
          rv[i1][i2]=1;
 FIXED: v[nwchr[k]]=1;
    }
   for(k=1;k<=n;k++)
    chr[i][k]=nwchr[k];
   }
}
```

```
//  PROGRAM 5.1
//  TSP WITH BACKHAULS USING LEXI-SEARCH
//  PATH APPROACH

#include<stdio.h>
#include<stdlib.h>
#include<dos.h>

int n,bias,d[41][41];
void bias_removal();

void main()
  {

    struct time t;
    float t1,t2;
    int i,j,k,l,i1,i2,nl,nbl,seed,temp,index,dis,temp1,check,count;
    int y[41],p[41],str[41],vv[41],x[41][41],elt,bound;
    int min,arc,z[41][41],p1[41],q[41];

    printf(" ENTER THE No. OF CITIES, A SEED AND BACKHAUL NODES : ");
    scanf("%d%d%d",&n,&seed,&nbl);
    nl=n-1-nbl; // No. of linehaul //

//  RANDOM MATRIX GENERATION
    srand((unsigned)seed);
    for(i=1;i<=n;i++)
     for(j=1;j<=n;j++)
      d[i][j]=(rand()%100)+1;
    for(i=1;i<=n;i++)
     d[i][i]=999;

//  CONDITIONS
    for(j=nl+2;j<=n;j++)
     d[1][j]=999;
    for(i=2;i<=nl+1;i++)
     d[i][1]=999;
    for(i=nl+2;i<=n;i++)
     for(j=2;j<=nl+1;j++)
      d[i][j]=999;

    gettime(&t);
    t1= 3600*t.ti_hour + 60*t.ti_min + t.ti_sec + 0.01*t.ti_hund;

//  CALLING BIAS REMOVAL FUNCTION
    bias_removal();

//  ALPHABET TABLE FOR BOUND CALCULATION
    for(i=1;i<=n;i++)
     { for(j=1;j<=n;j++)
      vv[j]=0;
       for(j=1;j<=n;j++)
      { min=999;
        for(k=1;k<=n;k++)
         if((vv[k]==0)&&(d[i][k]<min))
          { index=k;
            min=d[i][k];
          }
          z[i][j]=index;
          vv[index]=1;
      }
     }

    for(i=2;i<=nl+1;i++)
     for(j=nl+2;j<=n;j++)
      d[i][j]+=100;
     for(i=nl+2;i<=n;i++)
      d[i][1]+=100;

//  THE ALPHABET TABLE
    for(i=1;i<=n;i++)
     { for(j=1;j<=n;j++)
      vv[j]=0;
       for(j=1;j<=n;j++)
      { min=9999;
        for(k=1;k<=n;k++)
         if((vv[k]==0)&&(d[i][k]<min))
          { index=k;
            min=d[i][k];
          }
         x[i][j]=index;
         vv[index]=1;
      }
     }

    for(i=2;i<=nl+1;i++)
     for(j=nl+2;j<=n;j++)
      d[i][j]-=100;
     for(i=nl+2;i<=n;i++)
      d[i][1]-=100;

//   MAIN PROGRAM STARTS HERE
     min=900;
     for(j=2;j<=n;j++)
      { vv[j]=0;
      y[j]=0;
      p[j]=0;
      }
     dis=0;
     k=p[1]=i=vv[1]=1;
     y[nl+2]=nl-1;
 GS: k=k+1;
     j=y[k]+1;
 NC: temp=p[k];
     temp1=x[i][j];
     arc=d[i][temp1];
     check=dis+arc;
     if((k<=nl+1 && temp1>nl+1) || (arc>100) || (check>=min))
      goto JO;
     if(vv[temp1]==1)
 JB:  { j=j+1;
      goto NC;
      }
```

```
//    BOUND CALCULATION
      bound=0;
      index=n/2;
      for(i1=1;i1<=n;i1++)
       if(vv[i1]==0)
        { for(i2=1;i2<index;i2++)
         { elt=z[i1][i2];
           if((elt!=temp1)&&((vv[elt]==0)||(elt==1)))
            { bound+=d[i1][elt];
              goto next_row;
            }
         }
        elt=z[i1][index];
        bound+=d[i1][elt];
        next_row:
        }
      bound+=check;
      if(bound>=min)
       goto JB;
      p[k]=temp1;
      if(temp!=0)
       { y[k+1]=0;
       if(k==n1+1)
        y[k+1]=n1-1;
       }
      y[k]=j;
      vv[temp1]=1;
      dis=check;
      i=temp1;
      if(k==n)
       { dis+=d[i][1];
       if(dis>=min)
        goto PREV;
       for(j=1;j<=n;j++)
        str[j]=p[j];
       min=dis;
 PREV:  dis-=d[i][1];
       dis-=d[p[k-1]][i];
       vv[i]=0;
 JO:     k=k-1;
       if(k<=1)
        goto STOP;
       vv[p[k]]=0;
       dis-=d[p[k-1]][p[k]];
       j=y[k]+1;
       i=p[k-1];
       goto NC;
       }
      goto GS;

  STOP:
       gettime(&t);
       t2= 3600*t.ti_hour + 60*t.ti_min + t.ti_sec + 0.01*t.ti_hund;
       min+=bias;

  printf("\nN=%d |B|=%d SEED=%d SOL=%d TIME=%.2f",n,nbl,seed,min,t2-t1);
}
```

```
//  REMOVING BIAS
void bias_removal()
  { int i,j,rmin[41],cmin[41];
    for(i=1;i<=n;i++)
     { rmin[i]=9999;
       for(j=1;j<=n;j++)
      if(d[i][j]<rmin[i])
       rmin[i]=d[i][j];
     }
    for(i=1;i<=n;i++)
     for(j=1;j<=n;j++)
      d[i][j]-=rmin[i];
    for(j=1;j<=n;j++)
     { cmin[j]=9999;
       for(i=1;i<=n;i++)
      if(d[i][j]<cmin[j])
       cmin[j]=d[i][j];
     }
    for(j=1;j<=n;j++)
     for(i=1;i<=n;i++)
      d[i][j]-=cmin[j];
    bias=0;
    for(i=1;i<=n;i++)
     bias+=rmin[i]+cmin[i];
  }
```

```
// PROGRAM 5.2
// TSP WITH BACKHAULS
// USING SEQUENTIAL CONSTRUCTIVE SAMPLING APPROACH

#include<stdio.h>
#include<stdlib.h>
#include<dos.h>

void main()
 {
   struct time t;
   int nl,n,pop,i,j,k,l,il,i2,i3,dis,min,idx1,idx2,seed,temp,elt,bound;
   int c[1601],vv[41],v[41][41],d[41][41],x[41][41],str[41],p[41],y[41];
   int value1,value2,row[1601],col[1601],nbl,templ,st;
   int lm[41][41],bm[41][41];
   float t1,t2,sum,rnd,py[41];

    printf(" ENTER THE No. OF CITIES AND BACKHAUL NODES RESPECTIVELY:");
    scanf("%d%d",&n,&nbl);
    printf(" ENTER A SEED AND SAMPLE SIZE RESPECTIVELY:");
    scanf("%d%d",&seed,&pop);

    nl=n-1-nbl; // No. of linehaul //

//  RANDOM MATRIX GENERATION
    srand((unsigned)seed);
    for(i=1;i<=n;i++)
     for(j=1;j<=n;j++)
      d[i][j]=(rand()%100)+1;
    for(i=1;i<=n;i++)
     d[i][i]=999;

