Imperial College of Science, Technology and Medicine

(University of London)

The Management School

CONTRIBUTIONS TO THE SOLUTION OF THE SYMMETRIC TRAVELLING SALESMAN PROBLEM

by

Mohan Krishnamoorthy

A thesis submitted for the degree of

Doctor of Philosophy of the University of London

November 1990

Intelligence designs

But the heart does the modelling

Rodin

to

thatha, pati

and

my parents

ACKNOWLEDGEMENTS

Firstly, I would like to express my thanks to Professor Nicos Christofides, my supervisor, for his help, guidance and advice. Despite his busy schedules, he gave me his time and encouraged me on many an occasion, all of which made working with him a valuable learning experience.

I am grateful to Dr. W S Bardo of GEC-Marconi for having agreed to support this research by providing me with a grant, without which this PhD would not have been possible.

I would also like to acknowledge the help and encouragement of Dr J E Beasley who gave me his time for several discussions during the course of this work. Thanks also to Dr Anton Volgenant (of the Instituut voor Actuariaat en Econometrie, Universiteit van Amsterdam, The Netherlands) for having provided me with his optimal code for the TSP and most of the test data.

I would not have emerged out of this effort half as sane as I actually have if it had not been for the excellent team spirit and work atmosphere of the OR & Systems Research Unit of the Management School. The largely cosmopolitan nature of this group made this PhD a very enjoyable cultural learning process too. Of course, the numerous coffee breaks, heated debates and arguments meant that we solved more problems that beset the world than we did problems in Operational Research ! I am particularly thankful to George Megalokonomos for his encouragement, help and constant advice; to Andreas Johannsen for his support and friendship (it also helped that he lived a *mere* five minutes away from where I lived !); to "can-I-say-something" Yazid Sharaiha for his motivating influence and his entertaining

disruptions; to Niki Niktari for her generous help with the proof reading of certain chapters of this dissertation; to Ibrahim Osman and Merza Hassan for egging me on when the spirit was low; to Jacob Yadegar, with whom I collaborated fruitfully on the production of Chapter 5 of this dissertation; to Katy Steward for her help with the proof reading, her useful comments and suggestions. Any remaining errors should be attributable only to me ! The help I have received from all of them, particuarly during the last stages of my research, is greatly appreciated.

There have been many friends from outside the university who have earned my gratitude by virtue of their encouragement and support. In particular, I would like to acknowledge the support of Anna Garifalli and Helen Mundy.

Thanks also to Dr Norman Murray, the computer manager of the school, for patiently answering many of my naive questions on computing; to Ms Edna Archer and Ms Amanda Sage, the librarians, for their invaluable help; to Paul Belli and Mark Forster of the Imperial College computer centre for helping me get to grips with the mainframe computers I used during the course of this study.

The support, love and understanding I have received from my entire family is incalculable. The knowledge that they were always there when I needed them to assuage my worst fears and anxieties ensured the completion of this dissertation. Particular thanks go to Rajesh for having put up with my erratic schedules (particularly during the last few months of my PhD) and for being extremely supportive.

Last, but not the least, thanks to Lata for her constant encouragement and her patience.

ABSTRACT

The *Travelling Salesman Problem* (TSP) deals with a salesman who, having started from a particular (*home*) city, wishes to visit each city on a given list exactly once before returning home at a minimum total cost. The TSP has served as a test bed for almost every new algorithmic idea in combinatorial optimization. We discuss the motivation for the continued interest in finding good solutions to the TSP before reviewing some of the existing exact and heuristic solution methods to solve it. We also describe some of the popular practical applications of the TSP.

Three different relaxations are considered and applied for the TSP - the Shortest Spanning Tree relaxation, the Assignment Problem relaxation, and the Minimal Spanning Arborescence relaxation. These relaxations provide lower bounds on the solution of the TSP. The lower bounds in each case are derived from solving the respective lagrangean dual problems. The size of the problem is constantly reduced by deleting unwanted arcs from, and forcing required arcs into, the solution. This is achieved by sensitivity analysis methods and through investigating the structural properties of the graph. We also achieve large reductions in the problem size by exploiting the complementary dual properties of the minimal spanning arborescence and the assignment problem relaxations. We suggest a transformation of the symmetric travelling problem into an asymmetric one. This tightens the lower bounds obtained. The lower bounds are imbedded into an exact tree search procedure that incorporates some novel branching strategies.

We describe an upper bound heuristic procedure that is applied at the root node of the tree search. The heuristic, which makes use of the spanning trees produced during the subgradient ascent, provides tight upper bounds. In many of the problems that were tested, near optimal solutions are obtained.

The Peano-Cesaro plane-filling fractal curve maps the unit interval continuously and recursively into the plane. A fast and simple heuristic algorithm, based on this fractal curve, is described and used to solve very large-scale TSPs in the plane.

We present computational results and discuss the performance of all the algorithms that have been developed. Conclusions and pointers for further research are provided.

PREFACE

A few months before I began writing up this dissertation, I was travelling on a train between Madras and New Delhi. As is very common on Indian train journeys, I struck up a conversation with my fellow traveller who happened to be a physicist. "What do you do ?", he asked. I told him that I researched the travelling salesman problem. "Hasn't that problem been solved *yet* ?", he exclaimed.

My fellow travellers' disbelief was quite understandable. The travelling salesman problem (TSP) is one of the most extensively researched problems in operational research (OR) literature - "Literally person-centuries have been devoted to developing a sophisticated solution theory for the TSP", according to Fisher & Jaikumar [1981]. Yet, it retains its position as one of the most intriguing (and technically) "unsolved" paradigms in OR literature. According to Hoffman & Wolfe [1985] - "If, as in the TSP, the problem is to develop an algorithm that satisfies formal or informal standards of efficiency, then the TSP has not yet been solved".

The TSP contains two basic elements that make it an interesting and challenging problem: It is very easy to understand, yet difficult to solve. The simplicity of the problem ensures that research into it is not restricted to a handful of researchers, but to those from all branches of science. The difficulty of the problem acts as a challenge and attracts continued research. It is unlikely that this interest will cease. In the first chapter of this dissertation we will provide insights into the theoretical motivations for the continued search for good algorithms for the TSP. This motivation

is derived from the computational equivalence of hard problems[1], of which the TSP is an archetype.

In Chapter 1, we introduce the problem - its classification in the literature, its complexity and some of its commercial applications. A review of the current literature on the TSP is also provided. For a comprehensive and thorough treatment of all facets of the TSP, refer to the book by Lawler, Lenstra, Rinnooy Kan, & Shmoys [1985].

Chapters 2 to 5 contain the main contributions of this research. In Chapter 2, we analyze the minimal spanning tree as a relaxation for the TSP. We introduce some efficient methods to induce graph sparsity. We also give an upper bound algorithm (that is conveniently imbedded into the relaxation procedure) that produces tight upper bounds on the optimal solution value. In Chapter 3, we investigate the relationship between two complementary relaxations for the TSP - the assignment problem and the minimal spanning arborescence problem. The duals obtained through these two relaxations are used to induce greater sparseness in the graph in an iterative fashion. This, in turn, augments the lower bound. In Chapter 4, we describe a depth-first branch and bound algorithm that combines all the ideas developed in the earlier chapters. A transformation of the symmetric TSP into an asymmetric one helps tighten the lower bounds from the assignment and arborescence problems. These lower bounds are imbedded into a branch and bound procedure that incorporates some

[1]Mainly due to the efforts of Cook [1971] and Karp [1972], it became evident that many of the problems thought to be inherently hard are all computationally equivalent, in the sense that a polynomial algorithm for one of these problems could be used to solve all other problems in its class, in polynomial time.

new and interesting branching criteria. All these chapters include extensive empirical analysis. To achieve this, we use some well known road map and Euclidean problems from the literature, some new Euclidean problems that have been generated and some randomly generated problems. The geometry of fractals is used in the design of a new heuristic for the solution of large-sized Euclidean TSPs. We describe and analyze the performance of this algorithm in Chapter 5.

Finally, Chapter 6 provides conclusions from the study and gives some directions for further research.

TABLE OF CONTENTS

CHAPTER 1

Introduction

1.0 Outline

In this chapter we introduce the travelling salesman problem. We provide a brief overview of the problem with a view to its classification in the literature, its theoretical importance and its apparent intractability. This will provide some insights into the reasons for the continued search for good solution methods to solve the problem. We also review some of the successful commercial applications of the TSP. In this dissertation, we deal, in the main, with algorithms for the symmetric travelling salesman problem. Integer programming formulations for this variation of the TSP are provided. We offer a review and classification of some of the exact and approximate algorithms for the solution of the problem; past efforts are reviewed and indications of current trends are given. Finally, in this chapter, we make some preliminary remarks on the implementation of the algorithms that will be developed in this dissertation - we include in this, a discussion of the data that will be used to test our algorithms.

1.1 Overview of the problem

The *travelling salesman problem* (TSP) can be stated as follows:

> Consider a salesman who, having started from a particular city (his *home* city), wishes to visit each city on a given list **exactly once** before returning home. If the *cost* of travel between each of the pairs of cities on his list is known, the salesmans' problem is to select the order in which he visits the cities (his *tour*) so that the total cost of his travels is minimized.

The term *cost* here can be taken to mean distance, time, monetary or any other similar unit. If the cost of travel from a city i to a city j is the same as the cost of travel from j to i, then the resulting problem is a special case of the TSP, called the *symmetric travelling salesman problem* (STSP). The general problem - one where the costs are not required to be symmetric - is referred to as the *asymmetric travelling salesman problem* (ATSP) or simply, the TSP. The central theme of this dissertation is the development of algorithms for the STSP. For an interesting review of the history of and the motivation behind the continued research into the TSP refer to Hoffman & Wolfe [1985]. A major survey of research on the TSP is the excellent book by Lawler *et al.* [1985].

The optimal solution to the TSP is a collection of *arcs* that constitute a proper tour with minimum total cost. Selection of an optimal tour is not over a continuum but over the set of feasible tours. Optimization problems of this type are classified as

combinatorial. The TSP belongs to the class of problems known as *combinatorial optimization*, which can be defined (Lawler [1976]) as follows:

> Combinatorial optimization is the mathematical study of finding an optimal arrangement, grouping, ordering, or selection of discrete objects.

Typical problems in combinatorial optimization are: an optimal selection among various investment possibilities, the best grouping of customers, optimal ordering of jobs on machines, optimal location of facilities, etc. Major references on aspects of combinatorial optimization are Lawler [1976], Christofides *et al.* [1979], Papadimitriou & Steiglitz [1982], Nemhauser & Wolsey [1988].

An instance of a combinatorial optimization problem (like the TSP) can be seen as an implicit description of a finite set of feasible solutions. A weight function assigns to each of these solutions a value. The optimal solution is then that feasible solution with minimum (or maximum, depending on the problem) value. These problems are mostly well-defined in the sense that an optimal solution always exists if the set of feasible solutions is non-empty. Thus, it is not so much the *existence* of the set of feasible solutions or, indeed the existence of an optimal solution, but rather the *computational effort* required to obtain the optima which is of central interest in the design of algorithms for combinatorial optimization problems. The computational effort required to solve a problem forms a sound basis for its classification. To this extent, the TSP, which has served as a testing ground for almost every new

algorithmic idea in combinatorial optimization, has enabled the classification of problems as *hard* or *easy*.

1.2 Easy and hard problems

The optimal solution to combinatorial problems can, in principle, be found by complete enumeration. The major objective of research in this area is to avoid complete enumeration due to the computationally explosive nature of problems of this genre.

A major advance in the understanding of these problems was their classification into easy and hard problems. The TSP was one of the first optimization problems thought of as being hard in a specific sense (Edmonds [1965a]); problems for which algorithms are essentially some form of enumeration. To understand these concepts more clearly, we introduce the following informal definition for the *computational complexity* of an algorithm:

> The computational complexity of an algorithm may be defined as (a bound on) the number of elementary operations necessary to solve an instance of a problem, given its *size*, *n*.

Thus, for a set of instances of a problem of given size, this forms a measure of the worst-case behaviour of an algorithm to solve it. The size of an instance of a problem is measured by the length - the number of bits - of the shortest coding necessary to completely specify the input data.

Given an instance of size n and a real function $g(n)$ of n, we say that an algorithm is of complexity $O(g(n))$ if $f(n)$, the maximum time required to execute the algorithm is such that: $|f(n)| < c|g(n)|$. Here, c is a constant (which depends on the type of computer that is used) and $|K|$ denotes the absolute value of K.

Algorithms in $O(n)$ are called *linear*; those in $O(n^k)$ are called *polynomial*; those in $O(2^n)$ are called *exponential*. An algorithm is considered *good* or *efficient* if its worst-case complexity is bounded by a polynomial function. Problems for which an efficient algorithm exists are called easy. They belong to the class *P* of problems. *P* is a subclass of the class of problems which can be solved in polynomial time by a *nondeterministic*[1] algorithm. This class was introduced by Cook [1971] and Karp [1972]. The latter called the class *NP*, for *N*on-deterministic *P*olynomial time problems. The TSP belongs to an important subclass of *NP*, the class of problems known as *NP-complete* (see Karp [1972]). Every *NP*-complete problem "truly" belongs to the class *NP*, in the sense that if there exists a polynomial algorithm to solve it, then all problems in *NP* are polynomially solvable. However, the general conjecture is that such polynomial algorithms do not exist for *NP*-complete problems, for the implication would then be that *P*=*NP* (ie., all problems in *NP* are easy, which is unlikely).

For a review of the complexity of the TSP refer to Johnson & Papadimitriou [1985]. For more general surveys in this area, see Aho, Hopcroft & Ullman [1974], Garey &

[1]A *non-deterministic* algorithm is one that contains statements of choice in addition to the normal *deterministic* statements. The functioning of such an algorithm is analogous to dividing a program into concurrently processed subprograms. If all choices are enumerated, then a non-deterministic algorithm becomes a deterministic one.

Johnson [1979] and Papadimitriou & Steiglitz [1982]. The intractability of the TSP (and other hard problems like it) does not imply that it is unsolvable. The direct implication is that, in the worst-case, the time required to obtain an optimal solution grows at an exponential rate as the number of cities in the problem increases. Moreover, in such instances, there is no guarantee that an optimal solution will be found in a reasonable amount of computing time.

Hence, the classification of algorithms into two broad categories: (a) *exact solution algorithms* that produce the optimal solution, but which carry the risk of having to expend a lot of computing time and the disadvantage of being able to solve only small-sized instances; (b) *heuristic* or *approximate* algorithms that produce a feasible solution in a reasonable amount of computing time with the risk that it may be sub-optimal.

1.3 Notational and methodological preliminaries

Throughout this dissertation, we will use the notation P_X to define a problem X. We will denote the value of an optimal solution to problem X by $Z(P_X)$. We will use $G=(N,A)$ to denote both directed as well as undirected graphs. Here, $N=\{1,\ldots,n\}$ is the set of *n nodes*. In the undirected case (the STSP), $A=\{1,\ldots,m\}$ is the set of m undirected arcs, where $m=\frac{n(n-1)}{2}$; in the directed case (the ATSP), $A=\{(i,j)|i,j=1,\ldots,n\}$ will represent the set of n^2 directed arcs.

Below, we provide a brief introduction of important aspects related to algorithms which will be developed in this dissertation.

1.3.1 Lagrangean relaxation

Consider the following optimization problem:

Problem P_X:

$$\min_x \quad f(x)$$

$$\text{s.t} \quad g_i(x) = b_i \qquad (i = 1, \ldots, m)$$

$$x \in S$$

Here, x is a vector, $f(x)$ and $g_i(x)$, $(i = 1, \ldots, m)$ are arbitrary functions and S is the set of feasible solutions. A *relaxation* of problem P_X is given by:

Relaxation R_X:

$$\min_x \quad f(x)$$

$$\text{s.t.} \quad x \in S$$

A relaxation is, therefore, a subproblem of the original problem in which some of the constraints have been ignored. A *lagrangean relaxation* of the problem is obtained by associating a *lagrange multiplier* λ_i with each constraint $g_i(x) = b_i$. The problem P_X with the lagrangean objective function is then:

Problem $P_X(\lambda)$:

$$\min_{x,\lambda} \quad f(x) + \lambda\left(g(x) - b\right)$$

$$\text{s.t.} \quad x \in S$$

Here, $\lambda = \left(\lambda_1, \ldots, \lambda_m\right)$ and $g(x) = \left[g_1(x), \ldots, g_m(x)\right]^T$. For any given λ, $Z\left(P_X(\lambda)\right)$ is a *lower bound* on the optimal solution of the minimization problem P_X. The problem is then one of determining λ^*, the "best" multipliers that maximizes $Z\left(P_X(\lambda)\right)$ so that the best lower bound is rendered close to $Z\left(P_X\right)$.

For surveys of lagrangean relaxation for combinatorial optimization problems, refer to Geoffrion [1974], Shapiro [1979] and Fisher [1981].

1.3.2 Branch and bound

A branch and bound algorithm to solve the minimization problem P_X is an implicit enumeration procedure. It solves a combinatorial optimization problem by breaking up the set of feasible solutions into successively smaller, manageable subsets. A lower bound on the value of the best solution is obtained by solving a relaxed problem in each of the subsets under consideration. The solution obtained through this relaxation is either: (a) feasible for the original subproblem - no further enumeration of the subset is required; (b) with an associated lower bound which is worse than the value of the best feasible solution obtained so far (an *upper bound*) - in which case the subset is considered implicitly enumerated (*fathomed*); (c) infeasible for the original subproblem but with a lower bound less than the upper bound - in which case the subset is partitioned further and the enumeration continues.

This process is also known as a *tree search* procedure. The main ingredients of a branch and bound algorithm are:

- o *branching rules*: a rule for partitioning the feasible set S_i, of the current subproblem P_{X_i} into subsets $S_{i1},\ldots,S_{iq}$ with the help of *branching constraints* such that $\bigcup_{j=1}^{q} S_{ij}=S_i$.
- o *lower bounding method*: the relaxed subproblem to be solved.
- o *upper bounding method*: a heuristic algorithm for finding feasible solutions to problem P_X.
- o *search strategy*: rules for choosing the next subproblem to be considered. Some rules commonly used are depth first (LIFO), breadth first (FIFO), best first, and mixed strategies.

The branch and bound procedure can be depicted by a rooted tree where the root node corresponds to the original problem. The nodes of the tree correspond to subproblems in the enumeration process. The branching rules determine the successor nodes of a particular node in the search tree. To avoid any confusion in terminology between the nodes of the tree search and nodes of the graph, we will sometimes refer to the nodes of the tree search as tree nodes.

For general surveys on branch and bound methods see Garfinkel & Nemhauser [1972, Chapter 4], Balas [1975], Balas & Guignard [1979], Beale [1979], Garfinkel [1979] or Spielberg [1979].

1.4 Some applications of the TSP

The greatest value of the TSP is probably its theoretical importance. The TSP is an archetypal combinatorial optimization problem. This, combined with its intrinsic simplicity has encouraged a lot of research into the problem. The TSP is also of great practical importance even though "there are not many salesmen clamouring for an algorithm", Hoffman & Wolfe [1985]. Several real-world problems can be solved using a direct TSP formulation or a variant of it. Some examples are:

Job sequencing:

n jobs are to be processed on a machine, for example, a furnace. Each job has a start temperature t_i and a completion temperature T_i. If job j follows job i, the changeover cost c_{ij} is the cost of the change in temperature (the time lost) from T_i to t_j. The problem of determining an ordering of the jobs such that the total changeover cost is minimized is equivalent to solving a TSP with cost matrix $C=\left[c_{ij}\right]$. This example is attributable to Gilmore & Gomory [1964] who gave a polynomially solvable algorithm after identifying a special structure in a generalized version of the problem.

Vehicle routing:

Given n customers with known locations and demands for some commodity that is supplied from a single depot by m vehicles, each with a known capacity $Q_j \ (j=1,\ldots,m)$, the problem is to design routes for each of the vehicles so that the total travel cost is minimized. Given an assignment of customers to vehicles, the best route for a vehicle in this basic formulation of the routing problem is a TSP.

Computer wiring:

This type of problem is often encountered in the design of computer circuits (refer to Lenstra & Rinnooy Kan [1975]). One aspect of the problem consists of minimizing the total length of wiring used to inter-connect a given subset of *pins* with known position on *modules* such that exactly two wires are attached to any pin.

Automation in manufacturing:

A more recent and interesting example is the one that occurs in the production of printed circuit boards. Some of the manufacturing operations (like drilling and component insertion) are performed by numerically controlled machines. The problem of optimizing the sequence of machine movements in order to reduce the overall processing time can be modelled as an instance of the TSP. For further details on this application refer to McCallum [1986], Crama *et al* [1989].

There are many other important applications of the TSP. For further details, refer to Lenstra [1974], Lenstra & Rinnoy Kan [1975] and Lenstra [1976] and Garfinkel [1985].

1.5 Formulations of the STSP

In this section we give the well known integer programming formulations of the STSP. Consider a complete undirected graph $G=(N,A)$, where N is the set of n nodes and A is the set of m arcs. Let c_l be the cost of arc l. We sometimes use (i,j) to denote an arc (whose terminal nodes are i and j) and c_{ij} to denote its cost.

Let $x_l = 1$ if arc l is in the solution

$= 0$ otherwise.

The STSP can then be formulated as:

Problem P_{STSP} :

$$\min_x \sum_{l=1}^{m} c_l x_l \tag{1.1}$$

$$\text{s.t.} \sum_{l=1}^{m} x_l = n, \tag{1.2}$$

$$\sum_{l \in K_t} x_l \geq 2 \qquad \forall\ K_t \equiv (S_t, \bar{S}_t),\ S_t \subset N, \tag{1.3}$$

$$x_l \in \{0,1\}, \qquad l = 1,\ldots,m. \tag{1.4}$$

Here, $\bar{S}_t = N \setminus S_t$ and $K_t \equiv (S_t, \bar{S}_t)$ is the *cutset* of arcs in G. If i_l and j_l denote the terminal nodes of an arc l, then $K_t = \left\{ l \in A \,\middle|\, i_l \in S_t,\ j_l \in \bar{S}_t \right\}$.

Constraints (1.2) and (1.3) are equivalent to the following two sets of constraints:

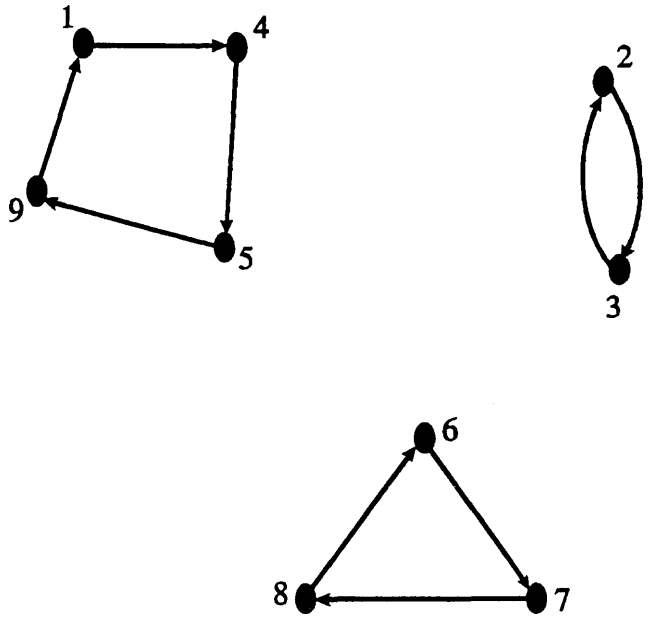
$$\sum_{l \in K_t} x_l \geq 1, \qquad \forall\ K_t \equiv \left(S_t, \bar{S}_t\right),\ S_t \subset N, \tag{1.5}$$

$$\sum_{l \in A_i} x_l = 2, \qquad i = 1, \ldots, n. \tag{1.6}$$

Here, A_i is the set of arcs incident at node i. The inequalities (1.5) represent the *subtour elimination* constraints, while those in (1.6) are the *degree constraints* which constrain the degree of each node, $i \in N$, to be equal to 2.

The subtour elimination constraints prevent the formation of *subtours* through subsets of the nodes. Such subtours in which the degree of each node is equal to 2 would result if these constraints are ignored (see Figure 1.1).

Figure 1.1 An example with subtours



If we use the notation (i,j) to denote an arc connecting nodes i and j, a compact formulation for the STSP is given by:

Problem P^{ij}_{STSP} :

$$\min_{x} \quad \sum_{i \in N} \sum_{j>i} c_{ij} x_{ij} \tag{1.7}$$

$$\text{s.t.} \quad \sum_{j<i} x_{ji} + \sum_{j>i} x_{ij} = 2, \qquad \forall\ i \in N, \tag{1.8}$$

$$\sum_{i \in S_t} \sum_{\substack{j \in S_t \\ j>i}} x_{ij} \leq |S_t| - 1, \qquad \forall\ S_t \subset N, \tag{1.9}$$

$$x_{ij} \in \{0,1\}, \qquad i,j \in N,\ j>i. \tag{1.10}$$

The subtour elimination constraints, (1.9) can also be expressed as follows:-

$$\sum_{i \in S_t} \sum_{\substack{j \in \bar{S}_t \\ j>i}} x_{ij} + \sum_{i \in \bar{S}_t} \sum_{\substack{j \in S_t \\ j>i}} x_{ij} \geq 2, \qquad \forall\ S_t \subset N. \tag{1.11}$$

1.6 Algorithms for the TSP based on branch and bound

Most of the exact algorithms for solving the TSP are of a branch and bound, enumerative type. These algorithms are classified according to the relaxation that is used to derive the lower bounds. For a thorough treatment, refer to Balas & Toth [1985] or Christofides [1979].

1.6.1 Bounds from the shortest spanning tree

Spanning tree based approaches for the STSP are derived by relaxing the degree constraints in problem P_{STSP} . We investigate this relaxation in great depth in Chapter 2 of this dissertation. This relaxation was first used for the STSP by Held & Karp [1970, 1971] and, simultaneously, by Christofides [1970]. The ideas that resulted out of the implementation of this relaxation for the STSP were instrumental in initiating the extensive use of lagrangean relaxation for combinatorial optimization problems. Subsequently, this relaxation has been applied for the STSP in effective algorithms by Helbig, Hansen & Krarup [1974], Smith & Thompson [1977], Volgenant & Jonker [1982] and Gavish & Srikanth [1983].

The branching rules in algorithms based on this relaxation generate subproblems in the tree search by fixing in (or out) of the TSP solution, arcs of the spanning tree incident at a node whose degree in the spanning tree is not equal to 2. Gavish & Srikanth [1983] select arcs of the spanning tree to be fixed in (or out) at a node of the tree search, such that the penalty for admitting the chosen arc into the TSP solution is maximal. Two subproblems are created that admit (alternatively, forbid) this arc. The computational results of Volgenant & Jonker [1982] indicate that a depth first branching strategy is more suitable for spanning tree based approaches.

Gavish & Srikanth [1983] exploit the spanning tree solutions generated during the tree search (that are infeasible for the STSP) to obtain upper bounds, using what they call an *imbedded heuristic*. In their approach, a series of chains is first obtained by removing from the spanning tree, arcs incident at nodes whose degree is greater than 2. These chains are connected to form a tour which is then improved using the

interchange heuristic of Lin & Kernighan [1973]. They report very tight upper bounds which enable many of the subproblems of the tree search to be discarded. Upper bounds are also obtained in the spanning tree based algorithm of Volgenant & Jonker [1982]. They use simplified forms of the heuristics of Christofides [1976] and Lin [1965] to obtain tight upper bounds.

Both Gavish & Srikanth [1983] and Volgenant & Jonker [1982] use problem reduction procedures based on sensitivity analysis. This procedure - termed by the latter as *branch chord exchanges* - involves the forcing of variables into (or out of) the problem.

Computational experience (see Balas & Toth [1985]) indicates that the most efficient branch and bound algorithms for the STSP are the spanning tree based approaches of Smith & Thompson [1977], Volgenant & Jonker [1982] and Gavish & Srikanth [1983]. Problems of up to 100 nodes can be solved using these algorithms in a reasonable amount of computing time. The main advantage of this approach is that spanning trees over each of the subproblems can be computed relatively easily using the efficient algorithms of Dijkstra [1959], Prim [1957], or Kruskal [1956]. The algorithms of Dijkstra and Prim have running time complexity of $O(n^2)$ while the algorithm of Kruskal is of complexity $O(\,|A|\log|A|\,)$, where $|A|$ is the number of arcs in the graph. In addition to the ease of computation, the lower bounds produced by the spanning tree based approaches are quite tight (normally to within 0.5% of the optimal STSP solution). Thus, this approach works very well for the STSP.

1.6.2 Bounds from the shortest spanning arborescence

The shortest spanning arborescence is used for directed graphs (ATSP) in a similar manner as the spanning tree is used for the STSP. Spanning arborescences can be constructed using the efficient algorithms of Edmonds [1967], Fulkerson [1974] or the algorithm of Tarjan [1977] which has a complexity of $O(n^2)$. However, the quality of bounds produced by this relaxation for the ATSP is, on average, poor when compared to the quality of the spanning tree bounds for the STSP. This major shortcoming makes procedures for the ATSP based on this relaxation inferior (see Smith [1975] and Hong [1978]) in comparison to those that use the assignment problem relaxation.

1.6.3 Bounds from the assignment problem

The assignment problem (AP) is the most direct and frequently used relaxation for the TSP. The removal of the subtour elimination constraints in the TSP formulation yields an assignment problem with the same cost function as the TSP. The first AP based approaches of this kind were those of Eastman [1958], and Shapiro [1966]. The term branch and bound evolved due to the algorithm of Little, Murty, Sweeny & Karel [1963] in which an assignment type relaxation is used for the TSP. Subsequently, improved AP based algorithms have been developed by Bellmore & Malone [1971], Smith, Srinivasan & Thompson [1977] and Carpaneto & Toth [1980].

The AP with the TSP cost function is a strong relaxation for the ATSP. Lower bounds are, on average, within 1% of the TSP optimal solution (see Balas & Toth

[1985]). However, this approach is weak for the STSP because of the prohibitively large number of subtours contained in the AP solution on graphs with symmetric costs. Jonker, De Leve, Van der Velde & Volgenant [1980] suggest a method to improve the AP lower bounds for the STSP which also attempts to reduce the number of subtours in the AP solution. They achieve this by transforming the distance matrix of the symmetric problem into an equivalent partially asymmetric one. However, even this transformation does not make the AP relaxation attractive for the STSP.

The branching rules used in conjunction with the AP relaxation are normally based on subtour breaking, or subtour patching inequalities. An efficient implementation of these rules is given by Smith, Srinivasan & Thompson [1977] who use a depth first branching scheme. Carpaneto & Toth [1980] also give a very efficient implementation of such branching rules along with a breadth first branching scheme.

The AP can be solved in at most $O(n^3)$ steps by the Hungarian algorithm (see Kuhn [1955], Christofides [1975] or Lawler [1976]). Typically, during the branch and bound, the introduction of branch constraints based on the AP solution of the problem at the node i of the tree search (say AP_i) necessitates the solution of an AP at a successor tree node j (say AP_j) in which an arc in the optimal solution of AP_i has to be excluded from AP_j. The solution to AP_i can be used to solve AP_j with the help of a simplified Hungarian method requiring at most $O(n^2)$ steps - see Bellmore & Malone [1971].

The violated subtour elimination constraints in the AP solution can be introduced into the objective function of the TSP in a lagrangean manner as in the case of the spanning tree. We can then solve the AP with a lagrangean objective function.

However, due to the large number of subtour elimination constraints (which increase exponentially with n), the number of multipliers in the lagrangean objective function increases exponentially rather than linearly with n. Balas & Christofides [1981] introduced a polynomially bounded procedure for approximating the maximum of a *restricted* lagrangean dual problem. Bounds obtained through this procedure were found to be, on average, less than 0.5% from the optimal (see Christofides [1979]). This approach is explained in greater detail in Chapter 3 of this dissertation.

1.6.4 Other approaches

The efforts of Fischetti & Toth [1988b] have led to a new solution approach called *additive bounding approach* for combinatorial optimization problems. This is a composite procedure that takes into consideration many different substructures that may be identified in a problem. Each of these substructures constitute a bounding procedure. The algorithm then delivers a sequence of increasing lower bounds (for a minimization problem) by considering the *residual costs* of a bounding procedure at one step as the input cost matrix for the bounding procedure used at the immediately subsequent step. This procedure is quite similar, in principle, to the procedure for obtaining bounds for the ATSP suggested by Christofides [1972] and also the restricted lagrangean approach of Balas & Christofides [1981]. The difference being that, while the earlier efforts use only the *reduced costs* from the assignment problem to obtain an increasing lower bound sequence, the additive bounding approach incorporates several bounding procedures sequentially. The additive approach has been applied to the ATSP (Fischetti & Toth [1988c]), the multiple depot

vehicle scheduling problem (Carpaneto, Dell'Amico, Fischetti & Toth [1988]), to the Prize Collecting TSP (Fischetti & Toth [1988a]) and to the STSP (Carpaneto, Fischetti & Toth [1989]).

The 2-matching relaxation for the STSP is analogous to the AP relaxation for the ATSP. Bellmore & Malone [1971] use it for the STSP in a similar manner to their use of the AP in their algorithm for the ATSP. The violated subtour elimination constraints can be introduced into the objective function and the resulting lagrangean dual problem can be solved using the restricted lagrangean approach of Balas & Christofides [1981]. However, the major obstacle to the use of this relaxation for the STSP has been the lack of an efficient implementation of a 2-matching algorithm (see Balas & Toth [1985]) although the problem is polynomially solvable (see Edmonds [1965b]).

The n-path problem as a relaxation of the TSP has been used by Houck, Picard, Queyranne & Vemuganti [1980]. Computational experience (Christofides [1979]) has indicated that the bound obtained by this method for the STSP is considerably weaker than those obtained from the spanning tree. It is, however, comparable to the bounds from the spanning arborescence for the ATSP. Furthermore, the complexity of algorithms for the spanning tree and arborescence is less than that of the dynamic programming algorithm for solving the n-path problem, which takes $O(n^3)$ steps. Thus, this relaxation for the TSP is only used when side constraints (like time windows and precedences) are added to the pure TSP as the extra constraints can be accommodated easily by the dynamic programming algorithm to solve the n-path problem.

Recently, there have been attempts to design algorithms for the solution of large-sized problems which can exploit the increasingly large computational power made available through new technology. An example is the design of both exact as well as heuristic parallel algorithms for parallel computers (as opposed to sequential algorithms on sequential computers). The algorithms of Mohan [1983] and Pekny & Miller [1988] are reasonably good implementations of parallel algorithms. The latter are able to solve ATSPs of size 3000 nodes with using the parallel version of an AP algorithm.

1.7 Linear programming based approaches

The linear programming relaxation of the TSP is one in which the integrality constraints are removed. In addition, the subtour elimination constraints are relaxed. The optimal TSP tour is then obtained by a cutting plane approach combined with (if necessary) a branch and bound procedure. The seminal paper in this regard is that of Dantzig, Fulkerson & Johnson [1954]. Miliotis [1976, 1978], Land [1979] and Fleischmann [1982] also used the LP relaxation, combined with the use of cutting planes, to solve medium-sized TSPs. However, the main drawback to approaches which incorporate cutting planes is that the integrality of the solution is not guaranteed at each stage of the procedure.

A major advance in the use of LP based TSP algorithms was due to Grötschel [1977, 1980] who developed linear characterizations of the TSP polytope. More of these inequalities, known as *facet-inducing* linear inequalities were introduced by Grötschel & Padberg [1979a, 1979b]. These inequalities, known as valid inequalities, are the

strongest possible cutting planes. They have been used by Grötschel [1980], Padberg & Hong [1980] and, most successfully, by Crowder & Padberg [1980], in conjunction with a branch and bound type of procedure to solve large-sized TSPs. All these procedures make use of sophisticated techniques to identify facet inducing inequalities. When no more facets can be identified, branching is performed. A recent advance in the development of LP based algorithms is the polyhedral approach of Padberg & Rinaldi [1987] who solve large instances of STSPs using a *branch and cut* algorithm. Their successes strongly indicate that polyhedral approaches could be the way forward to solving large instances of combinatorial optimization problems. A major reference in the use of LP based approaches for the TSP is Grötschel & Padberg [1985] and Padberg & Grötschel [1985], while Crowder, Johnson & Padberg [1983] review the use of these methods to solve general combinatorial optimization problems.

1.8 Heuristic algorithms

The TSP has been the focus of extensive research into the analysis and design of heuristics for combinatorial optimization problems. Many of the efficient heuristic approaches currently in use evolved from applications made for the TSP. The sheer number of heuristics for the TSP makes it impossible to review **all** such approaches in this section. We will, therefore restrict ourselves by summarizing the most efficient

heuristics for the TSP. In particular, we concentrate on the algorithms for the STSP and the *Euclidean* TSP[2].

Much of the research attention in heuristics for the TSP has been focused on algorithms for the STSP and the Euclidean TSP. The development of heuristics for the ATSP is still in a relative state of infancy when compared to the advanced levels of effectiveness that are possible by heuristics for the STSP. Akl [1980] proposed an algorithm for the ATSP based on the minimum spanning digraph. The savings method normally used in heuristics for vehicle routing instances has also been used for the ATSP (by Golden [1977], Frieze, Galbiati & Maffioli [1982], Golden, Bodin, Doyle & Stewart [1980]) has also been. Karp [1979] suggested a patching algorithm that uses assignment subtours to get upper bounds for the ATSP.

Major surveys on heuristics for the TSP are those of Rosenkrantz, Stearns & Lewis [1977], Golden, Bodin, Doyle & Stewart [1980], Adrabinski & Syslo [1983] and Golden & Stewart [1985]. Heuristics for the TSP can be classified into three major groups: *tour construction* procedures, *tour improvement* procedures and *composite* procedures.

Tour construction procedures commence with some initial subtour or starting point. A feasible TSP solution is constructed by inserting unvisited points into the tour being constructed in a sequential manner. The main components of this approach are: (a) the choice of the initial subtour (or starting point); (b) selection criterion which

[2]The *Euclidean* TSP is the problem of finding a tour of minimal length through n given *points* distributed anywhere in d-dimensional Euclidean space. The *distance* between any two points is given by the Euclidean distance metric.

decides on the point to be inserted next (eg. the nearest neighbour); (c) insertion criterion which decides on the position for the selected point in the growing tour (eg. the cheapest insertion rule). The savings algorithm of Clarke & Wright [1964] is an example of a tour construction procedure. Heuristics based on nearest addition, nearest insertion, cheapest insertion, or a farthest insertion principle belong to this class. Prominent heuristics belonging to this class are the arbitrary insertion algorithm of Rosenkrantz, Stearns & Lewis [1977], the convex hull insertion algorithm of Stewart [1977], and the greatest angle insertion procedure of Norback & Love [1977, 1979].

Tour improvement procedures improve some feasible initial tour through arc exchanges. This initial tour can even be a random tour in some cases. In a general step of the algorithm r arcs of the feasible tour are exchanged with r arcs not in the tour if: (a) the result is also a feasible tour, and (b) the length of the new tour is less than that of the previous tour. This is referred to as an *r-opt* procedure (Lin [1965]), where r refers to the number of arcs considered for exchange at each step. Although quite simple, 2-opt or 3-opt procedures produce good solutions in most cases. The most successful algorithm of this class is the *variable r-opt* algorithm of Lin & Kernighan [1973]. In their algorithm, the best value of r is decided dynamically at each iteration. Although it requires greater computational effort to code than either a 2-opt or a 3-opt procedure, this heuristic produces very good solutions that tend to be near-optimal. Or [1976] suggested a modified 3-opt procedure that also performs very well. For a description of this procedure, see Golden & Stewart [1985]. The ideas of this modified 3-opt approach are applied in an algorithm described in Chapter 2 of this dissertation.

Composite procedures begin with a tour construction approach and improve the resulting tour using a tour improvement approach.

There are, however, several specialized heuristic algorithms for the TSP that do not fall into any of the classes described above. Christofides [1976] developed a heuristic which combines a minimal spanning tree with a perfect matching on its odd degree nodes. A tour, known as an *Eulerian tour*, is obtained which may contain some nodes more than once. A feasible TSP tour is obtained from the Eulerian tour by deleting from it nodes that have already been encountered. This algorithm has a worst-case performance guarantee of $\frac{3}{2}$ times the optimal TSP solution. We describe this algorithm in greater detail in Chapter 2.

Karp [1977] gave a partitioning procedure for the Euclidean TSP which, for every $\varepsilon > 0$, produces a tour of cost not greater than $(1 + \varepsilon)$ times the optimal TSP solution with probability that tends to 1 as $n \to \infty$. Furthermore, the algorithm runs in time $k(\varepsilon)n + O(n \log n)$, where $k(\varepsilon)$ is a constant dependant on the choice of ε.

Kirkpatrick, Gelatt & Vecchi [1982, 1983] developed a novel tour improvement approach called *simulated annealing* which exploits the analogy between combinatorial optimization problems (like the TSP) and the statistical mechanics of large physical systems. This approach has been the subject of extensive research recently. The method (named simulated annealing because of its analogy to the physical phenomenon of annealing) is based on probabilistically accepting intermediate increases in the solution value - produced by the tour improvement steps - through a set of user-controlled parameters. For further details, see Golden & Skiscim [1986],

Bonomi & Lutton [1984]. For a general overview of simulated annealing, refer to Van Laarhoven & Aarts [1987] and Aarts & Korst [1988].

Bartholdi & Platzman [1983] apply the geometry of *fractals* to develop a scheme for solving combinatorial optimization problems. They apply it to design an $O(n \log n)$ algorithm for the Euclidean TSP (see Bartholdi & Platzman [1982] and Platzman & Bartholdi [1989]). To date, however, no empirical analysis on the performance of fractal-based TSP algorithms is available in the literature. In Chapter 5, we develop an algorithm based on a fractal curve that incorporates improvement ideas to get reasonably good solutions to very large-sized problems.

1.9 A note on the implementation of algorithms

In this section, we lay the foundations for the analysis of the algorithms that will be developed in the remaining chapters of this dissertation. In many cases, there may be a wide gap between the statement of an algorithm and its (computer) implementation. Some of the constructs of implementation could be the "best" algorithms to solve subproblems encountered; the "best" values for parameters like step-size, the number of iterations, and so on. We will try and explicitly outline all implementational aspects in an attempt to narrow the gap between the statement and the implementation of our algorithms.

Often, what is "best" is declared equivalent to what is "fastest". This again is dependent on the "best" algorithms (and codes) for subproblems; the "best" data structures; programming competence and so on. Moreover, codes can be made to run

faster by modifying them to suit a specific implementation on a specific computer and compiler. Wherever possible, the best codes, data structures and algorithms are used in our algorithms. However, we consider such implementational aspects to be beyond the scope of this research and therefore, question the sagacity of comparing algorithms merely on the basis of running time. However, we perform such comparisons and provide running times whenever we feel that a realistic and proper analysis is possible. In the absence of this, we use other methods of comparison (like number of iterations, number of trees, number of nodes of the tree search, and so on).

The test data used to evaluate the performance of our algorithms come from different sources. Table 1.1 provides a list of these problems along with their sources and optimal solution values. Other problems that are used will be referred to wherever appropriate. We have generated 15 Euclidean problems - 5 problems each with n=50, n=65, n=75, and n=100. The xy-coordinates for these problems have been drawn randomly generated on the integers $1, \ldots, 1000$. A listing of the coordinates for these problems, named KC500-KC504, KC650-KC654, KC750-KC754, and KC1000-KC1004 is provided in Appendix 1.1. The optimal solution for each of these problems is also given therein. The cost matrices for each of these problems were calculated using the Euclidean metric, rounded up to the nearest integer.

Table 1 - Description of some well-known test problems from the literature

Problem Name	Size n	Problem Type	Optimal Solution	Source
DF42	42	Road map	699	Karg & Thompson [1964]
HK48	48	Road map	11461	Held & Karp [1962]
GR48	48	Road map	5046	GrÖtschel [1977]
ST481	48	Euclidean	9729	Smith & Thompson [1977]
ST482	48	Euclidean	10680	Smith & Thompson [1977]
ST483	48	Euclidean	10180	Smith & Thompson [1977]
ST484	48	Euclidean	9984	Smith & Thompson [1977]
ST485	48	Euclidean	9844	Smith & Thompson [1977]
KT57	57	Road map	12955	Karg & Thompson [1964]
ST600	60	Euclidean	10374	Smith & Thompson [1977]
ST601	60	Euclidean	11703	Smith & Thompson [1977]
ST602	60	Euclidean	11777	Smith & Thompson [1977]
ST603	60	Euclidean	12699	Smith & Thompson [1977]
ST604	60	Euclidean	12497	Smith & Thompson [1977]
ST605	60	Euclidean	12262	Smith & Thompson [1977]
ST606	60	Euclidean	8073	Smith & Thompson [1977]
ST607	60	Euclidean	8553	Smith & Thompson [1977]
ST608	60	Euclidean	8903	Smith & Thompson [1977]
ST609	60	Euclidean	9156	Smith & Thompson [1977]
NCE50	50	Euclidean	427*	Christofides & Eilon [1969]
NCE75	75	Euclidean	536*	Christofides & Eilon [1969]
NCE100	100	Euclidean	632*	Christofides & Eilon [1969]
KF100A	100	Euclidean	21282	Krolak, Felts & Marble [1971]
KF100A	100	Euclidean	20749	Krolak, Felts & Marble [1971]
KF100A	100	Euclidean	22068	Krolak, Felts & Marble [1971]
GR120	120	Road map	6942	GrÖtschel [1980]
LK318	318	Euclidean	41345	Lin & Kernighan [1973]

* The *best known* values

CHAPTER 2

A Minimal Spanning Tree Relaxation of the Symmetric Travelling Salesman Problem

2.0 Outline

In this chapter we investigate the shortest spanning tree relaxation of the STSP. The relaxation approach was introduced by Held & Karp [1970] and, independently, by Christofides [1970]. We describe the basic ideas of the approach before reviewing the major improvements and modifications that have been suggested. We describe our relaxation procedure which incorporates some of the best ideas of the previous spanning tree based attempts. The original problem is considerably reduced by using sensitivity analysis techniques. These enable the identification of *superfluous* and *indispensable* arcs, those which respectively can be rejected from and forced into the optimal solution of the problem. Tests on the structure of the graph that defines the problem yield a greater number of superfluous and indispensable arcs. We also describe an upper bound algorithm that makes use of the spanning trees produced

during the procedure. This yields tight bounds at a reasonably low computational cost.