//  CONDITIONS
    for(j=nl+2;j<=n;j++)
     d[1][j]=999;
    for(i=2;i<=nl+1;i++)
     d[i][1]=999;
    for(i=nl+2;i<=n;i++)
     for(j=2;j<=nl+1;j++)
      d[i][j]=999;

    gettime(&t);
    t1= 3600*t.ti_hour + 60*t.ti_min + t.ti_sec + 0.01*t.ti_hund;

    for(i=2;i<=nl+1;i++)
     for(j=nl+2;j<=n;j++)
      d[i][j]+=100;

//  THE ALPHABET TABLE
    for(i=1;i<=n;i++)
     { for(j=1;j<=n;j++)
      vv[j]=0;
       for(j=1;j<=n;j++)
      { min=9999;
        for(k=1;k<=n;k++)
         if((vv[k]==0)&&(d[i][k]<min))
          { idx1=k;
```

```
            min=d[i][k];
          }
        x[i][j]=idx1;
        vv[idx1]=1;
     }
    }

    for(i=2;i<=n1+1;i++)
     for(j=n1+2;j<=n;j++)
      d[i][j]-=100;

// ALPHABET TABLE FOR BOUND CALCULATION
   for(i=1;i<=n;i++)
    for(j=1;j<=n;j++)
     v[i][j]=0;
      for(i=1;i<=n*n;i++)
      { min=9999;
        for(j=1;j<=n;j++)
         for(k=1;k<=n;k++)
          if((v[j][k]==0)&&(d[j][k]<min))
          { idx1=j;
            idx2=k;
            min=d[j][k];
          }
         row[i]=idx1;
         col[i]=idx2;
         v[idx1][idx2]=1;
      }
   c[0]=0;
   for(i=1;i<=n*n;i++)
    { elt=d[row[i]][col[i]];
      if(elt<999)
       c[i]=c[i-1]+elt;
    }

// MAIN PROGRAM
      min=9999;
      for(i=1;i<=n;i++)
       p[i]=0;
      for(i=1;i<=pop;i++)
       { for(j=1;j<=n;j++)
        { vv[j]=0;
          str[j]=0;
        }
       dis=0;
       str[1]=vv[1]=1;
       for(j=2;j<=n;j++)
        { l=0;
          st=1;
          if(j==n1+2)
           st=n1;
          for(k=1;k<=n;k++)
           y[k]=0;
          i1=n-j+1;
          temp=str[j-1];
          for(k=st;k<=n;k++)
           { elt=x[temp][k];
          if((vv[elt]==0) && (d[temp][elt]<999))
         { if(j<=nl+1 && elt>nl+1)
             goto NEXT;
            else
             { l=l+1;
               y[l]=elt;
               if(l>il/5.0+1)
                goto NEXT;
             }
         }
        }
   NEXT:
      py[0]=0;
      sum=l*(l+1)/2.0;
      for(k=1;k<=l;k++)
       py[k]=py[k-1]+ (l-k+1)/sum;
      i3=0;
  REPT: rnd=(rand()%100)*0.01;
      i3++;
      for(k=1;k<=l;k++)
       if(rnd>=py[k-1] && rnd<py[k])
        { str[j]=y[k];
        goto EXIT;
        }
      EXIT:
      dis+=d[temp][str[j]];
      if(dis>=min)
       goto OUT;
      temp1=str[j];

// BOUND CALCULATION
      bound=0;
      for(il=1;il<=10;il++)
       { if(vv[row[il]]==0)
        { if(vv[col[il]]==0 || col[il]==1)
            bound=c[j+il-1]-c[il-1];
          goto next_row;
        }
       }
      bound=c[j+10]-c[10];
      next_row:
      bound+=dis;
      if(bound>=min && i3<l)
       goto REPT;
      vv[str[j]]=1;
   }
   dis+=d[str[n]][1];
   if(dis<min)
    { min=dis;
      for(j=1;j<=n;j++)
       p[j]=str[j];
    }
  OUT:
   }
```

```
// MODIFIED 2-OPT MOVE
p[n+1]=1;
for(j=2;j<n-1;j++)
 for(k=j+2;k<=n;k++)
  if((j<nl && k<=nl+1) || (j>nl+1 && k>nl+1))
{value1=d[p[j-1]][p[j]]+d[p[j]][p[j+1]]+d[p[k-1]][p[k]]+d[p[k]][p[k+1]];
 value2=d[p[j-1]][p[k]]+d[p[k]][p[j+1]]+d[p[k-1]][p[j]]+d[p[j]][p[k+1]];
 if(value2<value1)
  { temp=p[k];
    p[k]=p[j];
    p[j]=temp;
  }
}

   min=0;
   for(j=1;j<=n;j++)
   min+=d[p[j]][p[j+1]];

   gettime(&t);
   t2= 3600*t.ti_hour + 60*t.ti_min + t.ti_sec + 0.01*t.ti_hund;
   printf("\nN=%d |B|=%d Best Solution=%d Time=%.2f ",n,nbl,min,t2-t1);
  }
```

```
//  PROGRAM 5.3
//  TSP WITH BACKHAULS USING HYBRID GENETIC ALGORITHM

#include<stdio.h>
#include<stdlib.h>
#include<dos.h>

int n,nl,nbl,pop,dis,bias,chr[326][41],d[41][41];

void bias_removal();
void create();
void objective(int);
void repro_duction();
void cross_over(float);
void mutation(float);
void seq_search();

void main()
  {
    struct time t;
    float pcr,pmt,t1,t2;
    int i,j,seed,gen,min,vv[41],str[41];

    printf("ENTER THE No.OF CITIES AND BACKHAUL NODES : ");
     scanf("%d%d",&n,&nbl);
    printf("ENTER A SEED AND POPULATION SIZE : ");
     scanf("%d%d",&seed,&pop);
   printf("ENTER PROBABILITY OF CROSSOVER AND MUTATION RESPECTIVELY:");
     scanf("%f%f",&pcr,&pmt);

    nl=n-1-nbl; // No. of linehaul //

//  RANDOM MATRIX GENERATION
    srand((unsigned)seed);
    for(i=1;i<=n;i++)
     for(j=1;j<=n;j++)
      d[i][j]=(rand()%100)+1;
    for(i=1;i<=n;i++)
     d[i][i]=999;

//  CONDITIONS
    for(j=nl+2;j<=n;j++)
     d[1][j]=999;
    for(i=2;i<=nl+1;i++)
     d[i][1]=999;
    for(i=nl+2;i<=n;i++)
     for(j=2;j<=nl+1;j++)
      d[i][j]=999;

    gettime(&t);
    t1= 3600*t.ti_hour + 60*t.ti_min + t.ti_sec + 0.01*t.ti_hund;

//  CALLING BIAS REMOVAL FUNCTION
    bias_removal();

//  TERMINATING CRITERION
    min=d[1][1];
    create();
    for(i=1;i<=pop;i++)
     { objective(i);
       if(dis<min)
      { min=dis;
        for(j=1;j<=n;j++)
         str[j]=chr[i][j];
      }
     }
    for(gen=1;gen<=4*n;gen++)
     { repro_duction();
       cross_over(pcr);
       mutation(pmt);
       seq_search();
       for(i=1;i<=pop;i++)
      { objective(i);
        if(dis<min)
         min=dis;
        for(j=1;j<=n;j++)
         str[j]=chr[i][j];
      }
     }
    min+=bias;
    gettime(&t);
    t2= 3600*t.ti_hour + 60*t.ti_min + t.ti_sec + 0.01*t.ti_hund;
    printf("\nN=%d |B|=%d Best Solution=%d Time=%.2f",n,nbl,min,t2-t1);
  }