2.1 Definitions

In this section we introduce some notations and definitions; some of these have previously been defined and used by Smith & Thompson [1977].

Consider a complete directed graph $G=(N,A)$ where N is the set of n nodes and A is the set of m arcs. Let c_l denote the cost of arc l $(l=1,\ldots,m)$. We also sometimes denote the arc between two nodes i and j as (i,j) and its cost as c_{ij}. A *cycle* in G is a connected subgraph of G in which exactly two arcs are incident at every node. A *hamiltonian cycle*, H of G is a connected subgraph of G in which exactly two arcs are incident at each node. The TSP is therefore the problem of finding the minimum cost hamiltonian cycle, $\hat{H}$ in G.

A *spanning tree* in G is defined as a connected subgraph of G without any cycles. Of the many spanning trees of G that are possible, the *shortest spanning tree* (SST) or the *minimal spanning tree* (MST) is a subgraph, denoted by $G_T=(N,A_T)$, for which the sum of the costs of its arcs is minimum. We will denote an SST simply by $T=(N,A_T)$, where $A_T \subset A$. The number of arcs of an SST incident to a node i is called the *degree* d_i^T of i or simply d_i. Let $V(T)$ denote the length of the SST.

An arc l of G is called a *branch* if $l \in T$ and it is called a *chord* if $l \notin T$. The *fundamental path* of a chord l is the unique path of branches of T between the nodes i_l

and j_l, where i_l and j_l denote the terminal nodes associated with chord l. For two non-trivial and disjoint subsets S and $\bar{S}$ $(=N\backslash S)$ of N, a *cutset* of arcs in G is defined by $\left\{ l \middle| i_l \in S,\ j_l \in \bar{S} \right\}$. The *fundamental cutset* of a branch l is the cutset of the two connected subgraphs of G that remain when l is removed from T.

Using the above definitions, we can state the following necessary and sufficient conditions for a spanning tree to be minimal (see Ford & Fulkerson [1962], p. 175):

MSC1: Every branch of the tree is at least as short as any chord in its fundamental cutset.

MSC2: Every chord of the tree is at least as long as any branch on its fundamental path.

Define G_s as a subgraph of G with the node set $N_s=N\backslash\{s\}$. A *minimal spanning s-tree* (MSsT), $T_s=\left(N_s,\ A_{T_s}\right)$ of G consists of an SST on G_s combined with the two shortest arcs incident to node s. These are referred to simply as *minimal s-trees* or *s-trees* and the node s is called the *root node.*

An arc is said to be *superfluous* if it cannot be a member of any optimal solution to the STSP on G. An arc is said to be *indispensable* if it must be a member of every optimal solution. An arc is *available* if it is indispensable or if it has not been identified as superfluous. A superfluous arc l is *rejected* from G by increasing c_l by a suitably large number. An indispensable arc l is *forced into* the set of arcs that defines the optimal tour by decreasing c_l by a suitably large number.

Several algorithms for solving the SST are available. The more efficient ones are those of Kruskal [1956], Prim [1957], and Dijkstra [1959]. The algorithms of Prim

and Dijkstra have $O(n^2)$ complexity while the algorithm of Kruskal is of complexity $O(|A| \log |A|)$, where the term $\log |A|$ is due to a sorting that has to be performed on A available arcs. Thus, in a complete graph the algorithms of Prim and Dijkstra are more efficient (see Kreshenbaum & Van Slyke [1972]), whilst, when $|A|$ is reasonably small, Kruskal's algorithm is more efficient.

2.2 The shortest spanning tree Relaxation A

Consider an arc $l \equiv (i_l, j_l)$, where $l \in A$, $l \in \hat{H}$. Removal of this arc from the optimal tour leaves a single path of n-1 arcs through all the nodes in N starting at i_l and ending at j_l. Since $V(T)$, the cost of the SST is a lower bound on the cost of this path, $V(T) + c_l$ forms a lower bound on the cost of the optimal STSP solution.

Consider the formulation of the P_{STSP} given by (1.1), (1.4), (1.5) and (1.6).

Let $y_l = 1$ if l is the longest arc in the STSP optimal solution,

$= 0$ otherwise.

Set $x_l = 0$ if $y_l = 1$.

Problem STSP can then be reformulated as:

Problem P^{T}_{STSP} :

$$\min_{x,y} \quad \sum_{l=1}^{m} c_l x_l + \sum_{l=1}^{m} c_l y_l \tag{2.1}$$

$$\text{s.t.} \quad \sum_{l \in A_i} \left(x_l + y_l \right) = 2 , \qquad i = 1 , \ldots , n , \tag{2.2}$$

$$\sum_{l \in K_t} x_l \geq 1 , \qquad \forall \; K_t \equiv \left(S_t , \bar{S}_t \right) , \; S_t \subset N , \tag{2.3}$$

$$\sum_{l=1}^{m} y_l = 1 , \tag{2.4}$$

$$x_l , y_l \in \{0,1\} , \qquad l = 1 , \ldots , m . \tag{2.5}$$

We observe that problem P^{T}_{STSP} is equivalent to problem P_{STSP}. We also note that the solution to problem P^{T}_{STSP} given by $x_l \; (l = 1 , \ldots , m)$ forms a spanning tree of graph *G*. We use this formulation of the STSP to obtain lower bounds from the spanning tree.

2.2.1 Lagrangean bounds from Relaxation A.

By relaxing the degree constraints (2.2) in the formulation P_{STSP}, we obtain the formulation of a relaxed problem which can be defined as follows:

Problem P_{T_L}:

$$\min_{x,y} \quad \sum_{l=1}^{m} c_l x_l + \sum_{l=1}^{m} c_l y_l \tag{2.6}$$

s.t. (2.3) - (2.5).

The solution to problem P_{T_L} given by $x_l \ (l = 1, \ldots, m)$ defines the SST whose cost is $V(T)$. Let L be the cost of the longest arc in the TSP solution. Then, $Z(P_{T_L}) = V(T) + L$ is a lower bound on $Z(P_{STSP})$. A better bound on the value of P_{STSP} can be obtained by including the relaxed degree constraints in the objective function of P_{T_L} in a lagrangean manner with the help of multipliers, $\pi = \{\pi_i \mid i \in N\}$; that is, by solving the problem:

Problem Relaxation A:

$$\min_{x,y} \quad \sum_{l=1}^{m} c_l x_l + \sum_{l=1}^{m} c_l y_l + \sum_{i \in N} \pi_i \left(\sum_{l \in A_i} x_l + y_l - 2 \right) \tag{2.7}$$

s.t. (2.3) - (2.5).

We note that if an arc l is included in T, the degrees of the terminal nodes associated with arc l are increased by one. Thus, the contribution of arc l to the total cost is: $(c_l + \pi_{i_l} + \pi_{j_l})$. The objective function (2.7) is therefore re-written to obtain the following formulation for the relaxed problem.

Problem $P_{T_L}(\pi)$:

$$\min_{x,y} \quad \sum_{l=1}^{m} \left(c_l + \pi_{i_l} + \pi_{j_l} \right) x_l + \sum_{l=1}^{m} \left(c_l + \pi_{i_l} + \pi_{j_l} \right) y_l - 2 \sum_{i=1}^{n} \pi_i \qquad (2.8)$$

s.t. (2.3) - (2.5).

Problem $P_{T_L}(\pi)$ can be decomposed into two independent problems:

Problem $P_T(\pi)$:

$$\min_{x \in T} \quad \sum_{l=1}^{m} \left(c_l + \pi_{i_l} + \pi_{j_l} \right) x_l \qquad (2.9)$$

$$\text{s.t.} \quad \sum_{l \in K_t} x_l \geq 1 , \qquad \forall\ K_t \equiv \left(S_t , \bar{S}_t \right),\ S_t \subset N ,$$

$$x_l \in \{0,1\}, \text{ and}$$

Problem $P_L(\pi)$:

$$\min_{y} \quad \sum_{l=1}^{m} \left(c_l + \pi_{i_l} + \pi_{j_l} \right) y_l \qquad (2.10)$$

$$\text{s.t.} \quad \sum_{l=1}^{m} y_l = 1 ,$$

$$y_l \in \{0,1\}.$$

Problem $P_T(\pi)$ defines the SST on G with modified costs $c_l' = \left(c_l + \pi_{i_l} + \pi_{j_l} \right), \forall\ l \in A$ and $i,j \in N$.

Let $\hat{L}$ be a lower bound on L with respect to the modified costs c_l'. Problem $P_L(\pi)$ is solved by simply setting $y_l = 1$ for that arc l with the smallest modified cost $c_l' > \hat{L}$.

Identify arc k such that $k = \min_{j \in A} \left\{ c_j' \middle| c_j' \geq \hat{L} \right\}$.

We then set $$\left\{ \begin{array}{l} y_l = 1 \text{ for } l = k \\ \quad = 0 \text{ otherwise} \end{array} \right\} \tag{2.11}$$

Proposition 2.1 The longest arc in a TSP solution must be at least as large as the second smallest arc from any node $v \in N$.

This follows from the knowledge that the longest arc in the TSP solution is at least as long as the second shortest arc in any cut K_t of G, where $K_t \equiv \left(S_t, \bar{S}_t\right)$, $S_t \subset N$.

We use Proposition 2.1 to define $\hat{L}_{1v}$, a lower bound on L as:

$$\hat{L}_{1v} = \left\{ \min_{l \in A_v} \left\{ c_l' \right\} \right\} \tag{2.12}$$

By removing a node v and the two arcs incident at v from the TSP solution, we obtain a hamiltonian chain through the nodes in $N \backslash \{v\}$. Consider $S \subset N \backslash \{v\}$ and define $\bar{S} = N \backslash \{v\} \backslash S$. Every cut $\left(S, \bar{S}\right)$ must contain at least one arc of the TSP solution. We use this to define $\hat{L}_{2v}$, a lower bound on L as:

$$\hat{L}_{2v} = \max_{S \subset N \backslash \{v\}} \left\{ \min_{l \in \left(S, \bar{S}\right)} \left\{ c_l' \right\} \right\} \tag{2.13}$$

For a particular node v, if we set $\hat{L}_v = \max\left[\hat{L}_{1v}, \hat{L}_{2v}\right]$, the longest arc in the TSP solution must be at least as long as:

$$\hat{L} = \max_{v \in N} \left\{ \hat{L}_v \right\} \tag{2.14}$$

The computation of $\hat{L}$ can be carried out as follows:

Let T be the SST solution to problem $P_T(\pi)$ for a given π.

Let $V_1 = \left\{ i \middle| d_i = 1 \right\}$, and $\bar{V}_1 = N \backslash \left\{ v_1 \right\}$.

Consider $v \in V_1$. Since the least cost arc from A_v must belong to T, (2.12) can be re-written as:

$$\hat{L}_{1v} = \min_{l \in A_v} \left[c_l' \middle| l \notin T \right] \tag{2.15}$$

Set $\hat{L}_v = \hat{L}_{1v}$.

Consider $v \in \bar{V}_1$. By removing v from T, k distinct subtrees are formed, where $k = d_v$. Let T_v^i denote the set of points in the ith subtree and $T_v^{-i} = N \backslash \{v\} \backslash T_v^i$. (2.13) can be re-written as:

$$\hat{L}_{2v} = \max_{i=1,\ldots,k} \left\{ \min_{l \in \left(T_v^i, T_v^{-i}\right)} \left\{ c_l' \right\} \right\} \tag{2.16}$$

Set $\hat{L}_v = \hat{L}_{2v}$.

$\hat{L}$ is calculated using (2.14) and problem $P_L(\pi)$ is solved using (2.11).

2.2.2 Computing the bounds

For a given π, $Z\left(P_T(\pi)\right)$ and $Z\left(P_L(\pi)\right)$ represent the optimal values of problems $P_T(\pi)$ and $P_L(\pi)$ respectively. We showed in Section 2.2.1 that the value $Z\left(P_{T_L}(\pi)\right)$ forms a lower bound on $Z\left(P_{STSP}^T\right)$. From (2.8) it is clear that:

$$Z\left(P_{T_L}(\pi)\right) = Z\left(P_T(\pi)\right) + Z\left(P_L(\pi)\right) - 2\sum_{i=1}^{n} \pi_i \leq Z\left(P_{STSP}^T\right)$$

The greatest lower bound of this kind is obtained by deriving the set of multipliers π^* such that:

$$Z\left(P_{T_L}(\pi^*)\right) = \max_{\pi} Z\left(P_{T_L}(\pi)\right) \tag{2.17}$$

Problem (2.17) is known as the *lagrangean dual* of STSP and the vector π^* defines the dual variables associated with the relaxed degree constraints. Iterative *ascent methods* are used to obtain the vector π that maximizes (2.17). A general description of this method is provided in a later section.

In obtaining solutions to problem $P_L(\pi)$ during the ascent, we observe that it takes considerably more computational effort to determine $\hat{L}_{2v}$ than $\hat{L}_{1v}$. Hence, $\hat{L}$ is set to $\hat{L}_{1v}$, which is easier to evaluate, for most of the iterations. The tighter bound described by (2.14) is not always used. We evaluate the bound by defining a parameter ω. $\hat{L}_v$ at iteration m is as described by (2.14) if m is a multiple of ω. $\hat{L}_v$ is set to $\hat{L}_{1v}$ otherwise. Thus $\hat{L}_{2v}$, the more expensive, but tighter lower bound on L, is only evaluated every ω iterations. After considerable experimentation, we found that the best results were obtained by setting $\omega = \left\lceil \frac{n}{8} \right\rceil$.

We tested this approach to obtain lower bounds on a few test problems. We coded the ascent procedure of Smith & Thompson [1977] and compared the results obtained from the two approaches. For the problem DF42 our procedure obtains a lower bound of 696.994 after evaluating 105 trees in 45.912 Cpu Secs of computing time on a CDC-CYBER/930. The corresponding figures for the ascent of Smith & Thompson are (696.980, 105 and 21.678). For the problem HK48 the results for our procedure are (151, 11444.978, 64.714), while those for the Smith & Thompson approach is

(150, 11444.183, 31.776). For the problem ST605 the results for the two approaches are (180, 122.114, 12188.324) and (185, 61.117, 12187.324) respectively.

After analysis of the results obtained on many such test problems, we concluded that the best lower bound at the end of the ascent, obtained by our formulation of the SST, is comparable and, in some cases, superior to that produced by existing formulations of the lagrangean problem. However, this increased lower bound is achieved at a much greater computational cost. We therefore did not pursue this method further.

2.3 The shortest spanning tree Relaxation B

We observe that every optimal solution to the STSP is an s-tree. However, not all s-trees are travelling salesman tours. An s-tree T_s is an optimal TSP tour only if exactly two branches meet at each node in N. Hence, the STSP can be formulated as follows:

> Determine the minimal spanning s-tree of G with the additional restriction that $d_i = 2$, $\forall\ i = 1, \ldots, n$.

The problem of determining the s-tree on G is therefore a relaxation of STSP. If we remove node s and the arcs incident at s from $\hat{H}$, the remaining graph is a path, P, through the vertices in N_s. A minimal spanning tree on the graph G_s is a lower bound on the cost of P. The cost of the two arcs that are removed from $\hat{H}$ to produce P is at least as large as the cost of the two shortest arcs incident to node s. Hence,

the cost of the *s*-tree is a lower bound on the cost of the travelling salesman tour. The mathematical formulation of the problem whose solution is T_s is given below:

Problem P_{T_S}:

$$\min_{l} \quad \sum_{l=1}^{m} c_l x_l \tag{2.18}$$

$$\text{s.t.} \quad \sum_{l=1}^{m} x_l = n, \tag{2.19}$$

$$\sum_{l=1}^{m} x_l \geq 1, \qquad \forall\ K_t \equiv \left(S_t, \bar{S}_t\right),\ S_t \subset N, \tag{2.20}$$

$$\sum_{l \in A_s} x_l = 2, \tag{2.21}$$

$$x_l \in \{0,1\}. \tag{2.22}$$

Problem P_{T_S} is the same as problem P_{STSP} with constraints (1.3) replaced by constraints (1.5), and with the degree constraints (1.6) relaxed except for node *i*=*s*.

2.3.1 Lagrangean bounds from Relaxation B

To obtain the lagrangean problem, the relaxed degree constraints are included in the objective function of problem P_{T_S} by means of lagrange multipliers $\pi = \{\pi_i | i \in N\}$. The formulation of this problem, called Relaxation B is given below:

Problem Relaxation B:

$$\min_{x \in T_s} \quad \sum_{l=1}^{m} c_l x_l + \sum_{i \in N} \pi_i \left(\sum_{l \in A_i} x_l - 2 \right) \tag{2.23}$$

s.t. Constraints (2.19) to (2.22)

The objective function (2.23) is re-written to obtain the following formulation for the relaxed problem.

Problem $P_{T_S}(\pi)$:

$$\min_{x \in T_s} \quad \sum_{l=1}^{m} \left(c_l + \pi_{i_l} + \pi_{j_l} \right) x_l - 2 \sum_{i \in N} \pi_i \tag{2.24}$$

s.t. Constraints (2.2) to (2.5)

Problem $P_{T_S}(\pi)$ defines the MSsT on G where the arc lengths c_l are transformed to $c_l' = c_l + \pi_{i_l} + \pi_{j_l}$. For any set of multipliers π, the transformation of arc lengths may change the set of minimal trees. It does not, however, influence the set of optimal STSP solutions (see Held & Karp [1970]). The multipliers are also referred to as *node weights* or *node penalties*.

Let T_s^{π} be the minimal s-tree associated with the set of node weights π. Let d_i be the degree of node i ($i \in N$) in T_s^{π}. Since $d_i = 2$, $\forall\, i \in N$ in any optimal solution, the length of the minimal TSP tour in terms of the transformed weights is: $Z(P_{STSP}) + 2\sum_{i=1}^{n} \pi_i$. The length of the minimal s-tree is: $V(T_s) + \sum_{i=1}^{n} \pi_i d_i$. Since the length of the s-tree is a lower bound on the length of the minimal tour, we have:

$$V(T_s) + \sum_{i=1}^{n} \pi_i d_i \le Z(P_{STSP}) + 2\sum_{i=1}^{n} \pi_i$$

For the optimal tour on the original arc lengths, we have:

$$V(T_s) + \sum_{i=1}^{n} \pi_i (d_i - 2) \leq Z(P_{STSP}) \tag{2.25}$$

Since the LHS of (2.25) is the solution to problem $P_{T_s}(\pi)$, $Z(P_{T_s}(\pi))$ is still a lower bound on $Z(P_{STSP})$.

Let $f(\pi)$ denote the *gap* that exists between the cost of the optimal tour and the MSsT with respect to c_l'.

$$f(\pi) = Z(P_{STSP}) - Z(P_{T_s}(\pi)) \tag{2.26}$$

The problem of minimizing the gap in (2.26) is equivalent to finding the set of multipliers π^* such that:

$$Z(P_{T_s}(\pi^*)) = \max_{\pi} Z(P_{T_s}(\pi)) \tag{2.27}$$

Problem (2.27) is referred to as the *lagrangean dual* of the STSP and the gap $f(\pi^*)$ is referred to as the *duality gap*. For surveys of lagrangean relaxation see Geoffrion [1974], Fisher [1981] and Shapiro [1979].

Held and Karp [1970] proposed *ascent methods* to identify π^* that renders $Z(P_{T_s}(\pi^*))$ to be close to the optimal solution. A similar procedure was simultaneously suggested by Christofides [1970]. Subsequently, improved methods have been developed by Helbig, Hansen & Krarup [1974], Smith & Thompson [1977], Volgenant & Jonker [1982] and Gavish & Srikanth [1983]. Successful heuristic iterative procedures to maximize (2.27) have also been implemented by Camerini, Fratta & Maffioli [1975] and Held, Wolfe & Crowder [1974].

2.3.2 Ascent methods

The objective function of (2.27) is piecewise linear and concave in π. An ascent procedure, known in literature as *subgradient optimization*, is used to obtain its maxima. For surveys of the subgradient ascent method see Held, Wolfe & Crowder [1974], and Sandi [1979].

The ascent is an iterative procedure that commences with an initial lagrange multiplier vector $\pi = \pi^0$, normally taken to be the zero vector. Gavish & Srikanth [1983] use a data reduction procedure which is similar to the first stage of any standard algorithm to solve the assignment problem. We found that this choice generally ensures a modest improvement in the lower bound at the end of the ascent at minimal additional computational cost.

At any iteration m of the ascent, solve $P_{T_S}(\pi)$ for $\pi = \pi^m$. If the s-tree T_s^π, is a tour or if $Z\left(P_{T_S}(\pi)\right) \geq U$, the ascent is terminated. Here, U is an upper bound on the optimal tour length. Otherwise, let d_i^m be the degree of node i in T_s^π and let p be a positive scalar not greater than 2. The multipliers for iteration m+1 of the ascent are updated using:

$$\pi_i^{m+1} = \pi_i^m + t^m\left(d_i^m - 2\right), \qquad \forall\ i \in N \tag{2.28}$$

where t^m is the *step length* defined by:

$$t^m = p * \frac{U - Z\left(P_{T_S}(\pi^m)\right)}{\sum_{i \in N}\left(d_i^m - 2\right)^2} \tag{2.29}$$

This method converges if $\sum_{m=1}^{\infty} t^m = \infty$ and $\lim_{m \to \infty} t^m = 0$.

These conditions are satisfied if we start the ascent procedure with p=2 and reduce it systematically to 0. Since a polynomial-time convergence cannot be guaranteed, stopping rules enable the identification of π^e at the end of the ascent. π^e is normally a good approximation of π^*. Smith & Thompson [1977] developed an efficient implementation of such an ascent that results in tight lower bounds. Volgenant & Jonker [1982] report good results by using an updating formula and a formula for the step length that are different from (2.28) and (2.29) respectively. The implementation of our ascent is similar to that of Smith & Thompson [1977].

2.3.3 Choice of the root node

In almost all SST based approaches to the STSP the choice of the root node s in s-trees is taken, arbitrarily, to be node 1. Held & Karp [1970] suggested (without actual implementation) that better bounds might be achieved through a more accurate selection of the root node.

Jonker [1986] suggests several choices for the root node, including the selection of a *central* node or a node on the convex hull of nodes in G. Their conclusion is that although the lower bound for π=0 varies substantially, the differences in the lower bounds obtained at the end of the ascent are so small that the extra computational effort in selecting the special root node is unnecessary. Bazaraa & Goode [1977] choose node s after evaluating n distinct s-trees, each with a different node as the root. They choose s such that:

$$s = \left(k \;\middle|\; k \in N,\ Z\left(P_{T_k}\right) = \max_{i \in N} \left\{ Z\left(P_{T_i}\right) \right\} \right)$$

We observed that the results at the end of the ascent do not justify the considerably higher computational effort involved in such a selection.