//  REMOVING BIAS
void bias_removal()
  { int i,j,rmin[41],cmin[41];
    for(i=1;i<=n;i++)
     { rmin[i]=9999;
       for(j=1;j<=n;j++)
      if(d[i][j]<rmin[i])
       rmin[i]=d[i][j];
     }
    for(i=1;i<=n;i++)
     for(j=1;j<=n;j++)
      d[i][j]-=rmin[i];
    for(j=1;j<=n;j++)
     { cmin[j]=9999;
       for(i=1;i<=n;i++)
      if(d[i][j]<cmin[j])
       cmin[j]=d[i][j];
     }
    for(j=1;j<=n;j++)
     for(i=1;i<=n;i++)
      d[i][j]-=cmin[j];
    bias=0;
    for(i=1;i<=n;i++)
     bias+=rmin[i]+cmin[i];
  }

// INITIALIZE THE POPULATION
void create()
 { int m,i,j,il,elt,count,vv[41];
```

```
   for(i=1;i<=pop;i++)
    { for(j=1;j<=n;j++)
       { vv[j]=0;
       chr[i][j]=0;
       }
     chr[i][1]=1;
     vv[1]=1;
     m=n1;
     for(j=2;j<=n;j++)
      { if(j>n1+1)
       m=n-j+1;
      elt=(rand()%m)+1;
      count=0;
      for(i1=2;i1<=n;i1++)
       if(vv[i1]==0)
        { count++;
          if(count==elt)
           { chr[i][j]=i1;
           goto rxx;
           }
        }
     rxx:
      vv[chr[i][j]]=1;
      m--;
      }
    }
 }

// CALCULATE THE VALUE OF THE OBJECTIVE FUNCTION
 void objective(int i)
  { int j;
    dis=0;
    for(j=1;j<n;j++)
     dis+=d[chr[i][j]][chr[i][j+1]];
    dis+=d[chr[i][n]][1];
  }

// STOCHASTIC REMAINDER SELECTION METHOD
void repro_duction()
 {  float fit[326],random,avg,expect,sum=0,frac[326];
    int i,j,k,assign,select,choice[326];
    for(i=1;i<=pop;i++)
     { fit[i]=0;
       objective(i);
       fit[i]=1.0/(float)(dis+1);
       sum+=fit[i];
     }
    avg=sum/pop;
    k=0;
    for(i=1;i<=pop;i++)
     { expect=fit[i]/avg;
       assign=expect;
       frac[i]=expect-assign;
       while(assign>0)
      { k=k+1;
        assign-=1;
        choice[k]=i;
       }
      }
     i=0;
     while(k<pop)
      { i=i+1;
        if(i>pop)
       i=1;
        if(frac[i]>0.0)
       { random=(rand()%100)*0.01;
          if(frac[i]>random)
           { k=k+1;
              choice[k]=i;
              frac[i]-=1.0;
           }
       }
      }
     for(i=1;i<=pop;i++)
      { select=choice[i];
        for(j=1;j<=n;j++)
       chr[i][j]=chr[select][j];
      }
 }
```

// C1-CROSSOVER OPERATION

```
  void cross_over(float pc)
   { int i,j,k,p,m,i1,i2,min,cost,cross,v1[41],v2[41],x,y,nwchr[21][41];
     float random;
     for(i=1;i<pop;i++)
      { random=(rand()%100)*0.01;
        if(random<pc)
       { for(p=1;p<=20;p=p+2)
          { for(j=1;j<=n;j++)
              { v1[j]=0;
              v2[j]=0;
              }
            for(i1=1;i1<=2;i1++)
             { if(i1<=1)
               { m=n1+1;
                 i2=1;
                 cross=(rand()%(m-1))+1;
               }
              else
               { m=n;
                 i2=n1+2;
                 cross=(rand()%(m-n1-2))+n1+2;
               }
              for(j=i2;j<=cross;j++)
               { x=chr[i][j];
                 y=chr[i+1][j];
                 nwchr[p][j]=x;
                 nwchr[p+1][j]=y;
                 v1[x]=1;
                 v2[y]=1;
               }
              k=0;
              for(j=cross+1;j<=m;j++)
      read1:       { k=k+1;
```

```
                x=chr[i+1][k];
                if(v1[x]==1)
                 goto read1;
                v1[x]=1;
                nwchr[p][j]=x;
               }
             k=0;
             for(j=cross+1;j<=m;j++)
    read2:       { k=k+1;
                y=chr[i][k];
                if(v2[y]==1)
                 goto read2;
                v2[y]=1;
                nwchr[p+1][j]=y;
               }
             }
          }

         min=999;
         for(p=1;p<=20;p++)
          { cost=0;
            for(j=1;j<n;j++)
             cost+=d[nwchr[p][j]][nwchr[p][j+1]];
            cost+=d[nwchr[p][n]][1];
            if(cost<min)
             { min=cost;
             for(j=1;j<=n;j++)
              chr[i][j]=nwchr[p][j];
             }
          }
       }
      }
   }
```

```
// BITWISE MUTATION OPERATION
  void mutation(float pm)
   { int temp,i,j;
     float random;
     for(i=1;i<=pop;i++)
      for(j=2;j<n;j++)
       if(j!=n1+1)
      { random=(rand()%100)*0.01;
        if(random<pm)
         { temp=chr[i][j];
           chr[i][j]=chr[i][j+1];
           chr[i][j+1]=temp;
         }
      }
   }
```

```
// SEQUENTIAL CONSTRUCTIVE SEARCH APPROACH
 void seq_search()
  { int
i,j,k,l,m,p,min,start,end,temp,x[3],cost[3],v[41],nwchr[326][41];
    for(i=1;i<pop;i++)
     { for(j=1;j<=n;j++)
      { nwchr[i][j]=chr[i][j];
```

```
        nwchr[i+1][j]=chr[i+1][j];
        v[j]=0;
      }
       chr[i][1]=v[1]=1;
       start=2;
       end=n1+1;
       for(p=1;p<=2;p++)
      { if(p==2)
          { start=n1+2;
            end=n;
          }
        for(k=start;k<=end;k++)
          { for(l=0;l<=1;l++)
              { cost[l]=999;
              x[l]=1;
              for(j=start-1;j<=end;j++)
                { if(nwchr[i+1][j]==chr[i][k-1])
                    { for(m=1;j+m<=end;m++)
                       if(v[nwchr[i+1][j+m]]==0)
                      { cost[l]=d[chr[i][k-1]][nwchr[i+1][j+m]];
                        x[l]=nwchr[i+1][j+m];
                        goto next;
                      }
                    }
                }
              next:
              }
           if(cost[1]==999 && cost[0]==999)
            { min=999;
              for(j=start;j<=end;j++)
             if(v[j]==0)
              { temp=d[chr[i][k-1]][j];
                if(temp<min)
                 { chr[i][k]=j;
                   min=temp;
                 }
              }
              goto ACCEPT;
            }
        chr[i][k]=x[0];
        if(cost[1]<cost[0])
         chr[i][k]=x[1];
ACCEPT:  v[chr[i][k]]=1;
        }
      }
    }
  }
```

```
// PROGRAM 6.1
// MIN-MAX TRAVELLING SALESMAN PROBLEM USING LEXI-SEARCH
// ADJACENCY APPROACH

#include<stdio.h>
#include<stdlib.h>
#include<dos.h>

void main()
  {
   struct time t;
   float t1,t2;
  int i,j,k,l,i1,i2,i3,n,seed,temp,index,dis,temp1,bias,check,elt,bound;
  int max,min,ass[46],str[46],y[46],p[46],vv[46],d[46][46],dist[46][46];
  int x[46][46],z[46][46],c[46][46];

    printf(" ENTER THE No. OF CITIES AND A SEED : ");
     scanf("%d%d",&n,&seed);

//  RANDOM MATRIX GENERATION
    srand((unsigned)seed);
    for(i=1;i<=n;i++)
     for(j=1;j<=n;j++)
      d[i][j]=(rand()%100)+1;
    for(i=1;i<=n;i++)
     d[i][i]=999;

    for(i=1;i<=n;i++)
     for(j=1;j<=n;j++)
      c[i][j]=d[i][j];

    gettime(&t);
    t1= 3600*t.ti_hour + 60*t.ti_min + t.ti_sec - 0.01*t.ti_hund;

//  ALPHABET TABLE
    for(i=1;i<=n;i++)
     { for(j=1;j<=n;j++)
      vv[j]=0;
       for(j=1;j<=n;j++)
      { min=999;
        for(k=1;k<=n;k++)
         if((vv[k]==0)&&(d[i][k]<min))
          { index=k;
            min=d[i][k];
          }
         z[i][j]=index;
         vv[index]=1;
      }
     }

     for(i=1;i<=n;i++)
      for(j=1;j<n;j++)
       for(k=j+1;k<=n;k++)
        if(d[i][j]>d[i][k])
       { temp=d[i][j];
         d[i][j]=d[i][k];
         d[i][k]=temp;
       }

     for(i=1;i<=n;i++)
      for(j=1;j<=n;j++)
       { dist[j][i]=d[i][j];
       x[j][i]=z[i][j];
       }

//   MAIN PROGRAM STARTS HERE
      min=900;
      for(i=1;i<=n;i++)
       { vv[i]=0;
       y[i]=0;
       p[i]=0;
       }
      dis=j=i=0;
 GS:  j=j+1;
      i=y[j]+1;
 NC:  temp=p[j];
      temp1=x[i][j];
      dis=dist[i][j];
      if(dis>=min)
       goto JO;
      if(vv[temp1]==1)
 JB:   { i=i+1;
       goto NC;
       }

//    CYCLE CHECKING
      if(temp1<j && j!=n)
       { k=temp1;
       for(i1=1;i1<j;i1++)
        { if(p[k]<j)
           k=p[k];
          else
           if(p[k]==j)
            goto JB;
           else
            goto BD;
        }
       }
    BD:

//    BOUND CALCULATION
      bound=0;
      for(i1=j+1;i1<=n;i1++)
       { for(i2=1;i2<n-1;i2++)
        { elt=x[i2][i1];
          if((elt!=temp1) && (vv[elt]==0))
           { if(bound<dist[i2][i1])
              bound=dist[i2][i1];
             goto next_col;
           }
        }
       next_col:
       }
      if(dis>bound)
```

```
        bound=dis;
       if(bound>=min)
        goto JB;
       p[j]=temp1;
       if(temp!=0)
        y[j+1]=0;
       y[j]=i;
       vv[temp1]=1;
       if(j==n)
        {
        max=0;
        for(i1=1;i1<=n;i1++)
         { check=c[i1][p[i1]];
           if(check>max)
            max=check;
         }
        if(max>=min)
         goto PREV;
        for(i1=1;i1<=n;i1++)
         ass[i1]=p[i1];
        min=max;
PREV:         vv[p[j]]=0;
JO:      j=j-1;
        if(j<1)
         goto STOP;
        vv[p[j]]=0;
        i=y[j]+1;
        goto NC;
        }
       goto GS;

 STOP:

        str[1]=1;
        for(i=2;i<=n;i++)
         { k=str[i-1];
         str[i]=ass[k];
         }

        gettime(&t);
        t2= 3600*t.ti_hour + 60*t.ti_min + t.ti_sec + 0.01*t.ti_hund;

        printf("\n  N=%d SEED=%d SOL=%d TIME=%.2f",n,seed,min,t2-t1);
}
```

```
// PROGRAM 6.2
// MIN-MAX TRAVELLING SALESMAN PROBLEM USING LEXI-SEARCH
// PATH REPRESENTATION
#include<stdio.h>
#include<stdlib.h>
#include<dos.h>
void main()
 { struct time t;
   float t1,t2;
  int i,j,k,l,i1,i2,i3,n,seed,temp,index,dis,temp1,check,elt,bound,iter;
   int str[41],y[40],p[40],vv[40],d[40][40],x[40][40],max,arc,min,max1;
    printf(" ENTER THE No. OF CITIES AND A SEED : ");
     scanf("%d%d",&n,&seed);
//   RANDOM MATRIX GENERATION
    srand((unsigned)seed);
    for(i=1;i<=n;i++)
     for(j=1;j<=n;j++)
      d[i][j]=(rand()%100)+1;
    for(i=1;i<=n;i++)
     d[i][i]=999;
    gettime(&t);
    t1= 3600*t.ti_hour + 60*t.ti_min + t.ti_sec + 0.01*t.ti_hund;
//   ALPHABET TABLE
    for(i=1;i<=n;i++)
     { for(j=1;j<=n;j++)
      vv[j]=0;
       for(j=1;j<=n;j++)
      { max=9999;
        for(k=1;k<=n;k++)
         if((vv[k]==0)&&(d[i][k]<max))
          { index=k;
            max=d[i][k];
          }
         x[i][j]=index;
         vv[index]=1;
      }
     }
//    MAIN PROGRAM STARTS HERE
     max=9999;
     for(j=1;j<=n;j++)
      { vv[j]=0;
      y[j]=0;
      p[j]=0;
      }
     k=i=p[1]=vv[1]=str[n+1]=1;
 GS: k=k+1;
     j=y[k]+1;
 NC: temp=p[k];
     temp1=x[i][j];
     if(vv[temp1]==1)
 JB:  { j=j+1;
      if(j>=n)
       goto JO;
      goto NC;
      }
     dis=d[p[k-1]][temp1];
     if(dis>=max)
```

```
      goto JO;
     bound=0;
     for(i1=1;i1<=n;i1++)
      { if(vv[i1]==0)
       { for(i2=1;i2<6;i2++)
          { elt=x[i1][i2];
            if((elt!=templ) && ((vv[elt]==0)||(elt==1)))
             { if(d[i1][elt]>bound)
              bound=d[i1][elt];
             goto next_row;
             }
          }
         elt=x[i1][6];
         if(d[i1][elt]>bound)
          bound=d[i1][elt];
       }
      next_row:
      }
     if(bound>=max)
      goto JB;
     p[k]=templ;
     if(temp!=0)
      y[k+1]=0;
     y[k]=j;
     vv[templ]=1;
     i=templ;
     if(k==n)
      { dis=d[templ][1];
      if(dis>=max)
       goto PREV;
      for(j=1;j<=n;j++)
       str[j]=p[j];
      max1=0;
      for(j=1;j<=n;j++)
       { check=d[str[j]][str[j+1]];
         if(check>max1)
          max1=check;
       }
      if(max1>=max)
       goto PREV;
      max=max1;
PREV:  vv[p[k]]=0;
JO:  k=k-1;
      if(k<=1)
       goto STOP;
      vv[p[k]]=0;
      j=y[k]+1;
      i=p[k-1];
      goto NC;
      }
     goto GS;
STOP:
      gettime(&t);
      t2= 3600*t.ti_hour + 60*t.ti_min + t.ti_sec + 0.01*t.ti_hund;
      printf("\n  N=%d SEED=%d SOL=%d TIME=%.2f",n,seed,max,t2-t1);
}
```