Gavish & Srikanth [1983] obtain a minimal tree T initially for $\pi = \pi^0$ and choose s such that: $s = \max_{i \in N} d_i$. By forcing the degree of this node to be 2 in all s-trees, they report a consistent 0.05 to 0.15% improvement in the best lower bound obtained. Our ascent produces very tight initial lower bounds when we adopt this choice of s, coupled with: (i) a choice of π^0 different from 0, and, (ii) implementation of the ascent of Smith & Thompson [1977].

2.4 The upper bound heuristic

In Section 2.3.2 we defined U as an upper bound on the optimal solution value. Any good heuristic can be used to generate this bound. We use the variable r-opt procedure of Lin & Kernighan [1973] to get an initial value for U. These bounds are, on average, within 1.5% of the optimal solution.

We use the s-trees obtained during the ascent in an attempt to tighten this initial upper bound through the use of an imbedded heuristic. This heuristic, which is similar to that of Volgenant & Jonker [1982], improves the initial value of U in all the problems we tested it on. We found that for a large proportion of the well known road map and euclidean problems, the optimal solution is obtained. The algorithm we describe here

is a two-phase algorithm which is a combination of (simplified forms of) two well known heuristics:

(a) the Christofides heuristic, (b) Lin's 3-optimal algorithm.

2.4.1 Phase 1: Heuristic matching

Christofides [1976] developed a heuristic which has a worst-case performance bound of $\frac{3}{2}$ times the optimal tour length. We give a brief description of Christofides' algorithm after introducing the following definitions:-

- Given a set $N_M \subseteq N$, where $|N_M|$ is even, a *(perfect) matching*, M is a collection of arcs $A_M \subseteq A$, such that each node in N_M is the end-point of exactly one arc in A_M. A minimum weight matching M^*, is one in which the total cost of arcs is a minimum.
- An *Eulerian graph* is a connected graph in which every node has an even degree. An *Eulerian tour* is defined as a cycle in an Eulerian graph in which each arc is traversed only once.

Christofides' algorithm is then described as follows:-

Step 1: Construct a minimal spanning s-tree on G.

Step 2: Construct the set $N_M = \{ i \mid i \in N,\ d_i \text{ odd} \}$, the set of odd degree vertices in the tree. Find M^* for the set N_M.

Step 3: The Eulerian graph is given by $(A_{T_s} \cup A_{M^*})$. Construct the Eulerian tour.

Step 4: A feasible TSP solution is obtained by finding *short cuts* in the Eulerian tour. This is achieved by forbidding re-visits in the original Eulerian tour to nodes that have already been visited.

Figure 2.1 shows the above stages of the algorithm applied to a 9-nodes example. Figure 2.1(a) shows the *s*-tree with the odd degree vertices encircled. In the Eulerian graph of Figure 2.1(b), the thickened lines indicate arcs in the minimum weight matching. Since there is no unique traversal of the Eulerian graph, the Eulerian tour of Figure 2.1(c) is just one of many that are possible. The TSP tour of Figure 2.1(d) constructed from the Eulerian tour is also not unique because the sequence of nodes in the tour depends on the short cut rules that are used.

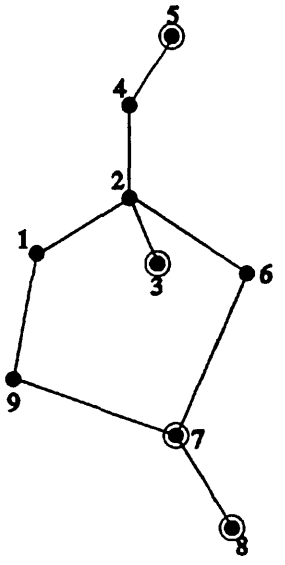
The following observations on Christofides' algorithm are immediate: (i) It can, conveniently, be imbedded into the subgradient ascent and applied to every *s*-tree obtained; (ii) The running time complexity of the algorithm is dominated by the minimum matching in Step 2 which can be found in time $O(n^3)$ - see Lawler [1976] or Papadimitriou & Steiglitz [1982].

For the imbedded heuristic we reduce the $O(n^3)$ complexity of the matching by merging Steps 2, 3, and 4 of the algorithm - we solve the minimum matching and find short cuts in the resulting Eulerian tours heuristically.

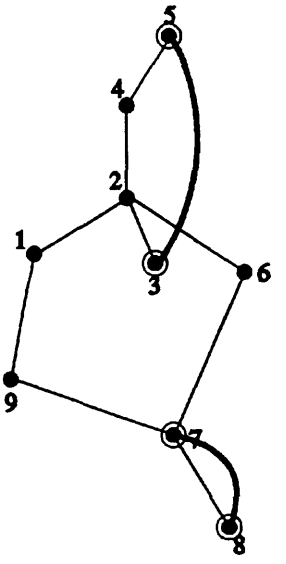
Let $c=\{p,\ldots,q\}$ define a path in the *s*-tree whose terminal nodes are p and q. c is called a *side branch* of the *s*-tree if $d_p=1$ and $d_q>2$. Define the set of nodes, $R=\left\{r \middle| r\in N,\ (r,q)\in T_s,\ r\notin c\right\}$. There must be at least two such nodes.

Choose a node r^* such that: $$r^*=\min_{r\in R}\left\{c_{rp}-c_{rq}\right\}$$

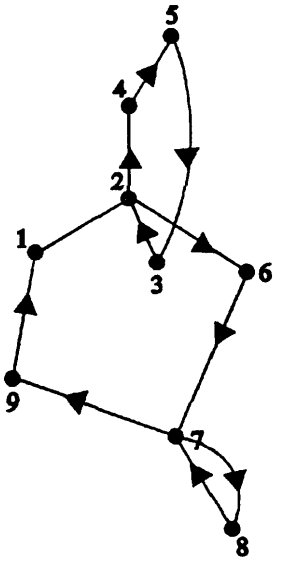
Figure 2.1 Stages of Christofides' Algorithm on a 9-nodes example



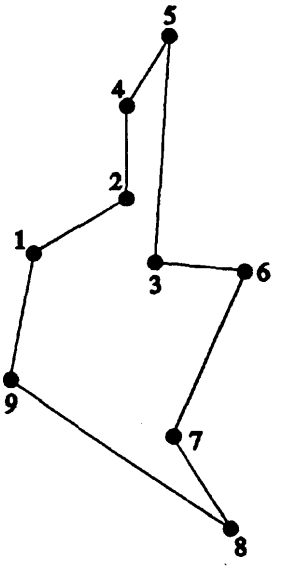
(a) Minimum Spanning 1-Tree (MS1T)



(b) Eulerian Graph: MS1T plus A Minimum Weight perfect Matching on odd-degree nodes



(c) Eulerian Tour: 1 2 4 5 3 2 6 7 8 7 9 1



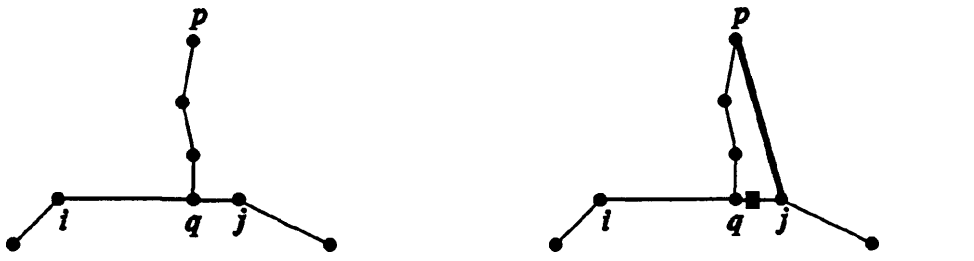
(d) TSP Tour: 1 2 3 4 5 6 7 8 9

Transformation of the side branch *c* (or, the transformation of *p* from a node of degree 1 to one of degree 2), into a part of a tour is possible through the deletion of arc

(r^*, q) from T_s, and the insertion of arc (r^*, p). The cost of this transformation of p from a node of degree 1 to one of degree 2 is: $C_p = c_{r^*p} - c_{r^*q}$. Let U_M be the upper bound obtained by this heuristic. U_M is first set to the value of the s-tree. After every transformation, the bound is updated using: $U_M \leftarrow U_M + C_p$. The procedure terminates when there are no nodes of degree 1 to transform; a feasible tour with upper bound U_M is obtained.

Figure 2.2 illustrates how the procedure is applied to convert a side branch, $c = (p, \ldots, q)$ to a part tour. i and j are the two nodes (not in c) connected to q in the tree, ie., $R = \{i, j\}$. Since, $(c_{jp} - c_{jq}) < (c_{ip} - c_{iq})$, arc (j, p) is inserted in place of (j, q) which is deleted.

Figure 2.2 A Heuristic Matching



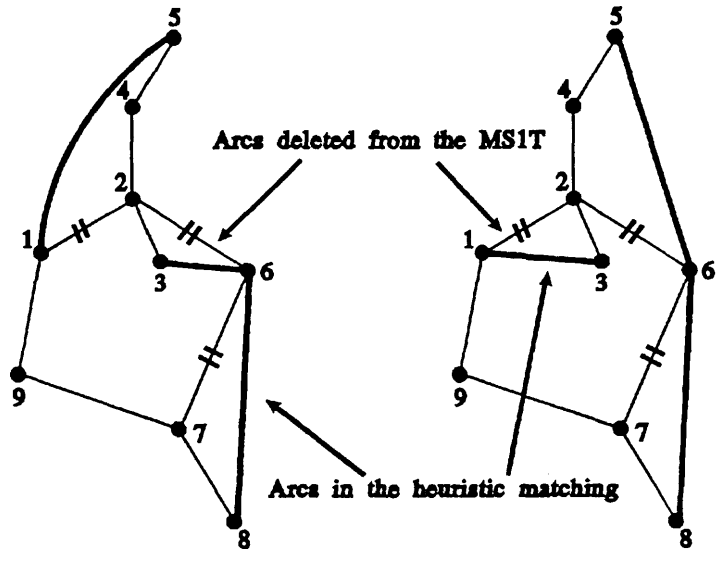
(a) A side-branch **(b) Converting a side-branch to a part tour**

An inexpensive conversion of a node with degree 1 into one of degree 2 depends on the length of the side branch of which it is a part. This, we observed, is in turn dependent on the order in which the nodes of degree 1 are chosen for conversion.

Hence, two passes of this procedure are performed: a *forward pass* (least node index first), and a *backward pass* (greatest node index first).

Figure 2.3 shows the different tours obtained through the two passes for the 9-nodes example of Figure 2.1(a).

Figure 2.3 Heuristic Matching on the 9-nodes example



(a) Tour from Pass-1:
1 5 4 2 3 6 8 7 9

(b) Tour from Pass-2:
1 3 2 4 5 6 8 7 9

This procedure can be employed to convert each *s*-tree in the ascent to a feasible tour. However, we only convert *s*-trees where the number of nodes with degree 1 is less than or equal to a parameter μ. μ is initialized to $\left\lceil \frac{n}{8} \right\rceil$ at the start of the ascent. As the number of ascent iterations increases, the *s*-trees are generally tour-like and approach TSP feasibility. Hence, to avoid excessive computation, μ is reduced by 1 after every 5 *s*-tree conversions until μ is 1.

The application of this procedure to identical trees would lead to the same feasible tour. In order to avoid potential repetitions of the procedure on identical trees that

could recur during the ascent, each s-tree that is already converted into a feasible tour is mapped into a single integer γ using the hashing function:

$$\gamma = \sum_{i \in N} i\left(d_i - 2\right)^2$$

For each s-tree that has nodes of degree 1 less than μ, the value γ is computed. If this value is present in the list of previously computed γ values, phase 1 of this heuristic procedure is not implemented. If the procedure is implemented, the values γ and U_M are recorded along with the feasible tour. It is probable that several s-trees could map onto the same value of γ. The only consequence, in this case, is that the heuristic will not be applied to some different s-trees.

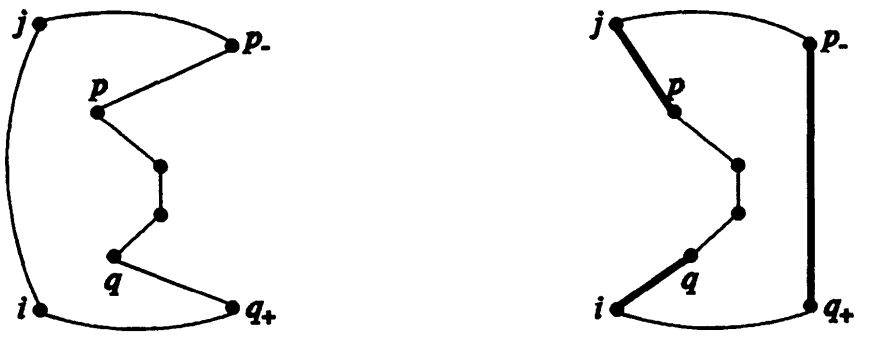
If the value of the upper bound at the end of phase 1 (ie., U_M) is less than a tolerance δ, we subject the associated feasible tour to phase 2 of our heuristic which we describe below. After considerable experimentation, we selected the value of δ as: $\delta = 1.06 * U_M^*$, where U_M^* is the value of the best upper bound obtained so far in the procedure.

2.4.2 Phase 2: Tour improvement through insertions

This is essentially an adaptation of the 3-Opt procedure of Lin [1965] and is based on the modification suggested by Or [1976] - see Golden & Stewart [1985]. We consider the exchanges that would accrue from inserting a chain of three, two, one or no arcs between an arc in a feasible tour obtained from phase 1. We thereby limit the number of 3-Opt exchanges and hence reduce the complexity of the procedure. This is because long chains of arcs in the s-tree are likely to appear unchanged in the optimal

solution and it is sufficient if we locally improve the transformations effected in phase 1 of our heuristic.

Figure 2.4 Example of tour improvement through insertions



(a) Current tour **(b) Improved tour**

Figure 2.4 illustrates how this procedure works for inserting a chain $c=\{p,\ldots,q\}$ of three arcs. Let Q denote the tour obtained from phase 1. We test to find the best insertion of this chain between all pairs of connected nodes i and j such that $(i,j)\in Q$ and $(i,j)\notin c$. If p_- is the point that precedes the chain c in the tour and, q_+ the point that succeeds it, then the insertion cost for inserting the chain of arcs from p to q between arc (i,j) is:

$$\delta_{ij}^{c}=\left(c_{p_-p}+c_{qq_+}+c_{ij}\right)-\left(c_{p_-q_+}+\min\left\{\left(c_{jq}+c_{pi}\right),\left(c_{jp}+c_{qi}\right)\right\}\right)$$

For a given chain, we find the best insertion: $\delta^{c}=\max_{\substack{(i,j)\in Q\\(i,j)\notin c}}\left\{\delta_{ij}^{c}\right\}$. If $\delta^{c}>0$, we apply the exchanges to the feasible tour and reduce the upper bound using $U_M \leftarrow U_M-\delta_{ij}^{c}$. If the updated value for U_M is less than the best upper bound U_M^*, then $U_M^* \leftarrow U_M$.

After considering all chains of 3 arcs, we consider chains of 2 arcs, and 1 arc and finally, chains of no arcs (insertion of nodes between an arc). The algorithm terminates when no further upper-bound-reducing insertions can be found. The tour improvement procedure involves far fewer exchanges than a regular 3-Opt procedure. Furthermore, the algorithm can be seen as equivalent to a 2-Opt procedure combined with insertions. Phase 1, the s-tree conversion is of linear time complexity, while phase 2 has complexity of $O(n^2)$.

2.5 Problem reduction

In this section we describe the two kinds of tests we carry out to induce graph sparsity. The first, called *Branch-Chord Exchanges*, is based on a sensitivity analysis of the tree at the end of the ascent. Tests on the structural properties of the graph result in further reductions in the size of the problem.

2.5.1 Branch-chord exchanges

At the end of the ascent we have, for $\pi = \pi^e$, the associated s-tree T_s^{π}. The procedure we describe here is for a general s-tree $(s \in N)$. It can be used for a 1-tree by substituting 1 for s. Also, in order to simplify the notation, we use T_s to denote the final s-tree $T_s^{\pi^e}$; π to denote π^e, the multipliers at the end of the ascent; $Z(T_s)$ to denote $Z\left(\mathrm{P}_{\mathrm{T_s}}(\pi^{\mathrm{e}})\right)$, the lower bound at the end of the ascent. We will also denote the cost of an arc l as c_l although this analysis will, at the end of the ascent, be used in conjunction with c_l', the modified costs.

Define: T_{s+ij} as the s-tree constructed from T_s by forcing chord (i,j) into it.

T_{s-kl} as the s-tree constructed from T_s by forcing branch (k,l) out of it.

We differentiate between two types of arcs: A_s, the set of arcs incident to node s; A_v $(v \neq s)$, the set of arcs not incident to node s. The results from the minimal spanning tree conditions MSC1 and MSC2 (Section 2.1) can, therefore, be defined as follows:-

(i) For a branch $(k,l) \in A_v$, T_{s-kl} is obtained from T_s by exchanging (k,l) with a chord (i,j) - the shortest chord in its fundamental cutset.

(ii) For a chord $(i,j) \in A_v$, T_{s+ij} is obtained from T_s by exchanging (i,j) with a branch (k,l) - the longest branch in its fundamental path.

(iii) Let (s,α) and (s,β) represent the two branches of T_s that are linked to node s such that $c_{s\alpha} \leq c_{s\beta}$. Identify the arc (s,γ) such that $c_{s\gamma} = \min_j \left\{ c_{sj} \middle| (s,j) \in A_s,\ j \neq \alpha,\beta \right\}$. $T_{s-s\alpha}$ (respectively, $T_{s-s\beta}$) is obtained from T_s by exchanging branch (s,α) (respectively, (s,β)) with chord (s,γ).

(iv) For a chord $(s,\gamma) \in A_s$, $T_{s+s\gamma}$ is obtained from T_s by exchanging chord (s,γ) with branch (s,β).

In general, a branch (k,l) is replaced in T_s by exchanging it with a chord (i,j) of cost c_{ij} to produce T_{s-kl}; a chord (i,j) is introduced into T_s by exchanging it with a branch (k,l) of cost c_{kl} to produce T_{s+ij}.

A branch (k,l) is indispensable if:

$$Z(T_{s-kl}) = Z(T_s) - c_{kl} + c_{ij} > U \tag{2.30}$$

A chord (i,j) is superfluous if:

$$Z(T_{s+ij}) = Z(T_s) - c_{kl} + c_{ij} > U \tag{2.31}$$

From (2.30) and (2.31), it is evident that the number of arcs marked superfluous or indispensable will increase if: (a) an upper bound of good quality is used, and (b) a tight lower bound is obtained at the end of the ascent.

The imbedded heuristic described in the previous section provides a very tight upper bound in most problems tested. Moreover, the value U in the right hand sides of (2.30) and (2.31) can be reduced to U-1 provided: (i) a feasible solution associated with U is available, (ii) the identification of one optimal solution is sufficient, (iii) all the entries in the cost matrix are integers. This enables us to identify a greater number of indispensable and superfluous arcs.

We use the efficient implementation of branch-chord exchanges suggested by Volgenant & Jonker [1983]. This enables the computations and analyses of all the values $Z(T_{s-kl})$, $\forall (k,l) \in A_{T_s}$, and $Z(T_{s+ij})$, $\forall (i,j) \notin A_{T_s}$, to be carried out in $O(n^2)$ operations. An $O(n^2)$ method for implementing this procedure has also been suggested by Carpaneto, Fischetti & Toth [1989].

Apart from the use of this approach in the 1-tree algorithm of Volgenant & Jonker [1982], branch-chord exchanges have been used in the past for other combinatorial optimization problems. Gabow [1977] and Katoh, Ibaraki & Mine [1981] generate

spanning trees in order using this approach in a branch and bound type of an algorithm. Savelsbergh & Volgenant [1985] use this approach in degree constrained minimum spanning tree problem. Beasley [1984] uses a similar approach for the Steiner tree problem.

The lower bounds on the left hand sides of the inequalities (2.30) and (2.31) can be increased by modifying some of the multipliers. We describe the procedure to affect these changes in the following section.

2.5.2 Modifying the multipliers for increased lower bounds

This method of increasing the lower bound is based on the sequential multiplier updating method of Held & Karp [1970]. In their ascent method, the lower bound is increased by changing the multipliers associated with only one node at each ascent step. Volgenant & Jonker [1983] also use these ideas to obtain increased bounds. However, in their case it is only implemented when some strict conditions are satisfied by the tree.

The lower bounds are increased only marginally through this method, perhaps because most of the multipliers are nearly optimal at the end of the ascent. However, this can be used on every s-tree in the branch-chord exchange analysis to increase the values of $Z(T_{s+.})$ and $Z(T_{s-.})$. Although computationally expensive, we modify the multipliers to increase the bounds associated with every s-tree in the branch-chord analysis. This is because of our hypothesis that in order to identify the optimal solution, any computational effort expended in reducing the size of the graph at the

root node, would greatly reduce the computational cost of a tree search that would be necessary.

We know that by changing the value of one multiplier at a time, alternative s-trees arise. It is however possible to change the values of some of the multipliers without changing the structure of the related tree. From (2.25) it is clear that by changing the values of π_i for i such that $d_i = 1$ or $d_i = 3$ (in such a way that the s-tree remains unaltered), the s-tree lower bound can be increased.

Consider a node j with $d_j = 1$. For each chord (i,j) incident at j, $(i \neq s)$, let$(\hat{i},\hat{j})$ denote the longest branch in the fundamental path of chord (i,j) which is not incident at j. As in the previous section, (s,α) and (s,β) denote the branches that meet node s. Determine δ_j such that:

$$\delta_j = \min\left\{ \max_{\substack{i \in N_s \\ \hat{i},\hat{j} \neq j}} \left\{ \left(c_{ij} - c_{\hat{i}\hat{j}} \right) \right\}, \left(c_{sj} - c_{s\beta} \right) \right\} \tag{2.32}$$

According to the minimal spanning tree condition MSC2, $\delta_j \geq 0 \left(\forall j \in N_s \right)$. In addition, the s-tree is still minimal with respect to the transformed arc costs resulting from the change: $\pi_j \leftarrow \pi_j - \delta_j$. If $\delta_j > 0$ strictly, it follows from (2.25) that the TSP lower bound has increased by δ_j.

Consider a node l with $d_l > 2$. For each branch (k,l) incident at l, $(k \neq s)$, let$(\hat{k},\hat{l})$ denote the shortest chord (not incident at l) in its fundamental cutset. Determineε_l such that:

$$\varepsilon_l = \min\left\{ \min_{\substack{(k,l) \in T_s \\ k \in N_s}} \left\{ \left(c_{\hat{k}\hat{l}} - c_{kl} \right) \right\}, \left(c_{sl} - c_{s\beta} \right) \right\} \tag{2.33}$$

It follows from the minimal spanning tree condition MSC1 that $\varepsilon_l > 0 \left(\forall\ l \in N_s \right)$. The s-tree is still minimal with respect to the transformed arc costs due to the change: $\pi_l \leftarrow \pi_l + \varepsilon_l$. If $\varepsilon_l > 0$, the TSP lower bound has been increased by $\varepsilon_l * \left(d_l - 2 \right)$.