```
// PROGRAM 6.3
// MIN-MAX TSP USING SEQUENTIAL CONSTRUCTIVE SAMPLING
// APPROACH

#include<stdio.h>
#include<stdlib.h>
#include<dos.h>

void main()
 {
   struct time t;
   int n,max,i,j,k,l,i1,i2,i3,dis,min,idx1,idx2,seed,temp,elt,bound;
   int vv[41],v[41][41],d[41][41],x[41][41],str[41],p[41],y[41];
   int temp1,check,pop,new1,new2,new3,new4,old1,old2,old3,old4;
   float t1,t2,rnd,sum,py[41];

   printf(" ENTER THE No. OF CITIES, SEED AND No. OF SAMPLES :");
    scanf("%d%d%d",&n,&seed,&pop);

//  RANDOM MATRIX GENERATION
    srand((unsigned)seed);
    for(i=1;i<=n;i++)
     for(j=1;j<=n;j++)
      d[i][j]=(rand()%100)+1;
    for(i=1;i<=n;i++)
     d[i][i]=999;

    gettime(&t);
    t1= 3600*t.ti_hour + 60*t.ti_min + t.ti_sec + 0.01*t.ti_hund;

//  ALPHABET TABLE
    for(i=1;i<=n;i++)
     { for(j=1;j<=n;j++)
      vv[j]=0;
       for(j=1;j<=n;j++)
      { min=9999;
        for(k=1;k<=n;k++)
         if((vv[k]==0)&&(d[i][k]<min))
          { idx1=k;
            min=d[i][k];
          }
         x[i][j]=idx1;
         vv[idx1]=1;
      }
     }

// MAIN PROGRAM
      min=9999;
      for(i=1;i<=pop;i++)
       { for(j=1;j<=n;j++)
        { vv[j]=0;
          str[j]=0;
        }
       str[1]=vv[1]=p[n+1]=1;
       for(j=2;j<=n;j++)
        { l=0;
          for(k=1;k<=n;k++)
         y[k]=0;
        i1=n-j+1;
        temp=str[j-1];
        for(k=1;k<=n;k++)
         { elt=x[temp][k];
           if((vv[elt]==0) && (d[temp][elt]<999))
          { l=l+1;
            y[l]=elt;
            if(l>i1/10.0+1)
             goto NEXT;
          }
         }
    NEXT: py[0]=0;
        sum=l*(l+1)/2.0;
        for(k=1;k<=l;k++)
         py[k]=py[k-1]+ (l-k+1)/sum;
        i3=0;
    REPT: rnd=(rand()%100)*0.01;
        i3++;
        if(i3>l)
         goto OUT;
        for(k=1;k<=l;k++)
         if(rnd>=py[k-1] && rnd<py[k])
          { str[j]=y[k];
          goto EXIT;
          }
        EXIT:
        dis=d[temp][str[j]];
        if(dis>=min)
         goto REPT;
        templ=str[j];

// BOUND CALCULATION
        bound=0;
        for(i1=1;i1<=n;i1++)
         { if(vv[i1]==0)
          { for(i2=1;i2<6;i2++)
             { elt=x[i1][i2];
               if((elt!=templ) && ((vv[elt]==0)||(elt==1)))
                { if(d[i1][elt]>bound)
                 bound=d[i1][elt];
                goto next_row;
                }
             }
            elt=x[i1][6];
            if(d[i1][elt]>bound)
             bound=d[i1][elt];
          }
           next_row:
         }
        if(bound>=min)
         goto REPT;
        vv[str[j]]=1;
      }
     dis=d[str[n]][1];
     if(dis<min)
      { for(j=1;j<=n;j++)
```

```
            p[j]=str[j];
           max=0;
           for(j=1;j<=n;j++)
            { check=d[p[j]][p[j+1]];
              if(check>max)
             max=check;
            }
           if(max>=min)
            goto OUT;
           min=max;
         }
        OUT:
        }

// MODIFIED 2-OPT MOVE
      for(j=2;j<n-1;j++)
       for(k=j+2;k<=n;k++)
      { old1=d[p[j-1]][p[j]];
        new1=d[p[j-1]][p[k]];
        old2=d[p[j]][p[j+1]];
        new2=d[p[k]][p[j+1]];
        old3=d[p[k-1]][p[k]];
        new3=d[p[k-1]][p[j]];
        old4=d[p[k]][p[k+1]];
        new4=d[p[j]][p[k+1]];
        if(new1<old1 && new2<old2 && new3<old3 && new4<old4)
         { temp=p[k];
           p[k]=p[j];
           p[j]=temp;
         }
      }

    min=0;
    for(j=1;j<=n;j++)
      { check=d[p[j]][p[j+1]];
        if(check>min)
          min=check;
      }

    gettime(&t);
    t2= 3600*t.ti_hour + 60*t.ti_min + t.ti_sec + 0.01*t.ti_hund;

    printf("\n  N=%d SEED=%d SOL=%d TIME=%.2f",n,seed,min,t2-t1);
}
```

```
//  PROGRAM 6.4
//  MIN-MAX TSP USING HYBRID GENETIC ALGORITHM

#include<stdio.h>
#include<stdlib.h>
#include<dos.h>

int n,pop,chr[601][41],d[41][41],dis;

void create();
void objective(int);
void repro_duction();
void cross_over(float);
void mutation(float);
void seq_search();

void main()
  {
    struct time t;
    float pcr,pmt,t1,t2;
    int i,j,seed,gen,min,str[72],vv[72];
    printf("ENTER THE No. OF CITY and A SEED RESPECTIVELY: ");
     scanf("%d%d",&n,&seed);
  printf("ENTER POP_SIZE,PROB.OF CROSSOVER AND MUTATION RESPECTIVELY:");
     scanf("%d%f%f",&pop,&pcr,&pmt);

//  RANDOMLY GENERATING THE DISTANCE MATRIX
    srand((unsigned)seed);
    for(i=1;i<=n;i++)
     for(j=1;j<=n;j++)
      d[i][j]=(rand()%100)+1;
    for(i=1;i<=n;i++)
     d[i][i]=999;

    gettime(&t);
    t1= 3600*t.ti_hour + 60*t.ti_min + t.ti_sec + 0.01*t.ti_hund;

//  TERMINATING CRITERION
    min=d[1][1];
    create();
    for(i=1;i<=pop;i++)
     { objective(i);
       if(dis<min)
      { min=dis;
        for(j=1;j<=n;j++)
         str[j]=chr[i][j];
      }
     }
    for(gen=1;gen<=4*n;gen++)
     { repro_duction();
       cross_over(pcr);
       mutation(pmt);
       seq_search();
       for(i=1;i<=pop;i++)
      { objective(i);
        if(dis<min)
         min=dis;
      }
     }
    gettime(&t);
    t2= 3600*t.ti_hour + 60*t.ti_min + t.ti_sec + 0.01*t.ti_hund;
    printf("\nN=%d Best Solution=%d Time=%.2f ",n,min,t2-t1);
 }

// INITIALIZE THE POPULATION
void create()
 { int m,i,j,i1,elt,count,vv[41];
   for(i=1;i<=pop;i++)
    { for(j=1;j<=n;j++)
       { vv[j]=0;
       chr[i][j]=0;
       }
      chr[i][1]=1;
      vv[1]=1;
      for(j=2;j<=n;j++)
       { elt=rand() % (n-j+1) +1;
       count=0;
       for(i1=2;i1<=n;i1++)
        if(vv[i1]==0)
         { count++;
           if(count==elt)
            { chr[i][j]=i1;
            goto rxx;
            }
         }
       rxx:
       vv[i1]=1;
       }
    }
 }

// CALCULATE THE VALUE OF THE OBJECTIVE FUNCTION
 void objective(int i)
  { int arc,j;
    dis=0;
    chr[i][n+1]=1;
    for(j=1;j<=n;j++)
     { arc=d[chr[i][j]][chr[i][j+1]];
       if(arc>dis)
      dis=arc;
     }
  }

// STOCHASTIC REMAINDER SELECTION METHOD
void repro_duction()
 {  float fit[601],random,avg,expect,sum=0,frac[601];
    int i,j,k,assign,select,choice[601];
    for(i=1;i<=pop;i++)
     { objective(i);
       fit[i]=1.0/(float)(dis+1);
       sum+=fit[i];
     }
    avg=sum/pop;
    k=0;
    for(i=1;i<=pop;i++)
     { expect=fit[i]/avg;
       assign=expect;
       frac[i]=expect-assign;
       while(assign>0)
      { k=k+1;
        assign-=1;
        choice[k]=i;
      }
     }
    i=0;
    while(k<pop)
     { i=i+1;
       if(i>pop)
      i=1;
       if(frac[i]>0.0)
      { random=(rand()%1000)*0.001;
        if(frac[i]>random)
          { k=k+1;
            choice[k]=i;
            frac[i]-=1.0;
          }
      }
     }
    for(i=1;i<=pop;i++)
     { select=choice[i];
       for(j=1;j<=n;j++)
      chr[i][j]=chr[select][j];
     }
}
```

// C1-CROSSOVER OPERATION

```
 void cross_over(float pc)
  { int i,j,k,m,cross,min,max,cost,v1[41],v2[41],x,y,nwchr[21][41];
    float random;
    for(i=1;i<pop;i++)
     { random=(rand()%100)*0.01;
       if(random<pc)
      { for(m=1;m<=20;m=m+2)
          { for(j=1;j<=n;j++)
             { v1[j]=0;
             v2[j]=0;
             }
            cross=(rand()%(n-2))+2;
            for(j=1;j<cross;j++)
             { x=chr[i][j];
             y=chr[i+1][j];
             nwchr[m][j]=x;
             nwchr[m+1][j]=y;
             v1[x]=1;
             v2[y]=1;
             }
            k=0;
            for(j=cross;j<=n;j++)
    read1:     { k=k+1;
             x=chr[i+1][k];
             if(v1[x]==1)
```

```
                goto read1;
               v1[x]=1;
               nwchr[m][j]=x;
               }
             k=0;
             for(j=cross;j<=n;j++)
    read2:      { k=k+1;
               y=chr[i][k];
               if(v2[y]==1)
                goto read2;
               v2[y]=1;
               nwchr[m+1][j]=y;
               }
           }
         min=999;

         for(m=1;m<=20;m++)
          { nwchr[m][n+1]=1;
            max=0;
            for(j=1;j<=n;j++)
             { cost=d[nwchr[m][j]][nwchr[m][j+1]];
              if(cost>max)
               max=cost;
              }
            if(max>=min)
             goto IGNOR;
            else
             { min=max;
              for(j=1;j<=n;j++)
               chr[i][j]=nwchr[m][j];
              }
           IGNOR:
           }
      }
     }
   }
```

// BITWISE MUTATION OPERATION

```
  void mutation(float pm)
   { int temp,m,i,j,r1,r2;
     float random;
      for(i=1;i<=pop;i++)
       { m=rand()%n+1;
         for(j=1;j<=m;j=j+2)
          { random=(rand()%100)*0.01;
            if(random<pm)
             { r1=rand()%(n-1)+2;
             r2=rand()%(n-1)+2;
             temp=chr[i][r1];
             chr[i][r1]=chr[i][r2];
             chr[i][r2]=temp;
             }
          }
       }
   }
```