The multipliers of the two nodes linked to node s can be increased using a slightly different analysis. If $d_i = 2$, $(i = \alpha, \beta)$, then:

$$\pi_i \leftarrow \pi_i + \min_{j \notin N_s} \left\{ c_{sj} - c_{s\beta} \right\},\ (i = 1, 2) \tag{2.34}$$

The s-tree still remains minimal after the changed set of multipliers. Although this does not increase the TSP lower bound, this would enable better bounds for the $Z\left(T_{s+.} \right)$ values associated with chords incident at either α or β.

At the end of the ascent, the lower bound $Z\left(T_s \right)$ is increased using the relations (2.32) and (2.33). This results in a reduction of approximately 91% of the available arcs. These arcs are declared superfluous using (2.31) without the need for altering the multipliers on the nodes associated with $T_{s+.}$. The multipliers are adjusted and an attempt is made to increase the $Z\left(T_{s+.} \right)$ values only in such cases. Similarly, approximately 30% of the *required arcs*[1] are declared indispensable using (2.30) without the need for altering the multipliers on the nodes associated with $T_{s-.}$. The multipliers are adjusted and an attempt to increase $Z\left(T_{s-.} \right)$ is made only in such cases.

[1] n arcs are *required* in an n node STSP instance.

2.5.3 Structural Tests

Apart from the use of branch-chord exchanges to induce graph-sparsity, we also perform some simple tests that result in further reductions in the size of the problem. Three immediate tests are:

T2.1: If a node i has only two available arcs incident to it, then both arcs must be forced into the solution.

T2.2: If two indispensable arcs are incident to a node i, then all other arcs incident to it must be rejected.

T2.3: An arc that connects the end nodes of a chain of (at most n-2) indispensable arcs, must be rejected.

Tests T2.1 and T2.2 are analogous to restricting each node to be of degree 2. Test 2.3 is equivalent to a subtour elimination constraint.

We also implement a few simple and effective tests based on the existence of Hamiltonian cycles in a graph. A graph is called *Hamiltonian* if it contains at least one Hamiltonian cycle. The tests we develop now are also based on the *separability* conditions in a graph. We first introduce a few basic definitions:

Definitions: A *cut point* or an *articulation point* of a graph is a node whose exclusion increases the number of connected components of the graph. An *isthmus* is an arc whose removal increases the number of connected components of the graph.

We can then state the following definition (see Berge [1973], Gondran & Minoux [1984]): A graph $G=(N,A)$ is said to be *h-connected* if: (i) there exists a node set $N' \subset N$ of cardinality h such that the removal of these nodes (and all arcs incident to them) disconnects the graph; (ii) there exists no such node set of cardinality less than h.

Thus a *1-connected* graph is one that contains one or more articulation points (such graphs are called *separable*). It is obvious that if G is 1-connected, then no hamiltonian cycles exist (although s-trees can). We use this to design tests to identify more arcs to force in and reject:

T2.4: Consider an available arc l (that is not indispensable) such that either or both of the terminal nodes of this arc are the end nodes of chains of indispensable arcs. If by forcing arc l into the solution, (and after applying the resultant changes due to T2.1, T2.2 and T2.3), the resulting graph is either disconnected or 1-connected, then arc l must be superfluous.

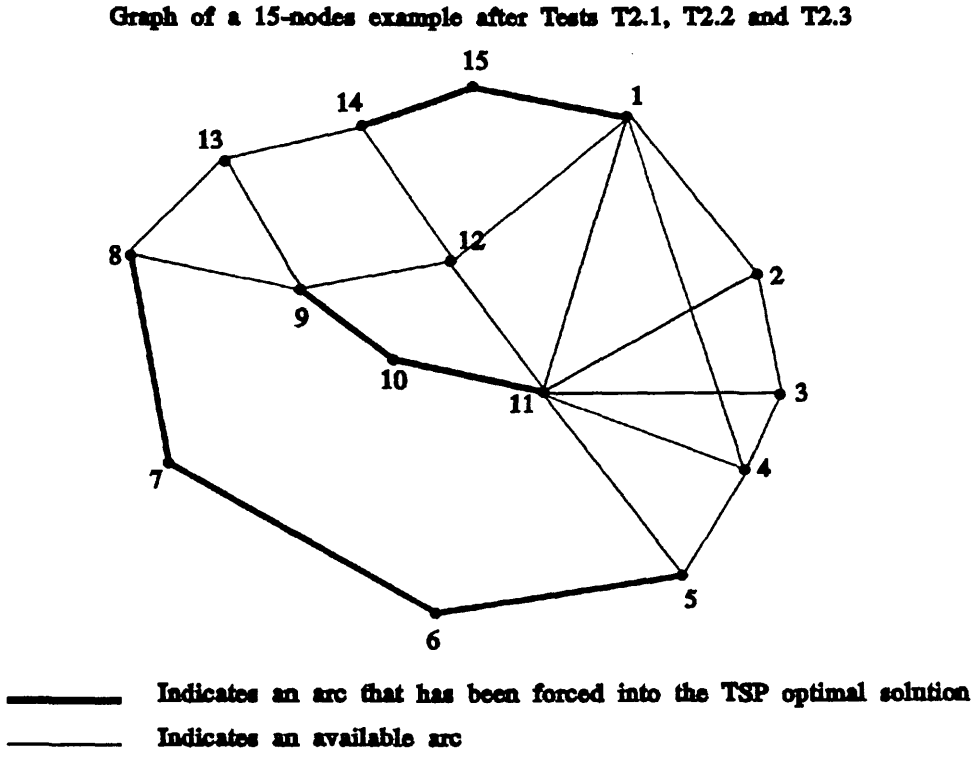
In a similar manner, we consider the effects on the graph of forcing an available (but not indispensable) arc out of the solution:

T2.5: Consider an available arc l (that is not indispensable) such that either or both of the terminal nodes of this arc are the end nodes of chains of indispensable arcs. If by rejecting arc l out of the solution (and after applying the resultant changes due to T2.1, T2.2 and T2.3), the resulting graph is either disconnected or 1-

connected, then arc l must be indispensable. For example, an isthmus in the graph must be indispensable.

We illustrate Test 2.4 through an example.

Figure 2.5 A 15-nodes example to illustrate Test T2.4

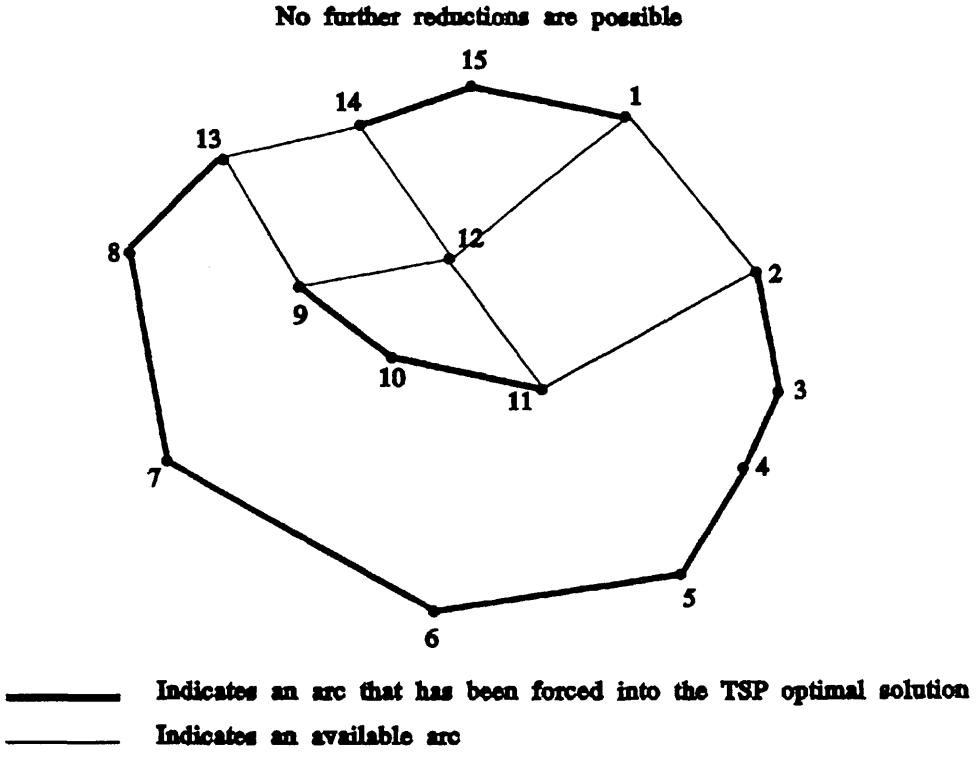


Consider the 15-node graph of Figure 2.5. The thickened lines indicate arcs that have been forced in.

- Consider the effect of forcing arc (8,9) into the solution: Arcs (8,13), (9,13) and (9,12) are rejected due to T2.2; Arc (5,11) is rejected due to T2.3; the remaining graph is 1-connected. Hence (8,9) is rejected. It follows from T2.1 that arc (8,13) can be forced in.

o Consider the effects of forcing arc (4,5) out of the solution: Arcs (5,11) is forced in due to T2.1; arcs (11,1), (11,2), (11,3), (11,4) and (11,12) are rejected due to T2.2; the remaining graph is 1-connected. Hence, arc (5,4) must be indispensable; It follows from T2.2 that arc (5,11) is rejected.

Figure 2.6 The 15-nodes example after application of T2.4



These tests are repeated till no further reduction in the size of the problem is possible. Figure 2.6 shows the graph of 15-node example after these tests are applied.

Both the tests are computationally inexpensive as they can be carried out in $O(n \mid A \mid)$ time. This is because the tests are applied to a maximum of n-2 terminal nodes of fixed chains. At each application, the 1-connected components of the graph can be identified in $O(\mid A \mid)$ time using the algorithm of Tarjan [1972] (see also

Gondran & Minoux [1984]). However, both tests together result in reasonable reductions in the size of the problem.

2.6 Computational results

In this section we report and analyze the computational results arising from the discussion in this chapter. This analysis will be split into the results for the upper bound heuristic, the ascent and results for the problem reduction tests. All our algorithms have been coded in FORTRAN 77 and run on a CYBER/930.

The test problems we use come from different sources. We use 23 well-known problems from the literature[2] and also 20 new Euclidean problems[3]. We also test our algorithms on 50 randomly generated symmetric problems.

2.6.1 Results for the upper bound heuristic

We denote the heuristic which we described in Section 2.4 by TSPH. Table 2.1 and Table 2.2 illustrate the results for TSPH on the well-known test problems and the newly generated Euclidean problems respectively.

The complete heuristic does not use much computing time. On average, the time spent on it is about 15% to 20% of the total time spent on the ascent. The quality of

[2]The optimal solution values, the sources, the size and the nomenclature of these problems are provided in Table 1.1.

[3]The newly generated Euclidean problems have been previously introduced in Section 1.9.

the bounds is however very tight. This is evident from Table 2.1 and Table 2.2. We use the respective *upper gaps* produced by other heuristics as an indication of the quality of the bounds obtained and as a basis of comparing TSPH with the other algorithms. The upper gap is defined as the difference between the upper bound and the optimal value as a percentage of the optimal value. We use K-Th-R, V-J and L-K to respectively denote the heuristics of Karg, Raymond & Thompson [1977][4], Volgenant & Jonker [1982], and Lin & Kernighan [1973] respectively.

Solutions produced by K-Th-R are on the average 0.59% above the optimal value for the 18 test problems used by Smith & Thompson [1977]. The upper gaps produced by J-V are on average 0.33% and 0.34% for the 23 well-known and 20 newly generated problem sets respectively. Upper gaps from L-K are on average 0.23% and 0.20% away from optimal for the same sets of problems. TSPH completely dominates all these algorithms and produces the best upper bounds for all problems tested (except ST603, for which, K-Th-R produces a better bound). The upper gaps from TSPH are on average only 0.076% and 0.023% away from optimal for the well-known and new problems respectively.

The heuristics K-Th-R, V-J, L-K and TSPH respectively solve 33%, 30%, 52% and 74% of the well-known problems to optimality. Of the 20 Euclidean problems, V-J, L-K, and TSPH respectively solve 7, 13 and 15 problems to optimality. This clearly indicates that TSPH completely dominates all the other heuristics it is compared with.

[4]Smith & Thompson [1977], in their 1-tree based STSP algorithm, use the heuristic algorithm of Karg & Thompson [1964] that incorporates improvements suggested by Raymond [1969].

Table 2.1 Computational results for the upper bound heuristic on some well known problems

Problem	K-Th-R	V-J	L-K	TSPH		
	Upper Gap	Upper Gap	Upper Gap	Upper Gap	# calls to Phase 2	Cpu Secs
DF42	0.000	0.00	0.00	0.00	4	15.18
HK48	0.440	0.08	0.41	0.00	5	23.38
GR48	-	0.65	0.00	0.00	4	21.53
KT57	0.440	0.90	0.60	0.44	5	37.56
GR120	-	0.36	0.03	0.03	11	435.93
ST481	0.610	1.04	0.72	0.01	5	22.33
ST482	0.000	0.00	0.00	0.00	5	22.21
ST483	0.000	0.00	0.00	0.00	4	21.78
ST484	0.000	0.29	0.30	0.00	5	21.28
ST485	0.000	0.00	0.00	0.00	5	21.95
ST600	0.960	0.00	0.97	0.00	6	43.58
ST601	0.420	0.05	0.00	0.00	6	45.53
ST602	1.990	0.00	0.00	0.00	6	44.22
ST603	0.000	0.49	0.38	0.11	6	53.41
ST604	0.430	0.24	0.00	0.00	5	42.98
ST605	0.130	0.11	0.00	0.00	5	42.43
ST606	1.440	0.09	0.00	0.00	6	39.31
ST607	1.220	0.21	0.06	0.00	6	45.12
ST608	0.020	0.38	0.00	0.00	6	48.64
ST609	2.560	0.00	0.00	0.00	6	47.03
NCE50	-	0.94	0.24	0.24	4	22.03
NCE75	-	1.31	1.50	0.93	5	64.48
NCE100	-	0.48	0.16	0.00	5	110.66

- Indicates that results for these problems are not available.

K-Th-R: the heuristic algorithm of Karg & Thompson [1964] with the modifications suggested by Raymond [1969].

V-J: the heuristic used by Volgenant & Jonker [1982] in their STSP algorithm.

L-K: the upper bound algorithm of Lin & Kernighan [1973].

TSPH: the heuristic described in Section 2.4.

Table 2.2 **Computational results for the upper bound heuristic on newly generated Euclidean problems**

Problem	V-J	L-K	TSPH		
	Upper Gap	Upper Gap	Upper Gap	# Calls to Phase 2	Cpu Secs
KC500	0.000	0.039	0.030	4	22.86
KC501	0.000	0.000	0.000	5	23.36
KC502	0.000	0.000	0.000	5	23.87
KC503	0.000	0.000	0.000	5	23.11
KC504	0.000	0.000	0.000	5	23.00
KC650	0.000	0.000	0.000	6	61.24
KC651	0.310	0.000	0.000	6	56.40
KC652	0.090	0.000	0.000	6	58.94
KC653	0.110	0.000	0.000	5	48.55
KC654	0.000	0.000	0.000	6	54.23
KC750	0.620	0.199	0.180	7	88.86
KC751	0.250	0.811	0.000	7	95.47
KC752	0.370	0.000	0.000	6	81.40
KC753	1.370	0.785	0.044	7	97.59
KC754	0.380	0.000	0.000	7	95.32
KC1000	0.170	1.274	0.013	9	188.29
KC1001	0.160	0.619	0.000	10	239.77
KC1002	1.390	0.000	0.000	9	219.11
KC1003	0.380	0.000	0.000	9	224.37
KC1004	1.190	0.346	0.198	9	245.40

2.6.2 Results for the *s*-tree ascent

Tables 2.3 and 2.4 compare the performance of the *s*-tree ascent described in this chapter - denoted as K-C - with results at the root node of the 1-tree algorithm of Volgenant & Jonker [1982] - denoted as V-J.

In our ascent we use the stopping rules and the parameter updating rules used by Smith & Thompson [1977]. Volgenant & Jonker [1982] use a fixed number of ascent iterations, a different multiplier updating formula, and a different formula for

calculating the scalar step size in their subgradient ascent. The end result of these two strategies is not that different either in terms of the number of trees calculated in the ascent or in terms of the *lower gaps* at the end of the ascent. The lower gap is defined as the difference between the optimal value and the lower bound as a percentage of the optimal value. On the average, 170 trees are calculated by K-C for the 23 well-known problems; 212 trees for the 20 newly generated Euclidean problems. Algorithm V-J evaluates 167 and 218 trees respectively for the two sets of test problems. The lower gaps derived from K-C are on average within 0.56% and 0.50% of the optimal value of problems in the two sets of problems, while the lower gaps due to V-J for the two sets are on average within 0.63% and 0.65% of the optimal value respectively. We can therefore conclude that K-C produces marginally better bounds than V-J. The reduction in the lower gaps is, on average, 17.09% at roughly the same computational cost. Both K-C and V-J solve one out of the 23 well-known test problems during the ascent. K-C solves 2 of the 20 Euclidean problems while V-J solves one. In the solved problems, we see that K-C converges to the optimal solution more rapidly than V-J - in $\frac{1}{4}$th the number of tree nodes required by V-J.

2.6.3 Results for the problem reduction tests

Tables 2.3 and 2.4 also present the number of arcs available and fixed by algorithms K-C and V-J after the ascent. The application of the branch-chord exchange analysis in K-C results in more arcs being declared superfluous and indispensable when compared to the values obtained by V-J. This is probably due to the tightness of both

our upper bounds as well as our lower bounds. The average *total gap* - which is the sum the upper and lower gaps - produced by TSPH and K-C over all the test problems is only 0.631%, while the total gap over all problems produced by V-J is 0.98%. The branch-chord analysis in K-C declares 91.6% and 29.6% of the available arcs on average to be superfluous and indispensable respectively. The branch-chord analysis in V-J declares only 89.3% of the arcs as superfluous and only 16.69% of the arcs are found to be indispensable. For the second stage of the problem reduction techniques, we use the simple structural tests T2.1 through to T2.5 till no further reductions are possible. Jonker & Volgenant use a series of tests including: (i) a non-optimality check that runs in $O(n^2)$ time; (ii) the structural tests T2.1, T2.2 and T2.3, and (iii) recalculation of 1-trees when a tree arc has been declared superfluous. Yet, we found that there was no major divergence in the average number of arcs declared superfluous and indispensable by K-C and V-J at the end of the second stage (represented in the Tables 2.3 and 2.4 under the columns titled: Structural properties). On the average K-C identifies 92.62% superfluous of the arcs to be superfluous. Hence, on average only 7.38% of the original graph remains[5]. On the average 41.1% of the required arcs are declared indispensable. At the end of this stage, V-J declares 91.93% superfluous and 44.2% indispensable arcs on average. If the required arcs are ignored the two methods can be compared on the basis of the arcs that remain amongst which decisions need to be made. In other words, if we define K as the number of arcs over which decisions have to be made expressed as a percentage of the problem size, ie., $K = \left\{ \frac{n(n-1)}{2} - n - n_s \right\} \Big/ n$, where n_s is the number of superfluous arcs, then the

[5]Note: This figure includes the n required arcs.

value K for the algorithms K-C and V-J are, on average, 1.65% and 1.835% respectively.

This gives an indication of the efficiency of the structural tests we use. Of the 23 well-known problems, 4 are solved during this stage of K-C. V-J solves 3 problems in this stage including DF42, the 42-node road map problem. Of the 20 new problems, K-C solves 3 in this stage. The one problem solved by V-J in this stage is solved by K-C during the ascent itself.

2.6.4 Results for randomly generated problems

We tested algorithm K-C on randomly generated problems and compared its performance with that of algorithm V-J which was run on the same problem instances. The problems were all generated in the standard manner: the cost matrix entries were drawn from a discrete uniform distribution on $[1, 1000]$. We generated a total of 50 problems in all; ten of each of the following sizes $n=50$, $n=60$, $n=70$, $n=80$ and $n=100$.

Table 2.5 shows the results for the random table problems. We observe that less trees are computed by K-C than by V-J although there is a small difference in the lower gaps obtained at the end of the ascent.

For the random table problems, the upper gaps produced by the heuristic TSPH within K-C are of a much better quality than those produced by V-J. As there are no initial upper bounds for these problems, we convert the first s-tree obtained in the ascent into a feasible tour using TSPH.

Table 2.3 Computational results for the *s*-tree ascent on some well-known problems

Problem		# of trees	Lower gap	Cpu Secs[1]	Branch-Chord Ex.		Structural properties	
					Available	Fixed	Available	Fixed
DF42	K-C	105	0.287	49.93	61	17	57	28
	V-J	126	0.300	-	66	19	Solved	-
HK48	K-C	145	0.149	77.65	66	22	63	35
	V-J	144	0.166	-	78	15	66	33
GR48	K-C	158	1.754	72.80	202	5	197	5
	V-J	144	1.775	-	274	0	196	7
KT57	K-C	184	0.368	125.68	180	10	173	16
	V-J	171	0.376	-	252	3	206	11
GR120	K-C	347	0.453	1419.21	410	12	406	12
	V-J	300	0.455	-	592	0	480	12
ST481	K-C	163	1.881	75.19	175	6	175	6
	V-J	144	1.890	-	230	0	180	3
ST482	K-C	140	0.185	73.95	70	19	61	31
	V-J	144	0.194	-	74	15	56	39
ST483	K-C	133	0.059	72.18	59	30	Solved	-
	V-J	144	0.064	-	61	33	Solved	-
ST484	K-C	157	0.664	71.89	101	8	99	13
	V-J	144	0.665	-	131	0	104	14
ST485	K-C	143	0.153	73.22	70	29	Solved	-
	V-J	144	0.153	-	69	25	Solved	-
ST600	K-C	176	0.225	144.97	107	25	76	43
	V-J	144	0.063	-	85	32	72	49
ST601	K-C	197	0.956	151.98	171	7	171	8
	V-J	144	0.965	-	179	3	132	21
ST602	K-C	32	Solved	136.40	-	-	-	-
	V-J	163	Solved	-	-	-	-	-
ST603	K-C	190	0.992	175.59	233	11	233	11
	V-J	180	0.995	-	216	0	165	18
ST604	K-C	174	0.148	142.70	103	26	Solved	-
	V-J	180	0.138	-	99	18	69	51
ST605	K-C	184	0.846	141.98	170	12	169	14
	V-J	180	0.878	-	186	0	138	22
ST606	K-C	155	0.025	130.23	68	46	Solved	-
	V-J	180	0.032	-	69	48	63	57
ST607	K-C	183	0.433	150.03	132	17	118	31
	V-J	180	0.440	-	167	4	114	30
ST608	K-C	191	1.119	161.30	211	7	211	7
	V-J	180	1.472	-	267	0	197	8

Table 2.3 Continued on following page...

Table 2.3 Continued...

Problem		# of trees	Lower gap	Cpu Secs	Branch-Chord Ex.		Structural properties	
					Available	Fixed	Available	Fixed
ST609	K-C	190	0.791	155.93	179	12	175	12
	V-J	180	0.795	-	184	0	118	27
NCE50	K-C	131	0.859	73.61	117	3	117	4
	V-J	150	0.873	-	185	0	146	3
NCE75	K-C	196	0.289	217.43	299	2	299	2
	V-J	225	0.311	-	420	0	326	5
NCE100	K-C	237	0.326	383.42	202	12	199	24
	V-J	300	0.343	-	380	0	334	5

K-C: indicates results for our s-tree ascent.