```
// SEQUENTIAL CONSTRUCTIVE SEARCH APPROACH
 void seq_search()
  { int i,j,k,l,m,temp,x[3],v[41],nwchr[41],cost[3],min;
    for(i=1;i<pop;i++)
     { for(j=1;j<=n;j++)
      { v[j]=0;
        nwchr[j]=0;
      }
       nwchr[1]=v[1]=1;
       for(k=2;k<=n;k++)
      { for(l=0;l<=1;l++)
         { x[l]=0;
           cost[l]=d[1][1];
           for(j=1;j<=n;j++)
            { if(chr[i+1][j]==nwchr[k-1])
             { for(m=1;m<=n;m++)
                { if(j+m>n)
                   goto next;
                  if(v[chr[i+1][j+m]]==0)
                   { cost[l]=d[chr[i+1][j]][chr[i+1][j+m]];
                   x[l]=chr[i+1][j+m];
                   goto next;
                   }
                }
             }
            }
           next:
         }
        if(cost[1]==d[1][1] && cost[0]==d[1][1])
         { min=d[1][1];
           for(j=1;j<=n;j++)
            if(v[j]==0)
             { temp=d[nwchr[k-1]][j];
             if(temp<min)
              { nwchr[k]=j;
                min=temp;
              }
             }
           goto ACCEPT;
         }
        nwchr[k]=x[0];
        if(cost[1]<cost[0])
         nwchr[k]=x[1];
ACCEPT:  v[nwchr[k]]=1;
      }
      for(j=1;j<=n;j++)
       chr[i][j]=nwchr[j];
     }
  }
```

BIBLIOGRAPHY

1. Balas, E., and N. Christofides (1981): **"A Restricted Lagrangian Approach to The Travelling Salesman Problem".** *Mathematical Programming, 21*, 19-46.
2. Bellman, R. E. (1962): **"Dynamic Programming Treatment of the Travelling Salesman Problem".** *Journal of ACM, 9*, 61-63.
3. Bellmore, M. and J. C. Mellone (1971): **"Pathology of Travelling Salesman Sub-tour Elimination Algorithms'.** *Operations Research, 19*, 278-307.
4. Bentley, J. L. (1992): **"Fast Algorithms for Geometric Travelling Salesman Problems".** *ORSA Journal of Computer, 4*, 387-411.
5. Bhatia, S. K., and A. N. Rocha (1987): **"Travelling Salesman".** *Computer Society of India Communications*, 3-10.
6. Bianco, L., Mingozzi, S., Ricciardelli, S., and Spadoni, M. (1994): **"Exact and Heuristic Procedures for the Travelling Salesman Problem with Precedence Constraints, Based on Dynamic Programming",** *Information Systems & Operational Research, 32*, 19-32.
7. Bland, R. G. and D. F. Shallcross (1989): **"Large Travelling Salesman Problems arising form Experiments in X-ray Crystallography: A Preliminary Report on Computation".** *Operations Research Letters, 8*, 125-128.
8. Chisman, J. A. (1975): **"The Clustered Travelling Salesman Problem".** *Computers Ops Res, 2*, 115-119.
9. Christofides, N. (1970): **"The Shortest Hamiltonian Chain of a Graph".** *SIAM Journal of Applied Mathematics, 19*, 689-696.
10. Clarke, G., and J. W. Wright (1964): **"Scheduling of Vehicles from a Central Depot to a Number of Delivering Points".** *Operations Research, 12*, 568-581.
11. Croes, A. (1958): **"A Method for Solving the Travelling Salesman Problem".** *Operations Research, 5*, 791-812.
12. Das, S. (1976): **"Routing And Allied Combinatorial Programming Problems: A Lexicographic Search Approach".** *Ph.D. Thesis*, Dibrugarh University, Assam, India.
13. Dantzig, G. B., D. R. Fulkerson, and S. M. Johnson (1954): **"Solution of a Large-scale Travelling Salesman Problem".** *Operations Research, 2*, 393-410.

14. Davis, L. (1985): **"Job-shop Scheduling with Genetic Algorithms"**. *Proceedings of an International Conference on Genetic Algorithms and Their Applications,* 136-140.

15. Dawkins, R. (1989): **"The Selfish Gene"**. 2nd edition, Oxford University Press, Oxford.

16. Deb, K. (1995): **"Optimization For Engineering Design: Algorithms And Examples".** Prentice Hall Of India Pvt. Ltd., New Delhi, India.

17. Deif, I. and L. D. Bodin (1984): **"Extention of the Clarke and Wright algorithm for solving the Vehicle Routine with Backhauling"**. *Proceeding of the Babson Conference on Software Uses in Transportation and Logistics Management* (Edited by A. E. Kidder), Babson Park, Mass., 75-96.

18. Eastman, W. L. (1958): **"Linear Programming with Pattern Constraints"**. *Ph. D. Thesis,* Harvard.

19. Frederickson, G. N., M. S. Hecht, and C. E. Kin (1978): **"Approximation algorithms for Some Routing Problems"**. *SIAM Journal of Computer, 7,* 178-193.

20. Freisleben, B. and P. Merz (1996): **"A Genetic Local Search Algorithm for Solving the Symmetric and Asymmetric TSP"**. *In the Proceedings of IEEE International Conference on Evolutionary Computation,* Nagoya, Japan, 616-621.

21. Gendreau, M., A. Hertz, and G. Laporte (1992): **"New Insertion and Post-Optimization Procedures for the Travelling Salesman Problem"**. *Operations Research, 40,* 1086-1094.

22. Gendreau, M., A. Hertz, and G. Laporte (1996): **"The Travelling Salesman Problem with Backhauls"**. *Computers Ops Res., 23,* 501-508.

23. Gendreau, M., G. Laporte, and A. Hertz. (1997):**"An Approximation Algorithm For the Travelling Salesman Problem with Backhauls"**. *Operations Research, 45,* 639-641.

24. Goldberg, D.E. (1989): **"Genetic Algorithms In Search, Optimization, And Machine Learning".** Addison-Wesley, New York.

25. Goldberg, D.E., and R. Lingle (1985): **"Alleles, Loci and the Travelling Salesman Problem"**. In [27], 154-159.

26. Gomory, R. E. (1963): **"An Algorithm for Integer Solutions to Linear Programs"**. *In Recent Advances in Mathematical Programming,* (R. L. Graves and P.Wolfe, eds), 269-302, McGraw-Hill, New York.
27. Grefenstette, J. J. (ed.) (1985): **"Proceedings of the 1st International Conference on Genetic Algorithms and Their Applications"**. Lawrence Erlbaum Associates, Hilladale, NJ.
28. Grefenstette, J. J. (ed.) (1987): **"Genetic Algorithms and Their Applications: Proceedings of the 2nd International Conference on Genetic Algorithms"**. Lawrence Erlbaum Associates, Hilladale, NJ.
29. Gupta, J. N. D. (1969): **"A General Algorithm for the n x m Flow-shop Scheduling Problems"**. *International Journal of Production Research, 7,* 1-7.
30. Held, M. and R. Karp (1971): **"The Travelling Salesman Problem and Minimum Spanning Trees: Part II"**. *Operations Research, 18,* 1138-1162.
31. Held, M. and R. Karp (1962): **"A Dynamic Programming Approach to Sequencing Problems"**. *SIAM Journal of Applied Mathematics, 10,* 196-210.
32. Held, M. and R. Karp (1971): **"The Travelling Salesman Problem and Minimum Spanning Trees: Part II"**. *Mathematical Programming, 1,* 6-25.
33. Holland, J.H. (1975): **"Adaptation In Natural And Artificial Systems"**. University Of Michigan Press, Ann Arbor, MI.
34. Johnson, D. S. (1990): **"Local Optimization and the Travelling Salesman Problem"**. *In Lectures Notes in Computer Science 443,* M. S. Paterson (ed.), Springer-Verlag, Berlin, 446-461.
35. Jongens, K., And T. Volgenant (1985): **"The Symmetric Travelling Salesman Problem"**, *Eur. J. Opl Res.*, 19, 68-75.
36. Kalantari, B., A.V. Hill, and S.R.Arora (1985):**"An Algorithm For The Travelling Salesman Problem With Pickup And Delivery Customers"**, *Eur. J. Opl Res., 22,* 377-386.
37. Kirkpatrick, S., C. D. Gellat and M. P. Vecchi (1983): **"Optimization by Simulated Annealing"**. *Science, 220,* 671-680.
38. Kothari, C. R. (1992): **"An Introduction to Operational Research"**. Vikas Publishing House Pvt. Ltd., New Delhi.

39. Laporte, G., Y. Nobert, and M. Desrochers (1985): **"Optimal Routng Under Capacity and Distance Restrictions"**. *Operations Research, 5*.

40. Lawler, E. L., J. K. Lenstra, A. H. G. Rinooy Kan, and Shnoys (eds.) (1985): **"The Travelling Salesman Problem: A Guided Tour in Combinatorial Optmization"**. John Wiley and Sons Ltd., New York.