V-J: indicates results obtained using the algorithm of Volgenant & Jonker [1982].

[1]: these times do not include the time for the upper bound heuristic. These are available from Table 2.1.

Thereafter, the algorithm follows the same steps as in the algorithm for road map and Euclidean problems except that the parameter μ which controls the number of calls that are made to TSPH is reduced from $\left[\frac{n}{8}\right]$ to $\left[\frac{n}{6}\right]$. The average upper gaps for random problems are within 0.684% of the optimum. Although this figure is worse than that obtained for the road map and Euclidean problems, it almost compares with the gaps obtained by the heuristic K-Th-R for the well-known problems.

K-C also results in an appreciably lesser number of available arcs and a greater number of fixed arcs in all the sets when compared to V-J. Also, more problems are solved at the root node - either during the ascent itself or during the problem reduction stage - by algorithm K-C than by V-J. K-C solves 7 out of 10 problems for $n=50$, 5 problems for $n=60$, 3 problems for $n=70$, 4 problems for $n=80$ and 3 problems for $n=100$ respectively. The initial ascent in the algorithm V-J solves 6, 3, 4, 4 and 3 problems respectively in the same sets.

Table 2.4 Computational results for the *s*-tree ascent on newly generated problems

Problem		# of trees	Lower gap	Cpu Secs[1]	Branch-Chord Ex.		Structural properties	
					Available	Fixed	Available	Fixed
KC500	K-C	142	0.262	76.589	94	22	87	28
	V-J	150	0.276	-	93	18	77	32
KC501	K-C	148	0.256	78.439	83	21	Solved	Solved
	V-J	150	0.242	-	87	12	66	37
KC502	K-C	142	1.158	79.321	154	8	154	10
	V-J	150	1.167	-	173	0	105	24
KC503	K-C	40	Solved	21.906	-	-	-	-
	V-J	146	Solved	-	-	-	-	-
KC504	K-C	136	Solved	76.789	-	-	-	-
	V-J	150	0.109	-	62	37	Solved	Solved
KC650	K-C	202	0.779	202.296	201	12	188	15
	V-J	195	0.778	-	203	0	88	47
KC651	K-C	186	0.153	186.415	111	31	Solved	Solved
	V-J	195	0.179	-	161	6	122	31
KC652	K-C	194	0.704	194.646	177	13	173	14
	V-J	195	0.705	-	185	0	118	37
KC653	K-C	202	0.790	164.503	191	11	188	15
	V-J	195	0.805	-	198	4	149	21
KC654	K-C	197	0.830	180.925	193	5	191	8
	V-J	195	0.823	-	197	0	157	14
KC750	K-C	239	1.319	295.629	415	3	415	3
	V-J	225	1.323	-	497	0	377	4
KC751	K-C	224	0.659	313.639	234	9	234	9
	V-J	225	0.664	-	298	0	242	7
KC752	K-C	223	0.687	272.187	215	18	209	19
	V-J	225	0.697	-	273	0	200	20
KC753	K-C	230	0.650	321.670	257	20	253	21
	V-J	225	0.654	-	585	0	445	4
KC754	K-C	222	0.860	313.507	268	9	268	9
	V-J	225	0.861	-	349	0	236	15
KC1000	K-C	296	0.342	629.760	251	18	251	18
	V-J	300	0.347	-	328	0	255	17
KC1001	K-C	308	0.450	785.408	286	19	282	22
	V-J	300	0.445	-	845	0	691	0
KC1002	K-C	265	0.081	714.390	156	42	119	79
	V-J	300	0.081	-	225	12	155	64
KC1003	K-C	320	1.254	742.034	559	4	559	4
	V-J	300	1.267	-	700	0	548	3
KC1004	K-C	332	1.565	807.760	759	0	759	0
	V-J	300	1.575	-	1213	0	958	0

[1]: these times do not include the time for the upper bound heuristic.

Table 2.5 Computational Results for the *s*-tree ascent on randomly generated problems

		# of trees		Lower gap		Upper gap		# Available		# Fixed	
	n	K-C	V-J	K-C	V-J	K-C	V-J	K-C	V-J	K-C	V-J
Average	50	109	129	0.08	0.08	0.49	1.19	72	95	38	33
Std. Dev.		31	21	0.11	0.13	0.83	1.82	36	66	19	21
Maximum		150	150	0.36	0.40	2.48	5.32	153	252	4*	0*
Average	60	145	168	0.25	0.21	0.37	1.72	97	136	38	30
Std. Dev.		33	24	0.27	0.27	0.63	2.43	43	78	23	23
Maximum		177	180	0.72	0.70	1.96	6.74	172	288	3*	0*
Average	70	183	206	0.12	0.34	0.83	1.85	131	178	34	37
Std. Dev.		25	9	0.11	0.75	0.88	2.45	57	129	30	30
Maximum		211	210	0.35	2.57	2.48	6.58	221	407	2*	0*
Average	80	210	229	0.10	0.09	0.79	2.00	163	251	41	40
Std. Dev.		31	15	0.12	0.13	0.85	2.86	86	223	34	34
Maximum		242	240	0.34	0.34	2.30	8.75	313	756	2*	0*
Average	100	265	286	0.19	0.15	0.94	3.01	253	468	38	38
Std. Dev.		37	22	0.17	0.16	1.02	4.40	138	502	42	45
Maximum		300	300	0.55	0.55	3.23	15.11	549	1841	1*	0*

*: these entries are the minimum (and not maximum) values.

2.7 Conclusions

In this chapter, we presented a relaxation procedure for the STSP based on the minimal spanning tree relaxation. This procedure can be applied at the root node of a branch and bound algorithm. We also described a subgradient ascent procedure which is used to maximize the lower bounds. The lower gaps and the number of trees required to achieve this is comparable to (and in most cases better than) those of the best 1-tree based algorithm for the STSP. Tight upper bounds on the optimal solution value are obtained by a heuristic algorithm which is conveniently imbedded into the ascent procedure. We also described some strong problem reduction tests. These result in a very sparse graph in which around 92% of the arcs, on average, are declared superfluous and a third of the (required n) arcs are declared indispensable. We feel that this number can be further increased by recalculating s-trees when an arc in the tree has been declared superfluous at the problem reduction stage. The resulting new lower bound has to be greater than the previous one. If this new lower bound is greater than the upper bound minus one, then the best upper bound solution must be optimal. Since the total gaps at the end of the ascent are small, this procedure results in more problems being solved at the root node of the tree search.

CHAPTER 3

The Assignment Problem, The Minimal Spanning Arborescence and Complementary Duality

3.0 Outline

In this chapter we investigate the assignment problem and the minimal spanning arborescence problem as relaxations of the TSP. Lower bounds from the assignment problem are obtained through the restricted lagrangean approach of Balas & Christofides [1981]. A new bounding procedure is suggested for this approach. Bounds are also derived from the minimal spanning arborescence relaxation. The respective bounds and reduced costs are used to identify superfluous and indispensable arcs. In addition, we also exploit the complementary nature of the dual problems to reduce the size of the problem and also to increase to the lower bound.

3.1 Formulations and substructures

Consider a complete directed or undirected graph $G=(N,A)$, where N is the set of n nodes and A is the set of m arcs. In the directed case, $A=\{(i,j)|i,j=1,\ldots,n\}$. Let c_{ij} be the cost of the arc (i,j). The TSP is then the problem of finding a minimum cost Hamiltonian circuit, $\bar{H}$ in the graph G with cost matrix given by $[c_{ij}]$.

In Chapter 1 we introduced a formulation for the STSP. In Chapter 2 we described how that formulation was used to get lower bounds from the SST. The SST is a special substructure identified in the formulation of problem P_{STSP}. In this section we identify two substructures in the general formulation of the TSP. Although this formulation is more suitable for defining the ATSP and although the STSP can be formulated more compactly (see Section 1.5), we use it to define the STSP and *almost symmetric* ATSPs. In Chapter 4, we will give a transformation for the STSP which, when applied at the end of the s-tree procedure (discussed in Chapter 2) results in a partially asymmetric graph. We now introduce a formulation for the general TSP (or, the ATSP) where $c_{ij} \neq c_{ji}$ for some or all (i,j).

Let x_{ij} = 1 if arc (i,j) is in the optimal TSP tour

= 0 otherwise

We then formulate the TSP as follows:

Problem P_{TSP}:

$$\min_{x} \quad \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij}\, x_{ij} \tag{3.1}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} x_{ij} = 1, \qquad \forall\, j \in N, \tag{3.2}$$

$$\sum_{j=1}^{n} x_{ij} = 1, \qquad \forall\, i \in N, \tag{3.3}$$

$$\sum_{i \in S_t} \sum_{j \in \bar{S}_t} x_{ij} \geq 1, \qquad \forall\, S_t \subset N : S_t \neq \varnothing, \tag{3.4}$$

$$\sum_{i \in N} \sum_{j \in N} x_{ij} = n, \tag{3.5}$$

$$x_{ij} \in \{0,1\}, \qquad \forall\, (i,j) \in A. \tag{3.6}$$

Constraints (3.2) and (3.3) impose the value of one on the in-degree and the out-degree of each node. The constraint (3.5) is the result of surrogating the constraints (3.2) and (3.3). Although redundant in the formulation of problem P_{TSP}, we have retained it. The reason for including it will become clearer later in this chapter. The constraints (3.4) are the subtour elimination inequalities that impose solution-graph connectivity. There are many ways to express constraint (3.4). It can also be expressed as:

$$\sum_{i \in S_t} \sum_{j \in S_t} x_{ij} \leq |S_t| - 1, \qquad \forall\, S_t \subset N \tag{3.7}$$

We can halve the number of constraints in (3.4) by replacing them with:

$$\sum_{i \in S_t} \sum_{j \in \bar{S}_t} x_{ij} \geq 1, \qquad \forall\ S_t \subset N: r \in S_t \tag{3.8}$$

or, with:

$$\sum_{i \in S_t} \sum_{j \in \bar{S}_t} x_{ij} \geq 1, \qquad \forall\ S_t \subset N: S_t \neq \varnothing,\ r \notin S_t \tag{3.9}$$

where r is an arbitrarily chosen fixed node.

Substructures of this formulation can be identified, each defining a well-structured relaxation whose solution gives a valid lower bound on the value of the optimal solution to problem P_{TSP}. In each of these substructures, one of the constraint sets of P_{TSP} is relaxed. We now identify two such substructures.

3.1.1 The assignment problem substructure

The problem defined by the objective function (3.1) with constraints (3.2), (3.3) and (3.6) is the well known *min-sum assignment problem* (AP). The AP can be defined as the following graph problem: identify a minimum cost collection of disjoint subtours visiting all nodes of G. It is a special case of the transportation problem without capacity restrictions. It can also be viewed as a perfect matching problem on a bipartite graph.

The AP is a relaxation of P_{TSP} in which the subtour elimination constraints have been relaxed. If the optimal solution to the AP consists of only one subtour, then the relaxed constraint is also satisfied. Then, the AP solution is optimal for problem

P_{TSP} as well. If the AP solution is not a tour, each subset S_t of nodes in G that are visited by the same subtour determines a violated subtour elimination constraint.

The AP can be solved in $O(n^3)$ time using the *Hungarian algorithm* (see Kuhn [1955], Christofides [1975], or Lawler [1976]).

3.1.2 The *r*-arborescence substructure

Another substructure that can be identified within the formulation of the TSP is the *shortest spanning r-arborescence problem* SSA_r, or the *minimal r-arborescence* of G. Just as shortest spanning s-trees are referred to, simply, as s-trees, we refer to the shortest spanning r-arborescences as r-arborescences. Before discussing this substructure in problem P_{TSP} we introduce the following notations and definitions:

> A *spanning arborescence* rooted at node r (SA_r) is a subgraph of G with no subtours and exactly one arc directed into every node $j \in N$, $j \neq r$. Of all the spanning arborescences that exist the *shortest spanning arborescence* rooted at r (SSA_r) is a subgraph, denoted by $G_R = (N, A_R)$, for which the sum of the costs of its arcs is a minimum. We will denote an SSA_r simply by $R = (N, A_R)$, where $A_R \subset A$. The *r-arborescence* of G, denoted by $R_r = (N, A_{R_r})$ consists of an SSA_r as well as a least cost arc directed into the *root node r*. The r-arborescence of G is defined as a subgraph of G in which the in-degree of each node (including the root node r) is 1 and each node can be reached from the root node r.

The SSA_r problem is a relaxation of problem P_{TSP} in which the out-degree constraints (3.3) have been relaxed. If the optimal solution of the SSA_r with the objective function of problem P_{TSP} is such that each node has out-degree equal to one, then the relaxed constraints (3.3) are satisfied; each node which has out-degree different from 1 constitutes a violation of a constraint from (3.3).

The SSA_r can be solved by the efficient polynomial time algorithms of Edmonds [1967], Bock [1971], or Fulkerson [1974]. The algorithm of Edmonds was discovered independently by Chu & Liu [1965]. Edmonds [1967] provides a correctness proof for his algorithm based on concepts of linear programming. Karp [1971] gives a purely combinatorial correctness proof. Tarjan [1977] gave an efficient implementation of Edmond's algorithm requiring $O(n^2)$ steps for complete graphs. The same algorithm has an $O(\mid A \mid \log n)$ complexity for sparse graphs. Camerinini, Fratta & Maffioli [1979] corrected an error in Tarjan's implementation. Specialized and specific implementations that find r-arborescences in sparse graphs are also available. Gabow, Galil & Spencer [1984], and Gabow, Galil, Spencer & Tarjan [1986] make use of sophisticated data structures in algorithms of complexity $O\left(n \log n + \mid A \mid \log\log\log_{(\mid A \mid /n+2)} n\right)$ and $O(n \log n + \mid A \mid)$ respectively. Fischetti & Toth [1988b] give an efficient $O(n^2)$ implementation of Edmonds' algorithm. They also propose an efficient $O(n^2)$ algorithm for the computation of the linear programming reduced costs associated with the solution of the SSA_r.

3.1.3 Complementary substructures

The graph that defines the optimal solution to problem P_{TSP} is a subgraph of G possessing two main properties:

(i) it is strongly connected, in the sense that a path exists in it between every pair of nodes,

(ii) the degree of each node is two.

The subgraph of G that defines the optimal solution to the AP possesses property (ii). It does not necessarily satisfy property (i). The subgraph of G that defines the solution to the r-arborescence possesses property (i) while the property (ii) is not necessarily satisfied. Thus, the solutions to the AP and the r-arborescence possess properties that are complementary with respect to the properties of the optimal TSP solution.

In this chapter, we will show how these two substructures and their respective solutions can be used in a procedure to obtain lower bounds for the TSP. We will also exploit the complementary nature of the two substructures to tighten the lower bounds obtained through an iterative procedure.

3.2 Bounds from the assignment problem

The assignment problem (AP) can be formulated as:

Problem P_{AP}:

$$\min_{x} \quad \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij}\, x_{ij}$$

s.t. (3.2), (3.3) and (3.6)

This is a relaxation of the TSP, where the subtour elimination constraints (3.4) or (3.7), or any other constraint similar to these have been dropped. Balas & Christofides [1981] include *both* (3.4) and (3.6) as well as some positive linear combinations of other similar constraints and express the resulting family of subtour elimination inequalities in their generic form as:

$$\sum_{i \in N} \sum_{j \in N} a_{ij}^{t}\, x_{ij} \geq a_{0}^{t}, \qquad t \in T. \tag{3.10}$$

The integer programming formulation of the TSP then consists of the objective function (3.1) with the constraints (3.2), (3.3), (3.6) and (3.10). Let X denote the set of feasible solutions to problem P_{AP}. The relaxed constraints can be introduced into the objective function of problem P_{AP} by associating a lagrangean multiplier ω_t, $t \in T$, with the tth constraint from (3.10) which is violated. The lagrangean problem is then written as:

$$\min_{x \in X} \quad \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij}\, x_{ij} - \sum_{t \in T} \omega_t \left(\sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij}^{t} x_{ij} - a_{0}^{t} \right) \tag{3.11}$$

s.t. (3.2), (3.3) and (3.6)

The lagrangean objective function (3.11) can be simplified and the resulting formulation can be expressed as:

Problem $P_{AP}(\omega)$:

$$\min_{x \in X} \quad \sum_{i \in N} \sum_{j \in N} \left(c_{ij} - \sum_{t \in T} \omega_t a_{ij}^t \right) x_{ij} + \sum_{t \in T} \omega_t a_0^t \qquad (3.12)$$

s.t. (3.2), (3.3) and (3.6)

The value $Z\left(P_{AP}(\omega)\right)$ forms a lower bound on the value of the optimal TSP. The maximum such bound is found by identifying a multiplier vector $\omega = \omega^*$ such that:

$$Z\left(P_{AP}(\omega^*)\right) = \max_{\omega \geq 0} Z\left(P_{AP}(\omega)\right) \qquad (3.13)$$

Problem (3.13) is the lagrangean dual of the TSP. The solution to this problem could be solved by a subgradient optimization procedure similar to the approach used in the case of the *s*-tree relaxation (Chapter 2). However, since the number of subtour elimination constraints increases exponentially with problem size, the number of multipliers ω_t in (3.12) has an exponential rate of increase with *n*. Hence, the ascent, as described in Chapter 2, becomes computationally infeasible. Balas & Christofides [1981] developed a successful method to solve (3.13) known as the *restricted lagrangean* approach. The lagrangean dual, (3.13) is replaced by a restricted lagrangean dual given by:

$$\max_{\omega \in W} Z\left(P_{AP}(\omega)\right) \tag{3.14}$$

where,
$$W = \left\{ \omega \,\middle|\, \omega \geq 0 \text{ and } \exists\ (u^*, v^*) \text{ such that } \begin{array}{r} c_{ij} - u_i^* - v_j^* - \sum_{t \in T} \omega_t a_{ij}^t = 0 \text{ if } \bar{x}_{ij} = 1 \\ \geq 0 \text{ if } \bar{x}_{ij} = 0 \end{array} \right\} \tag{3.15}$$

Here, $\bar{x} \in X$ is the optimal solution to the AP and u_i^* and v_j^*, $(i, j = 1, \ldots, n)$ are the optimal AP duals. (3.14) thus restricts the multipliers ω_t to values that, together with u_i^* and v_j^* form a feasible solution to the dual of the LP given by (3.1), (3.2), (3.3), and (3.10) along with $x_{ij} \geq 0$, $\forall\ i, j \in N$. (3.14) is then solved by a polynomially bounded, sequential, non-iterative and approximate procedure which determines a set of multipliers ω^e such that $Z\left(P_{AP}(\omega^e)\right)$ is a good approximation of $Z\left(P_{AP}(\omega^*)\right)$. A brief description of this procedure to find ω^e is provided in the following section.

3.2.1 The restricted lagrangean method to improve the AP lower bounds

This procedure starts by solving problem P_{AP} with the TSP cost matrix $C = [c_{ij}]$ and obtains the solution $\bar{x}$ and the corresponding duals u_i^* and v_j^*. Values to the multipliers ω_t are assigned in a sequential manner without changing the values assigned earlier. *Bounding procedures* identify inequalities from (3.10) which are violated by $\bar{x}$. Each violated inequality *admits* a positive multiplier if there exists a $\omega_t > 0$, which together with the multipliers already chosen, satisfies the constraints in W given by (3.15). The introduction of each violated inequality into (3.14) strengthens the lower bound. Let T_M denote the set of violated inequalities from T

that are identified by the bounding procedure M. The value of the strengthened lower bound after the application of bounding procedure M is given by:

$$V(BCM) = V(BCM-1) + \sum_{t \in T_M} \omega_t a_t^0, \tag{3.16}$$

where, $V(BCM-1)$ is the value of the lower bound obtained after the (M-1)th bounding procedure. The initial lower bound is: $\sum_{i=1}^{n} u_i^* + \sum_{j=1}^{n} v_j^*$.

At any stage of the procedure, $\bar{C} = [\bar{c}_{ij}]$ denotes the reduced cost matrix, where $\bar{c}_{ij} = c_{ij} - u_i^* - v_j^* - \sum_{t \in T} \omega_t a_{ij}^t$. The graph $G_0 = \{N, A_0\}$, where $A_0 = \{(i,j) | \bar{c}_{ij} = 0\}$ is known as the *admissible graph* - a spanning subgraph of G containing arcs with zero reduced costs. The inclusion of each violated inequality into the restricted lagrangean objective function adds at least one new arc to the admissible graph. As long as G_0 is not strongly connected, new valid inequalities that are violated by the solution to problem P_{AP} can be found, each of which admit positive multipliers. When no more such inequalities can be found by the bounding procedures, G_0 is regarded as a strongly connected graph. If the admissible graph becomes Hamiltonian and if a tour can be found in G_0, whose arcs satisfy the constraints (3.10) with strict equality for all $t \in T$, then the tour is optimal for the problem P_{TSP}.

3.2.2 Bounding procedures

Balas & Christofides [1981] identify three main bounding procedures, 1,2 and 3. Each of these procedures are polynomially bounded. We use the bounding procedures 1 and 3 and also extend the results of bounding procedure 3 to form an additional

procedure. These three procedures we use are termed BC1, BC3 and BC31 respectively. Each of these procedures identifies a particular type of valid inequality from (3.10) which is violated. The three types of inequalities identified by these three procedures are denoted by T_1, T_3 and T_{31} respectively. The components of ω corresponding to these inequalities will be denoted by $\lambda=\left(\lambda_i\right)_{i\in T_1}$, $\gamma=\left(\gamma_i\right)_{i\in T_3}$ and $\sigma=\left(\sigma_i\right)_{i\in T_{31}}$.

Bounding procedure BC1

This procedure admits multipliers by identifying inequalities (3.4) which are violated by $\bar{x}$. Let T_1 represent this set of violated inequalities. Then, BC1 admits a positive multiplier λ_t for every $t\in T_1$. This set of inequalities can be represented as:

$$\sum_{(i,j)\in K_t} x_{ij}\geq 1, \qquad t\in T_1, \tag{3.17}$$

where K_t is a *directed cutset* defined as $K_t\equiv\left(S_t,\bar{S}_t\right)=\left\{(i,j)\in A\,\middle|\, i\in S_t,\ j\in\bar{S}_t\right\}$. A positive multiplier λ_t will be admitted by that cutset K_t which satisfies:

$$K_t\cap A_0=\varnothing \tag{3.18}$$

Choose any node $i\in N$. A node $j\in N$ is said to belong to the *reachable set* $R(i)$ of i if there is a directed path from i to j in G_0. If $R(i)=N$, then there is no cutset K_t with $i\in S_t$ that satisfies (3.18). We then choose another node from N and repeat the process. If $R(i)=N\ \forall\ i\in N$, this procedure is stopped; G_0 is strongly connected and $K_t\cap A_0\neq\varnothing$ for all cutsets K_t in G.