41. Lin, S. (1963): **"Computer Solutions of the Travelling Salesman Problem"**. *Bell Systems Technical Journal, 44*, 2245-2269.

42. Lin, S., and B. W. Kernighan (1973): **"An Efficient Heuristic Algorithm for the Travelling Salesman Problem"**. *Operations Research, 21*, 498-516.

43. Little, J.D.C., K.G. Murthy, D.W. Sweeny, and C. Karel (1963): **"An Algorithm For The Travelling Salesman Problem"**. *Operations Research, 11*, 972-989.

44. Lokin, F. C. J. (1978): **"Procedures For Travelling Salesman Problems With Additional Constraints"**. *Eur. J. Opl Res, 3*, 135-141.

45. Mak, K., and A. J. Morton (1993): **"A Modified Lin-Kernighan Travelling Salesman Heuristic"**. *Operations Research Letters, 13*, 127-132.

46. Martin, O., S. W. Otto, and E. Felten (1991): **"Large-Step Markov Chains for the Travelling Salesman Problem Incorporating Local Search heuristics"**. *Operations Research Letters, 11*, 219-224.

47. Martin, O., S. W. Otto, and E. Felten (1991): **"Large-Step Markov Chains for tthe Travelling Salesman Problem"**. *Computer System, 5*, 299-326.

48. Metropolis, N., A. W. Rosenbluth, M. N. Roshenbluth, A. H. Teller, and E. Teller (1953): **"Equation of State Calculation by Fast Computing Machines"**. *Journal of Chemical Physics, 21*, 1087-1091.

49. Michalewicz, z, (1994): **"Genetic Algorithms + Data Structures = Evolution Problems"**. Second Edition, Springer-Verlag, New York.

50. Miller, C. E., A. W. Tucker, and R. A. Zelmin (1960): **"Integer Programming Formulations and Travelling Salesman Problems"**. *Journal of ACM, 7*, 326-332.

51. Mingozzi, A., L. Bianco, and S. Ricciardelli (1997): **"Dynamic Programming for the Travelling Salesman Problem with Time Window and Precedence Constraints"**. *Operations Research, 45*, 365-377.

52. Mühlenbein, H. M. Gorge-Schleuter, and O. Krämer (1988): **"Evolution Algorithms in Combinatorial Optimization"**. *Parallel Computing, 7*, 65-85.

53. Oliver, I. M., D. J. Smith, and J.R.C. Holland (1987): **"A Study of Permutation Crossover Operators on the Travelling Salesman Problem"**. In [28], 224-230.

54. Oliveira, P., S. McKee, and C. Coles (1994): **"Genetic Algorithms and Optimizing Large Non-linear Systems"**. *In Procedings of the IMA conference on Artificial Intelligence in Mathematics* (eds): J. H. Johnson, S. McKee and A. Vella, Oxford University Press. 305-312, Oxford.

55. Or, I. (1976): **"Travelling Salesman – Type Combinatorial Problem and Their Relation to the Logistics of Regional Blood Banking"**. *Ph. D. Dissertation*, North-Western University, Evanston, III. (Ill).

56. Pandit, S.N.N. (1962): **"The Loading Problem"**. *Operations Research, 11*, 639-646.

57. Pandit, S.N.N. (1963): **"Some Quantatitive Combinatorial Search Prblems"**. *Ph. D. Thesis*, IIT, Kharagpur.

58. Pandit, S.N.N. (1964): **"An Intelligent Approach to Travelling Salesman Problem"**. *Symposium in Operations Resaerch*, IIT, Kharagpur.

59. Pandit, S.N.N., S.C. Jain, and R. Misra (1964): **"Optimal Machine Allocation"**. *Journal of Institute of Engineers, 44*, 226-240.

60. Pandit, S.N.N., and M. Ravikumar. (1993): **"A Lexicographic Search For Strongly Correlated 0-1 Knapsack Problems"**. *OPSEARCH, 30*, 97-116.

61. Papadimitriou, C. H. and K. Steglitz (1997): **"Combinatorial Optimization: Algorithms and Complexity"**. Prentice Hall of India Private Limited, India.

62. Psaraftis, H.N. (1980): **"A Dynamic Programming Solution To The Single Vehicle Many-To-Many Immediate Request Dial-A-Ride Problem"**, *Transportation Science*, 14.

63. Psaraftis, H.N. (1983): **"An Exact Algorithm for the Single Vehicle Many-To-Many Dial-A-Ride Problem with Time Windows"**, *Transportation Science, 17*, 351-357.

64. Radcliffe, N.J., P.D. Surry (1995): **"Formae and variance of fitness".** In D. Whitley and M. Vose (Eds.) (1995) *Foundations of Genetic Algorithms 3*, Morgan Kaufmann, San Mateo, CA, 51-72.

65. Ramesh, M. (1997): **"A Lexisearch Approach To Some Combinatorial Programming Problems"**, *Ph.D. Thesis*, University Of Hyderabad, India.

66. Ramesh, T. (1980): **"(Traveller Purchaser Problem): Some Problems in Min-Max and Combinatorial Programming"**. *Ph. D. Dissertation*, Kakatiya University, Warangal.

67. Ravikumar, C. P. (1992): **"Solving Larege-scale Travelling Salesperson Problems on Parallel Machines"**. *Microprocessors and Microsystems, 16(3),* 149-158.

68. Ravikumar, M. (1994): **"Data Guided Algorithms In Combinatorial Optimization"**. *Ph.D. Thesis*, Osmania University, India.

69. Reddy, S. C. (1994): **"Quasi Assignment Problem: A Lexi Search Approach"**. *M. Phil. Thesis*, Osmania University, India.

70. Reeves, C.R. (Ed.) (1993): **"Modern Heuristic Techniques For Combinatorial Problems"**. Orient Longman Ltd, India.

71. Reinelt, G. (1992): **"Fast Heuristics for Large Geometric Travelling Salesman Problems"**. *ORSA Journal of Computer, 4*, 206-217.

72. Reinelt, G. (1995) **"TSPLIB: A library of sample instances for the TSP (and related problems) from various sources and of various types"** *Web site: http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/*

73. Rooij, J. F., L. C. Jain, And R. P. Johnson. (1996): **"Neural Network Training Using Genetic Algorithms"**. World Scientific Publishing Co. Pte. Ltd., Singapore.

74. Savelsbergh, M. W. P. (1990): **"An Efficient Implementation of Local Search Algorithms for Constrained Routing Problems"**. *European Journal of Operational Research, 47*, 75-85.

75. Schwefel, H. P., and R. Männer (eds.): **"Proceedings of the 1st International Conference on Parallel Problem Solving from Nature (PPSN)"**. *Lecture Notes in Computer Science, 496,* Springer-Verlag.

76. Scroggs, R.E., and A.L. Therp (1972): **"An Algorithm For Solving The Travelling Salesman Problem In Restricted Context".** Privately Communicated.

77. Shapiro, D.M. (1966): **"Algorithms for the Solution of the Optimal Cost and Bollteneck Travelling Salesman Problems".** *Sc.D. Thesis*, Washington University, St. Louis, MO.

78. Smith, S. F. (1984): **"Adaptive Learning Systems".** In R. Forsyth (ed.) *Expert Systems- Principles and Case Studies*, 168-189.

79. Smith, T. H. C., V. Srinivasan, and G. L. Thompson (1977): **"Computational Performance of Three Sub-tour Elimination Algorithms for Solving Asymmetric Travelling Salesman Problems".** *Annals of Discrete Mathematics, 1*, 495-506.

80. Srinivas, K. (1989): **"Data Guided Algorithms In Optimization And Pattern Recognition".** *Ph.D. Thesis*, University Of Hyderabad, India.

81. Stewart (Jr), W. R. (1987): **"Accelerated Branch Exchange Heuristics for Symmetric Travelling Salesman Problems".** *Networks, 17*, 423-437.

82. Sumana, A. (1995): **"Optimization By Simulated Annealing: An Empirical Evaluation".** *M.Phil. Thesis*, University Of Hyderabad, India.

83. Sundara Murthy, M. (1979): **"Some Combinatorial Search Problems (A Pattern Recognition Approach)".** *Ph. D. Thesis*, Kkatiya University, Warangal.

84. Ulder, N. L. J., E. H. L. Aarts, H.-J. Bandelt, P. J. M. Van Van Laarhoven, E. Pesch (1991): **"Genetic Local Search Algorithms for the Travelling Salesman Problem".** In [73], 109-116.

85. Whitley, D., T. Starkweather and D. Shaner (1991): **"The traveling salesman and sequence scheduling: Quality solutions using genetic edge recombination".** In L. Davis (Ed.) (1991) *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York, 350-372.

86. Zweig, G. (1995): **"An effective Tour Construction and Improvement Procedure for the Travelling Salesman Problem".** *Operations Research, 43*, 1049-1057.