If $R(i) \neq N$ for some $i \in N$, then $K_t = \left(R(i),\ N - R(i)\right)$ satisfies (3.18). The largest initial value of λ_t corresponding to this violated inequality is:

$$\lambda_t^0 = \min_{(i,j) \in K_t} \left\{ \bar{c}_{ij} \right\} \tag{3.19}$$

and $\left[\bar{c}_{ij} \right]$ is updated using:

$$\begin{aligned} \bar{c}_{ij} &= \bar{c}_{ij} - \lambda_t^0 \ \forall\ (i,j) \in K_t \\ &= \text{unchanged otherwise} \end{aligned}$$

This adds to A_0 all arcs for which $\bar{c}_{ij}$ has just become 0. The search then commences for a new violated cutset.

The final values of λ_t increase the lower bound by: $\sum_{t \in T_1} \lambda_t$. If $V(BC1)$ represents the value of the lower bound at the end of BC1, then:

$$V(BC1) = Z\left(\mathrm{P_{AP}}\right) + \sum_{t \in T_1} \lambda_t \tag{3.20}$$

The maximum number of cutsets that have to be examined are $(h-1)(h+2)/2$, where h is the number of subtours in $\bar{x}$. The worst-case complexity of this procedure can be shown to be $O(n^4)$.

Bounding procedure BC3

This procedure admits multipliers by identifying *articulation points* in the admissible graph. An articulation point of G_0 is a node k whose removal leaves G_0 with more than one component. Denote the set of nodes in the two components as S_t^a and S_t^b. Every tour must contain an arc that belongs to at least one of the cutsets $K_t^a = \left\{ (i,j) \in A \,\middle|\, i \in S_t^a, j \in S_t^b \right\}$ and $K_t^b = \left\{ (i,j) \in A \,\middle|\, i \in S_t^b, j \in S_t^a \right\}$. Any tour must thus satisfy:

$$\sum_{(i,j) \in K_t^a \cup K_t^b} x_{ij} \geq 1 \tag{3.21}$$

This procedure searches the set of nodes N for articulation points. If an articulation point $k \in N$ is found, then the cutsets K_t^a and K_t^b are formed and a positive multiplier γ_t can be admitted since:

$$K_t^a \cap A_0 = K_t^b \cap A_0 = \varnothing \tag{3.22}$$

The largest value that can be assigned to γ_t is:

$$\gamma_t = \min_{(i,j) \in K_t^a \cup K_t^b} \left\{ \bar{c}_{ij} \right\} \tag{3.23}$$

Note that if G_0 has no articulation point, then for any node k, the maximum of (3.21) is 0 and no inequality (3.21) admits a positive multiplier. The reduced cost matrix is updated using:

$$\begin{aligned} \bar{c}_{ij} &= \bar{c}_{ij} - \gamma_t \ \forall \ (i,j) \in K_t^a \cup K_t^b \\ &= \text{unchanged otherwise} \end{aligned}$$

The admissible graph is updated accordingly. If T_3 denotes the set of violated inequalities in (3.21), then the lower bound at the end of BC3 is given by:

$$V(BC3) = Z(\mathrm{P_{AP}}) + \sum_{t \in T_1} \lambda_t + \sum_{t \in T_3} \gamma_t \tag{3.24}$$

The procedure is applied to all n nodes. Testing for connectivity requires $O(|A|)$ steps. Hence this procedure can be performed in $O(n|A|)$ steps.

Bounding procedure BC31

This is an extension of the procedure BC3. This procedure identifies pairs of nodes in the admissible graph whose removal leaves G_0 with more than two components. In general, if the removal of p nodes from G_0 leaves it with more than p components, a positive multiplier can be identified. We implement this procedure for $p=2$.

Consider a pair of nodes $\{k,l\} \in N$ whose removal from G_0 leaves it with three disconnected components. Denote the set of nodes in the three disconnected components as S_t^c, S_t^d and S_t^e. Every tour must contain at least one arc that belongs to the cutsets $K_t^c=\{S_t^c, S_t^d\}$, or $K_t^d=\{S_t^d, S_t^c\}$, or $K_t^e=\{S_t^c, S_t^e\}$, or $K_t^f=\{S_t^e, S_t^c\}$, or $K_t^g=\{S_t^d, S_t^e\}$, or $K_t^h=\{S_t^e, S_t^d\}$. Define:

$$K_t^{31}=\left\{(i,j)\middle|(i,j)\in K_t^c \cup K_t^d \cup K_t^e \cup K_t^f \cup K_t^g \cup K_t^h\right\} \tag{3.25}$$

In other words, every tour must satisfy:

$$\sum_{(i,j)\in K_t^{31}} x_{ij} \geq 1 \tag{3.26}$$

where K_t^{31} is defined using (3.25). Procedure BC31 investigates whether every pair of nodes in N satisfies (3.26). If a pair $\{k,l\}$ is found that violates (3.26), a positive multiplier σ_t can be admitted since:

$$K_t^c \cap A_0 = K_t^d \cap A_0 = K_t^e \cap A_0 = K_t^f \cap A_0 = K_t^g \cap A_0 = K_t^h \cap A_0 = \varnothing \tag{3.27}$$

The largest value that can be assigned to σ_t due to the violated inequality (3.26) is:

$$\sigma_t = \min_{(i,j)\in K_t^{31}} \left\{\bar{c}_{ij}\right\} \tag{3.28}$$

The reduced cost matrix is updated using:

$$\bar{c}_{ij} = \bar{c}_{ij} - \sigma_t \;\forall\; (i,j) \in K_t^{31}$$
$$= \text{unchanged otherwise}$$

The admissible graph is updated accordingly as some more zero reduced cost arcs will be introduced by this procedure. If T_{31} denotes the set of violated inequalities in (3.26), then the lower bound at the end of procedure BC31 is given by:

$$V(BC31) = Z(\mathrm{P_{AP}}) + \sum_{t \in T_1} \lambda_t + \sum_{t \in T_3} \gamma_t + \sum_{t \in T_{31}} \sigma_t \qquad (3.29)$$

Procedure BC31 is applied to all pairs of nodes. Since testing for the connected components requires $O(|A|)$ steps (Tarjan [1972]), the procedure can be implemented in $O(n|A|)$ steps. We illustrate the three bounding procedures with an example in the following section.

3.2.3 An example

Consider the 10-node TSP whose input cost matrix is shown in Table 3.1. Table 3.2 shows the optimal solution $\bar{x}$ to the AP ($\bar{x}_{ij} = 1$ for those cells (i,j) which have a bold zero in them; $x_{ij} = 0$ otherwise). The solution consists of two subtours: {2,10} and {1,8,9,3,7,5,6,4}. The numbers in the matrix represent the reduced costs associated with the AP optimal solution and the numbers in the boxes along the rim represent the optimal AP dual vectors u and v.

Table 3.1 Input cost matrix of a 10-nodes example.

	1	2	3	4	5	6	7	8	9	10
1	-	6	19	22	16	20	5	3	9	11
2	11	-	23	20	14	15	7	1	18	2
3	18	9	-	17	6	15	1	15	14	15
4	5	3	12	-	25	6	19	14	9	21
5	16	4	12	13	-	4	17	8	10	18
6	8	25	23	3	6	-	11	15	16	10
7	14	15	5	15	1	17	-	12	6	6
8	18	6	18	3	15	4	23	-	3	16
9	8	15	3	19	6	24	8	19	-	23
10	25	2	8	4	4	24	14	16	14	-

Table 3.2 Optimal solution to the AP.

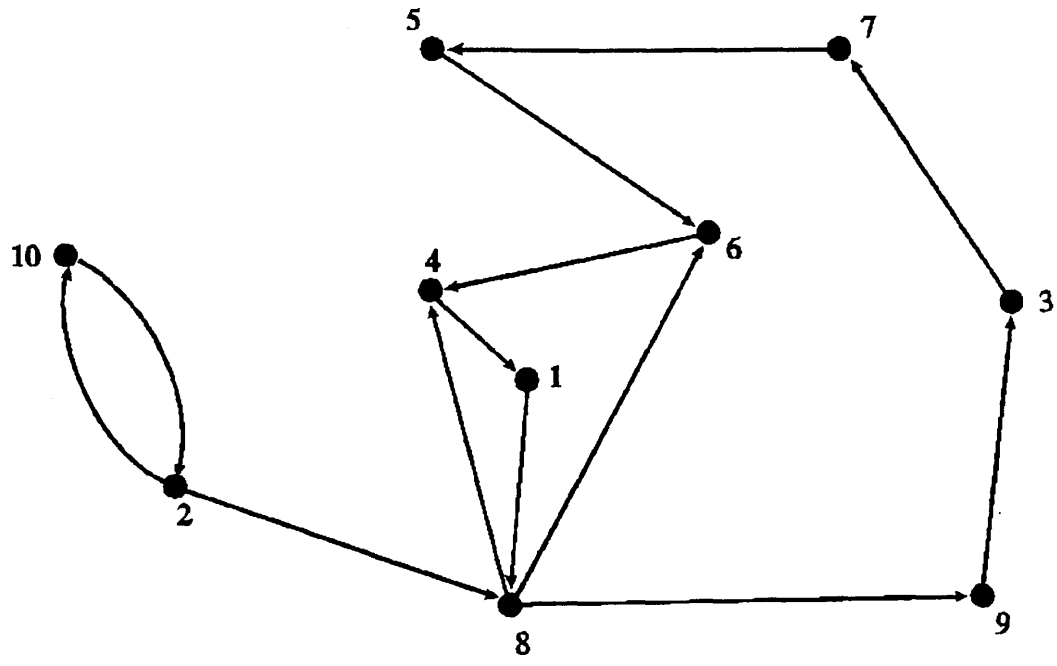
	1	2	3	4	5	6	7	8	9	10	
1	-	2	14	17	13	14	2	**0**	4	7	2
2	6	-	20	17	13	11	6	0	15	**0**	0
3	13	7	-	14	5	11	**0**	14	11	13	0
4	**0**	1	9	-	24	2	18	13	6	19	0
5	11	2	9	10	-	**0**	16	7	7	16	0
6	3	23	20	**0**	5	-	10	14	13	8	0
7	9	13	2	12	**0**	13	-	11	3	4	0
8	13	4	15	0	14	0	22	-	**0**	14	0
9	3	13	**0**	16	5	20	7	18	-	21	0
10	20	**0**	5	1	3	20	13	15	11	-	0
	5	2	3	3	1	4	1	1	3	2	

The initial lower bound is $Z(\mathrm{P_{AP}})=27$. The admissible graph corresponding to this solution is shown in Figure 3.1.

Bounding procedure BC1

Cutset $K_1=(\{1,3,4,5,6,7,8,9\},\{2,10\})$ violates the inequality (3.17). Since $K_1 \cap A_0=\varnothing$, we use (3.19) to admit a positive multiplier $\lambda_1=\overline{c}_{4,2}=1$.

Figure 3.1 The admissible graph G_0 defined by the optimal assignment

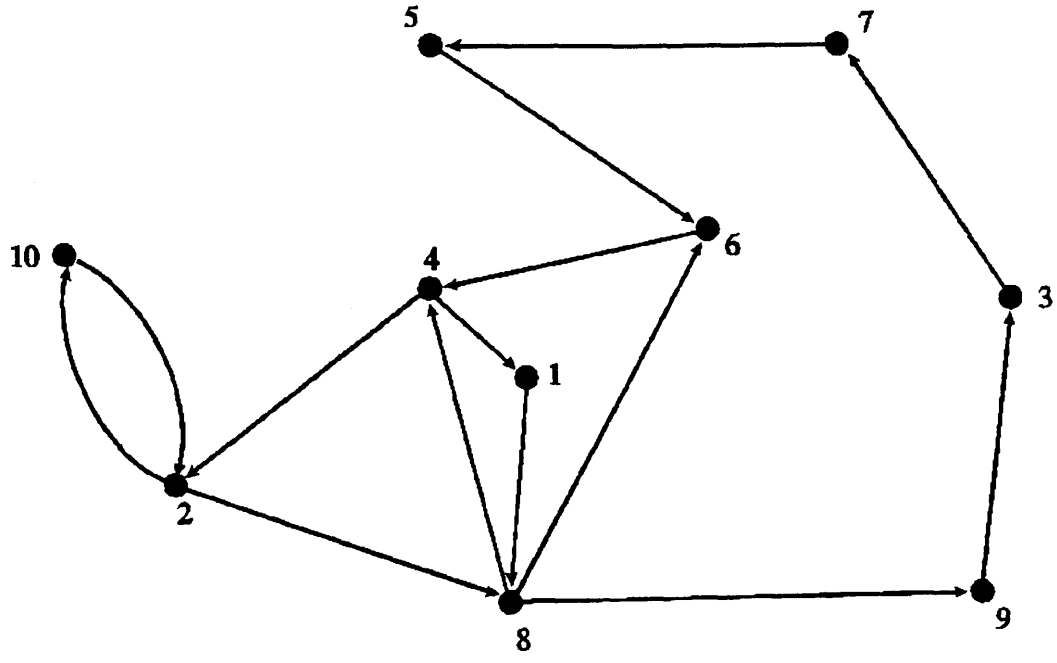


The procedure is then terminated as there are no more inequalities (3.17) that are violated. The new lower bound is $V(BC1) = 27 + 1 = 28$. The updated reduced cost matrix is shown in Table 3.3 and the corresponding admissible graph is shown in Figure 3.2.

Table 3.3 Reduced cost matrix after procedure BC1.

	1	2	3	4	5	6	7	8	9	10
1	-	1	14	17	13	14	2	**0**	4	6
2	6	-	20	17	13	11	6	0	15	**0**
3	13	6	-	14	5	11	**0**	14	11	12
4	**0**	0	9	-	24	2	18	13	6	18
5	11	1	9	10	-	**0**	16	7	7	15
6	3	22	20	**0**	5	-	10	14	13	7
7	9	12	2	12	**0**	13	-	11	3	3
8	13	3	15	0	14	0	22	-	**0**	13
9	3	12	**0**	16	5	20	7	18	-	20
10	20	**0**	5	1	3	20	13	15	11	-

Figure 3.2 The admissible graph G_0 after procedure BC1



Bounding procedure BC3

Node 2 is an articulation point of G_0 (in Figure 3.2). Removal of node 2 from G_0 leaves it with 2 disconnected components $S_1^a=\{1,3,4,5,6,7,8,9\}$ and $S_1^b=\{10\}$. Define the cutsets:

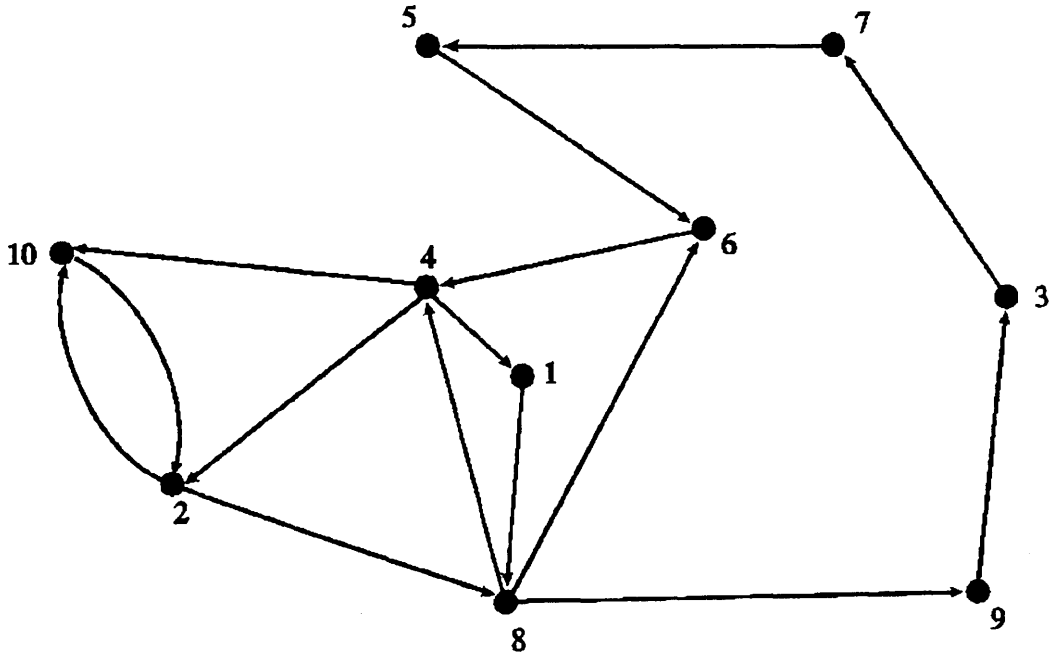
$$K_1^a=\big(\{1,3,4,5,6,7,8,9\},\{10\}\big) \text{ and } K_1^b=\big(\{10\},\{1,3,4,5,6,7,8,9\}\big).$$

The arc cutset $K_1^a \cup K_1^b$ violates the inequality (3.21). Since $K_1^a \cap A_0=K_1^b \cap A_0=\varnothing$, we use (3.23) to admit a positive multiplier $\gamma_1=\bar{c}_{4,10}=1$. The procedure stops as there is no other articulation point in G_0. The new lower bound is $V(BC3)=27+1+1=29$. The updated reduced cost matrix is shown in Table 3.4 and the corresponding admissible graph is shown in Figure 3.3.

Table 3.4 Reduced cost matrix after procedure BC3.

	1	2	3	4	5	6	7	8	9	10
1	-	1	14	17	13	14	2	**0**	4	5
2	6	-	20	17	13	11	6	0	15	**0**
3	13	6	-	14	5	11	**0**	14	11	11
4	**0**	0	9	-	24	2	18	13	6	17
5	11	1	9	10	-	**0**	16	7	7	14
6	3	22	20	**0**	5	-	10	14	13	6
7	9	12	2	12	**0**	13	-	11	3	2
8	13	3	15	0	14	0	22	-	**0**	12
9	3	12	**0**	16	5	20	7	18	-	19
10	19	**0**	4	0	2	19	12	14	10	-

Figure 3.3 The admissible graph G_0 after procedure BC3



Bounding procedure BC31

Consider the removal of the nodes 4 and 8 and all the arcs connected to them from the admissible graph represented in Figure 3.3. The remaining contains three disconnected components $S_1^c=\{1\}$, $S_1^d=\{2,10\}$ and $S_1^e=\{3,5,6,7,9\}$.

We define the directed cutsets: $K_1^c=\big(\{1\},\{2,10\}\big)$, $K_1^d=\big(\{2,10\},\{1\}\big)$, $K_1^e=\big(\{1\},\{3,5,6,7,9\}\big)$, $K_1^f=\big(\{3,5,6,7,9\},\{1\}\big)$, $K_1^g=\big(\{2,10\},\{3,5,6,7,9\}\big)$, and $K_1^h=\big(\{3,5,6,7,9\},\{2,10\}\big)$.

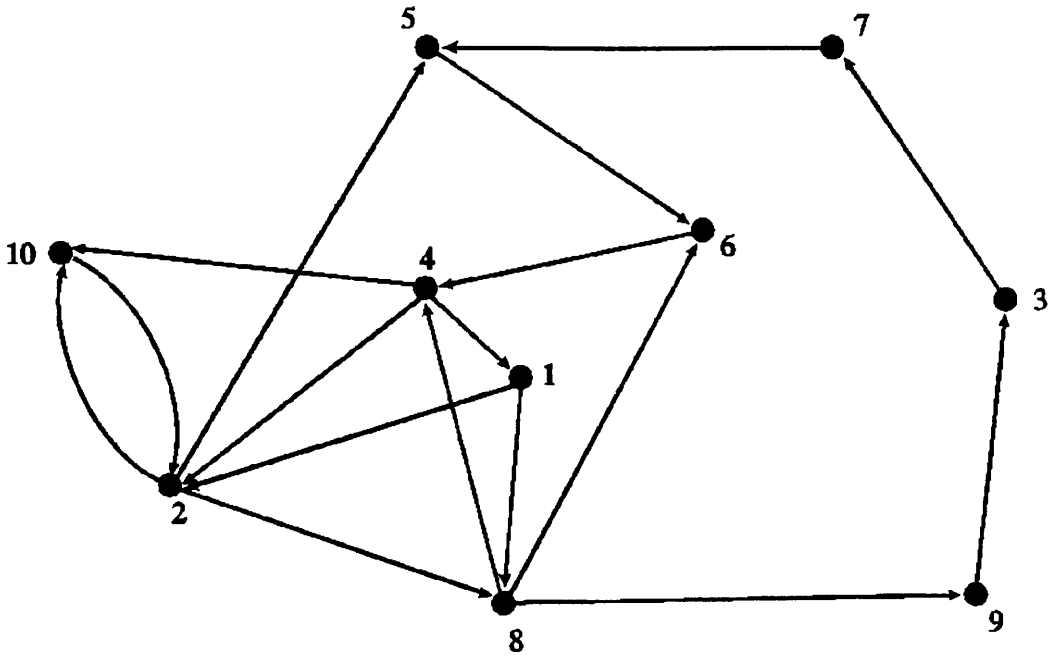
The relation (3.25) is used to define the cutset K_1^{31} which violates (3.26).

Since $K_1^c \cap A_0=K_1^d \cap A_0=K_1^e \cap A_0=K_1^f \cap A_0=K_1^g \cap A_0=K_1^h \cap A_0=\varnothing$, we use (3.27) to admit a positive multiplier $\sigma_1=\bar{c}_{1,2}=\bar{c}_{2,5}=1$. The procedure terminates as there are no other pairs of points in G_0 whose removal causes a violation of (3.26). The new lower bound is $V(BC3)=27+1+1+1=30$. The updated reduced cost matrix is shown in Table 3.5 and the corresponding admissible graph is shown in Figure 3.4.

Table 3.5 Reduced cost matrix after procedure BC31.

	1	2	3	4	5	6	7	8	9	10
1	-	1	13	17	12	13	1	**0**	3	4
2	5	-	19	17	12	10	5	0	14	**0**
3	12	5	-	14	5	11	**0**	14	11	10
4	**0**	0	9	-	24	2	18	13	6	17
5	10	0	9	10	-	**0**	16	7	7	13
6	2	21	20	**0**	5	-	10	14	13	5
7	8	11	2	12	**0**	13	-	11	3	1
8	13	3	15	0	14	0	22	-	**0**	12
9	2	11	**0**	16	5	20	7	18	-	18
10	18	**0**	3	0	1	18	11	14	9	-

Figure 3.4 The admissible graph G_0 after procedure BC31



3.3 The minimal spanning *r*-arborescence problem

The procedure to obtain bounds from the *r*-arborescence relaxation in an undirected graph, is analogous to the application of the *s*-tree relaxation approach for an undirected graph. This relaxation was first used by Held & Karp [1970] for the ATSP. Computational studies carried out by Smith [1975], and Hong [1978] suggest that this is a weak relaxation for the ATSP when compared to the AP relaxation.

The *r*-arborescence problem can be formulated as:

Problem P_{R_r}:

$$\min_{x} \quad \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}$$

$$\text{s.t.} \quad \sum_{i \in S_t} \sum_{j \in \overline{S}_t} x_{ij} \geq 1, \qquad \forall\ S_t \subset N: r \in S_t \tag{3.31}$$

$$\sum_{i \in N} \sum_{j \in N} x_{ij} = n, \tag{3.32}$$

$$\sum_{i \in N} x_{ir} = 1, \tag{3.33}$$

$$x_{ij} \in \{0,1\} \qquad \forall\ (i,j) \in A. \tag{3.34}$$

The solution to the above problem is a spanning subgraph denoted by R_r in which the in-degree of every node (except the root node r) is 1. As in the case of the s-tree relaxation, a careful choice of the root node could yield good results. However, we take the root node arbitrarily to be node 1. We, nevertheless, retain the denotation of the problem as the r-arborescence (as opposed to the 1-arborescence).

The problem of finding the r-arborescence can be decomposed into two problems:

(P3.1) find an SSA_r in G, and

(P3.2) find a minimum cost arc (i,r) in G.

Problem (P3.1) the SSAr can be formulated as follows:

Problem P_R:

$$\min_{x} \quad \sum_{i=1}^{n} \sum_{i=1}^{n} c_{ij} x_{ij}$$

$$\text{s.t.} \quad \sum_{i \in S_t} \sum_{j \in \bar{S}_t} x_{ij} \geq 1, \qquad \forall\ S_t \subset N : r \in S_t, \tag{3.35}$$

$$\sum_{i=1}^{n} \sum_{j=1}^{n} x_{ij} = n - 1, \tag{3.36}$$

$$x_{ij} \in \{0,1\}, \qquad \forall\ (i,j) \in A. \tag{3.37}$$

Efficient algorithms are available to solve problem P_R (see Section 3.1.2). We use the $O(n^2)$ implementation of the algorithm of Edmonds [1967] given by Fischetti & Toth [1988b].

3.3.1 Lagrangean bounds from the *r*-arborescence substructure

Problem P_{R_r} is a relaxation of problem P_{TSP} in which the out-degree constraints (3.3) have been relaxed. A tighter relaxation can be obtained by including the relaxed degree constraints in the objective function of problem P_R with the use of lagrange multipliers $\delta = \{\delta_i \mid i \in N\}$. The formulation of this problem is given below:

$$\min_{x \in R_r} \quad \sum_{i=1}^{n} \sum_{i=1}^{n} c_{ij} x_{ij} + \sum_{i=1}^{n} \delta_i \left(\sum_{j=1}^{n} x_{ij} - 1 \right) \tag{3.38}$$

s.t (3.31) - (3.34)

The objective function (3.38) is re-written to obtain the following formulation for the relaxed problem with the lagrangean objective function:

Problem $P_{R_r}(\delta)$:

$$\min_{x \in R_r} \sum_{i=1}^{n} \sum_{j=1}^{n} \left(c_{ij} + \delta_i\right) x_{ij} - \sum_{i=1}^{n} \delta_i \tag{3.39}$$

s.t. (3.31) to (3.34)

Problem $P_{R_r}(\delta)$ defines an r-arborescence on G where the cost of an arc has been transformed using: $\hat{c}_{ij} = c_{ij} + \delta_i$, $\forall\ (i,j) \in N$. The optimal solution, $Z\left(P_{R_r}(\delta)\right)$ to the above problem forms a lower bound on the value of $Z\left(P_{TSP}\right)$. The maximum lower bound is obtained by obtaining a multiplier vector $\delta = \delta^*$ such that:

$$Z\left(P_{R_r}(\delta^*)\right) = \max_{\delta} Z\left(P_{R_r}(\delta)\right) \tag{3.40}$$

The iterative subgradient ascent method described in Chapter 2 can be used to solve the problem of maximizing the lagrangean dual, (3.40). At any stage m of the ascent, we solve problem $P_{R_r}(\delta)$ for a given set of multipliers $\delta = \delta^m$. If the r-arborescence corresponding to the transformed arc costs, $\hat{C} = \left[\hat{c}_{ij}\right]$, is a tour, the ascent is terminated and the corresponding lower bound, $Z\left(P_{R_r}(\delta)\right)$ is the optimal solution value to problem P_{TSP}. Otherwise, let d_i^m be the out-degree of node i. Let p be a positive scalar not greater than 2. The multipliers for the iteration m+1 of the ascent are updated using:

$$\delta_i^{m+1} = \delta_i^m + t^m\left(d_i^m - 1\right), \quad \forall\ i \in N \tag{3.41}$$

where t^m, the scalar step length, is computed using:

$$t^m = p * \frac{U - Z\left(P_{R_r}(\delta^m)\right)}{\sum_{i=1}^{n} \left(d_i - 1\right)} \tag{3.42}$$

If the optimal solution is not identified by the ascent, a set of multipliers δ^e is obtained which is a good approximation of δ^*.

For the 1-tree relaxation of the STSP, Held & Karp [1971] suggest that the optimal dual variables obtained by solving an AP with the STSP cost matrix can be used as the basis for obtaining a good set of starting lagrange multipliers. For the *r*-arborescence relaxation of problem P_{TSP}, we have a similar result.

Consider the dual of problem P_{AP}. It can be formulated as follows:

Problem D_{AP}:

$$\max_{u,v} \quad \sum_{i=1}^{n} u_i + \sum_{j=1}^{n} v_j$$

$$\text{s.t.} \quad c_{ij} - u_i - v_j \geq 0 \qquad \forall\ i,j \in N$$

Let the variables u_i^* and v_j^*, $i,j \in N$ define the optimal variables to problem D_{AP}. If we take the initial lagrange multipliers as $\delta_i^0 = -u_i^*$, $i \in N$ and evaluate the *r*-arborescence with respect to the transformed arc costs, the value of the lower bound obtained is:

$$\begin{aligned} Z\left(P_{R_r}(\delta^0)\right) &= \left(\sum_{(i,j) \in R_r} c_{ij} + \delta_i^0 \right) - \sum_{i=1}^{n} \delta_i^0 \\ &= \sum_{(i,j) \in R_r} c_{ij} + \sum_{i=1}^{n} \left(d_i^0 - 1 \right) \delta_i^0 \end{aligned}$$

From the constraints of the AP dual problem, and from initializing the lagrangean vector to the optimal row duals of the AP it follows that:

$$
\begin{aligned}
Z\left(\mathrm{P}_{\mathrm{R}_r}(\delta^0)\right) &\geq \sum_{(i,j)\in R_r} \left(u_i^* + v_j^*\right) + \sum_{i=1}^{n} \left(1 - d_i^0\right) u_i^* \\
&= \sum_{i=1}^{n} d_i^0 u_i^* + \sum_{j=1}^{n} v_j^* + \sum_{i=1}^{n} \left(1 - d_i^0\right) u_i^* \\
&= \sum_{i=1}^{n} u_i^* + \sum_{j=1}^{n} v_j^* \\
&= Z\left(\mathrm{P}_{\mathrm{AP}}\right)
\end{aligned}
$$

Thus by taking $\delta_i^0 = -u_i^*$, $\forall\ i \in N$, the initial lower bound is at least as great as $Z\left(\mathrm{P}_{\mathrm{AP}}\right)$. After this initialization, the multipliers are updated using (3.41). However, since the algorithm to identify the *r*-arborescence requires the cost of every arc to be nonnegative, we impose the following strong condition on the values of the multipliers:

$$
\delta_i^{m+1} = \max\left\{0,\ \delta_i^m + t^m\left(d_i^m - 1\right)\right\}, \qquad \forall\ i \in N \tag{3.43}
$$

Moreover, we found that the use of (3.43) slows down the rate of the ascent. So, we initialized the multipliers using:

$$
\delta_i^0 = -u_i^* + \rho * \left\{ \max_{(i,j)\in A} \left\{c_{ij}\right\} \right\}, \qquad \forall\ i \in N
$$

Note that the use of this as the initial multiplier vector does not affect the lower bounds obtained. We observed that quick convergence was achieved in the ascent by initializing the constant ρ to be 1.25.

Smith [1975] developed an implementation of the ascent for this problem and obtained bounds for the ATSP. We use the parameter updating rules and the ascent terminating rules given in Smith [1975] for our ascent.

3.3.2 LP duals of the minimal spanning arborescence problem

Consider the formulation of SSA_r, ie., problem P_R. The constraints (3.35) ensure the strong connectivity of the subgraph R which defines the solution to the problem. Since $c_{ij} > 0 \ \forall \ (i,j) \in A$, and if we assume $c_{ii} = \infty \ \forall \ i \in N$, then constraint (3.36) becomes redundant in the formulation. Also, constraint (3.37) can be relaxed to:

$$x_{ij} \geq 0, \qquad \forall \ (i,j) \in A. \tag{3.44}$$

This produces an LP formulation of the SSA_r (see Edmonds [1967] and Fulkerson [1974]) defined by the objective function (3.1) and the constraints (3.35) and (3.44). Let $K_t \equiv \left(S_t, \bar{S}_t\right) = \left\{(i,j) \middle| i \in S_t, \ j \in \bar{S}_t : \ r \in S_t\right\}$ denote an *r-cutset* or an *r-cut* in (3.35). If we denote the family of such *r*-cuts in G by K, then the dual of the SSA_r is represented as:

Problem D_R:

$$\max \quad \sum_{l \in K} y_l \tag{3.45}$$

$$\text{s.t.} \quad c_{ij} - \sum_{\substack{l \in K \\ (i,j) \in l}} y_l \geq 0, \qquad \forall \ (i,j) \in A, \tag{3.46}$$

$$y_l \geq 0, \qquad \forall \ l \in K. \tag{3.47}$$

The LHS of (3.46) represents the LP reduced costs on the arcs obtained through this formulation. Let y_l^* denote the optimal solution to problem D_R and let c_{ij}^* denote the optimal reduced costs. Then,

$$c_{ij}^* = c_{ij} - \sum_{\substack{l \in K \\ (i,j) \in l}} y_l^*, \qquad \forall \ (i,j) \in A \tag{3.48}$$

The following four necessary and sufficient conditions satisfy the optimality of x_{ij}^* and y_l^* for the problems P_R and D_R.

RSC1: the primal solution x_{ij}^* satisfies (3.35) and (3.44),

RSC2: $y_l \geq 0 \ \forall \ l \in K$ and,

the reduced costs are nonnegative for each $(i,j) \in A$,

RSC3: $c_{ij}^* = 0$ if $x_{ij}^* > 0$,

RSC4: $\sum_{(i,j) \in l} x_{ij}^* = 1$, for each $l \in K$ such that $y_l^* > 0$.

The algorithm of Edmonds [1967] is *dual feasible* in the sense that at each step of the algorithm, the dual feasibility conditions RSC2 are satisfied. The primal feasibility conditions are attained only at the last step.

3.4 Bounds from complementary duality

Consider the formulation of problem P_{TSP} given by (3.1), the constraints (3.2), (3.3), (3.8), (3.5) and (3.6). The AP substructure is defined by (3.1) and the constraints (3.2), (3.3) and (3.6). The *r*-arborescence substructure is defined by (3.1) and the constraints (3.8), (3.5) and (3.6) with only one constraint from (3.2) for $j = r$, which is included in the *r*-arborescence formulation. The formulations are, therefore, complementary with respect to the formulation of problem P_{TSP}.

The AP procedure identifies the optimal LP dual variables associated with the degree constraints (3.2) and (3.3), given by u_i^* and v_j^* respectively, $\forall \ i,j \in N$. The best set of lagrangean dual variables associated with the relaxed connectivity constraints

are estimated by the restricted lagrangean bounding procedures. This set of duals is given by ω_t^e, $\forall\ t \in T$. The r-arborescence procedure identifies the optimal LP dual variables, $y_l^*\ \forall\ l \in K$, associated with the connectivity constraints. The best set of lagrangean dual variables associated with the relaxed degree constraints is estimated by the r-arborescence ascent. This set of duals is given by $\delta_i^e\ \forall\ i \in N$. In this sense, the two relaxation approaches are complementary dual procedures for problem P_{TSP}. We use this complementary dual property of the two substructures to design a sequential algorithm (Algorithm CD-TSP) to tighten the lower bounds produced by either of the substructures.

3.4.1 Problem reduction through the AP reduced costs

Let $\bar{C}^* = \left[\bar{c}_{ij}^*\right]$ represent the AP reduced cost matrix at the end of the application of bounding procedure BC31. The lower bound on problem P_{TSP} is $V(BC31)$. Given an upper bound, U, on the value of the optimal solution, if an arc (i,j) satisfies:

$$V(BC31) + \bar{c}_{ij}^* > U, \tag{3.49}$$

then arc (i,j) is declared superfluous to the problem and can be removed from the problem graph. An arc is removed by simply setting its cost to a large positive integer.

If all the input costs are integers and if the feasible tour associated with the upper bound is available, then U can be reduced to $U-1$.

3.4.2 Problem reduction through the r-arborescence reduced costs

Given the optimal dual variables y_l^* associated with every $l \in K$, the reduced costs are calculated using (3.48). The reduced cost of an arc (i,j) in the problem P_{R_r} is also given by c_{ij}^*. In addition, the reduced costs of all the arcs entering node r are changed because of the additional arc in the solution to problem P_{R_r} defined by the constraint (3.33). The additional arc is identified by the subproblem (P3.2). The reduced costs of arcs entering node r are then given by:

$$c_{ir}^* = c_{ir} - \min_{\substack{i \in N \\ i \neq r}} \{ c_{ir} \}, \qquad \forall\ i \in N \tag{3.50}$$

If an arc (i,j) satisfies:

$$Z\left(P_{R_r}(\delta^e)\right) + \hat{c}_{ij}^* > U, \tag{3.51}$$

then arc (i,j) is declared superfluous to the problem and can be removed from the problem graph. In (3.51), $\hat{C}^*$ refers to the reduced costs obtained using (3.48) and (3.50) on the cost matrix transformed using: $\hat{c}_{ij} = c_{ij} + \delta_i^e$, $\forall\ (i,j) \in A$.

3.4.3 The complementary dual algorithm

A step by step description of algorithm CD-TSP is given below:

Input: Graph $G=(N,A)$; cost matrix C; upper bound U.

Step 1: Solve the AP on the cost matrix C. $\bar{x}$ represents the optimal AP solution; the lower bound is $Z(P_{AP})$.

If $\bar{x}$ is a tour, output $\bar{x}$ and $Z(P_{AP})$. STOP.

The restricted lagrangean bounding procedures BC1, BC3 and BC31 are applied to augment the lower bound, $Z(\mathrm{P_{AP}})$.

At the end of BC31, $V(BC31)$ denotes the best lower bound attained. $\bar{C}^*$ is the corresponding reduced cost matrix and G_0^e denotes the admissible graph.

The reduced costs are used in conjunction with an upper bound U to identify superfluous arcs using (3.49). If an arc (i,j) is identified as superfluous, the cost matrix is updated using $c_{ij} := c_{ij} + M$, where M is a large positive number. The original graph is thus reduced.

Step 2: Solve the r-arborescence on this reduced graph and perform the ascent. $Z\left(\mathrm{P_{R_r}}(\delta^e)\right)$ is the best lower bound at the end of the ascent and G_{R_r} is the subgraph consisting of arcs in the r-arborescence. $\hat{C}^*$ is the corresponding reduced cost matrix.

If G_{R_r} defines a tour, output G_{R_r} and $Z\left(\mathrm{P_{R_r}}(\delta^e)\right)$. STOP.

The reduced costs are used along with U to identify superfluous arcs using (3.51); the cost matrix is updated accordingly.

Step 3: If any of the arcs in $\bar{x}$ have been declared superfluous in Step 2, then the AP solution is no longer optimal. By definition, an increase in the lower bound from the AP relaxation is possible.

If no such arcs can be found, Go to Step 4.

The AP is again solved on the updated cost matrix.

If G_{R_r} defines a tour, output G_{R_r} and $Z\left(\mathrm{P_{R_r}}(\delta^e)\right)$. STOP.

$\bar{x}$, $V(BC31)$, $\bar{C}^*$ and G_0^e are recalculated.

(3.49) is used to identify more superfluous arcs.

Go to Step 6.

Step 4: If any of the arcs in G_0^e were declared superfluous in Step 2, then the admissible graph could become weakly connected.

If no such arcs can be found Go to step 5.

The bounding procedures BC1, BC3 and BC31 are applied to the updated reduced cost matrix. The new lower bound and reduced costs are evaluated.

(3.49) is used to identify more superfluous arcs.

Go to Step 6.

Step 5: Output the best lower bound given by: $\max\left\{V(BC31), Z\left(\mathrm{P}_{\mathrm{R}_r}(\delta^e)\right)\right\}$.

STOP.

Step 6: If any of the arcs in G_{R_r} have been declared superfluous, the *r*-arborescence is no longer optimal. By definition, an increase in the lower bound from the *r*-arborescence relaxation is possible.

If G_{R_r} defines a tour, output G_{R_r} and $Z\left(\mathrm{P}_{\mathrm{R}_r}(\delta^e)\right)$. STOP.

The best set of multipliers δ^e are used to evaluate a new solution G_{R_r} using the updated cost matrix. The augmented lower bound, $Z\left(\mathrm{P}_{\mathrm{R}_r}(\delta^e)\right)$ is evaluated.

(3.51) is used to identify superfluous arcs.

Go to Step 3.

The optimal solution to the AP obtained in Step 1 can be used, along with a modified version of the Hungarian algorithm, to reduce the complexity of re-solving an AP in Step 3 of algorithm CD-TSP from $O(n^3)$ to $O(n^2)$. Bellmore & Malone [1971] use this in their AP based branch and bound algorithm. An efficient implementation of

the modified Hungarian algorithm for re-solving APs over subsets in a branch and bound procedure is given by Carpaneto & Toth [1980]. This requires, on average, less than $O(n^2)$ steps. However, in Step 3, we use the $O(n^3)$ Hungarian algorithm because of the complexities introduced when more than one arc from the optimal solution to the AP (obtained in Step 1) is declared superfluous in Step 2.

We can tighten the bound further by incorporating branch-chord exchanges in the optimal r-arborescences (in Step 2 and Step 6) to identify indispensable arcs. Consider an arc (i,j) that is declared indispensable in Step 2. If $(i,j) \notin \bar{x}$, the AP solution is no longer optimal: the lower bound can be increased. However, we did not implement this idea and leave it as one possible avenue for further research.

3.5 Computational results

We use the same problems to test the effectiveness of algorithm CD-TSP as we used for the s-tree ascent procedure. All the sub-procedures within CD-TSP were coded in FORTRAN 77 and run on a CYBER/930.

The AP relaxation subprocedure is the first step of algorithm CD-TSP. Table 3.6 and Table 3.7 represent the results for the AP bounding procedures on 22 well-known symmetric (road map and Euclidean) problems and the 20 newly generated problems respectively. Computational studies in the past have indicated that lower gaps obtained from the AP with the STSP cost function are poor when compared to bounds that can be obtained from the AP with the ATSP cost function. The second column in each of Table 3.6 and Table 3.7 verify this claim. The initial AP lower gaps for

the well-known problems and the new problems are, on average, 23% and 25.5% from the optimal solution respectively. The reason for the poor performance can be explained by the tendency for the AP solution to retain the symmetry of the input costs. In other words, if $x_{ij}=1$, then, since $c_{ij}=c_{ji}$, the likelihood of $x_{ji}=1$ is also very high.

In the literature, computational studies on the performance of the restricted lagrangean approach to increase the AP lower bounds have been confined to ATSPs. Little has been reported for STSPs. Tables 3.6 and 3.7 report the performance of the restricted lagrangean approach for the STSP test problems.

The second column in each of the tables shows the percentage improvement over the initial AP lower bound obtained by procedure BC1. An average improvement of nearly 15.5% is possible by the application of this procedure. This substantial improvement can be attributed to the many subtours in the solution to the AP. Therefore, many violated connectivity constraints are identified by BC1. The procedure BC3 improves the lower bound at the end of BC1 by on average, 3.5% over all tested problems. The improvement obtained by the new procedure (BC31) on the lower bounds at the end of BC3 is an average of 0.7% over all the problems. The improvement is not as substantial as those obtained by BC1 and BC3 but is not unimportant when one is referring to the lower gap. One possible explanation for the modest increase in the lower gaps is that the admissible graph at the end of the application of BC3 is already quite strongly connected. Our studies have indicated that the small improvement obtained by BC31 is also instrumental in increasing the number of superfluous arcs by nearly 3.5%.

Table 3.6 Results for the AP bounding procedures on well-known problems

Problem	Initial AP Lower gap	BC1 (%Imp)	BC3 (%Imp)	BC31 (%Imp)	Final AP Lower gap
DF42	31.39	17.11	4.33	1.54	5.91
HK48	16.12	10.00	1.19	0.42	3.89
GR48	22.00	11.70	3.81	0.00	5.21
KT57	22.76	15.11	3.02	0.06	3.45
ST481	31.10	16.19	5.86	0.00	6.59
ST482	20.16	13.70	3.13	0.02	2.45
ST483	26.80	19.94	3.21	0.00	2.43
ST484	18.75	11.36	2.31	2.40	1.79
ST485	16.86	9.12	3.06	0.00	3.91
ST600	20.06	10.54	3.10	0.29	5.03
ST601	22.98	13.68	3.94	0.20	3.87
ST602	25.23	14.92	6.72	0.29	1.82
ST603	38.09	19.03	8.79	0.32	6.31
ST604	33.34	25.06	1.40	0.17	4.97
ST605	28.46	15.05	3.19	1.40	6.71
ST606	26.31	16.03	2.82	0.52	5.32
ST607	29.35	18.96	3.03	0.16	5.37
ST608	30.83	19.69	3.12	0.00	6.01
ST609	23.65	12.77	4.40	0.45	4.57
NCE50	11.55	5.77	0.99	0.49	3.91
NCE75	12.16	5.03	2.00	0.39	4.29
NCE100	9.81	5.08	0.50	0.17	3.81

The last columns in Tables 3.6 and 3.7 indicate the final lower gaps at the end of the application of the AP subprocedure in algorithm CD-TSP. These are, on average, 4.5% away from the optimal solution values for the well-known problems and 5.3% away from the optimal for the new problems. These lower gaps do not compare favourably with the very tight gaps attained by an *s*-tree ascent (Chapter 2). However, these bounds are useful since they are used to reduce the size of the problem. The reduction tests that are applied at the end of the AP subprocedure, result in an average of 59% of the available arcs being declared as superfluous. This reduction in the size of the problem is of importance in getting good lower bounds from the *r*-arborescence

ascent in Step 2 of CD-TSP. Without this reduction it is possible that many arcs may appear in r-arborescences that are redundant to the problem.

Table 3.7 Results for the AP bounding procedures on newly generated problems

Problem	Initial AP Lower gap	BC1 (% Imp)	BC3 (% Imp)	BC31 (% Imp)	Final AP Lower gap
KC500	36.08	26.08	2.43	0.41	4.94
KC501	21.65	13.91	1.54	0.48	4.67
KC502	22.55	12.18	3.38	1.44	4.17
KC503	32.06	22.13	4.82	0.39	2.76
KC504	19.72	11.24	3.45	0.13	3.89
KC650	26.61	16.64	4.28	0.00	4.10
KC651	19.52	9.98	4.59	0.00	3.91
KC652	21.22	12.90	3.48	0.24	3.51
KC653	27.53	17.83	2.07	1.21	4.76
KC654	15.72	7.21	3.28	0.30	4.19
KC750	23.75	14.07	2.38	0.65	5.29
KC751	20.06	13.18	2.62	0.19	3.17
KC752	34.22	17.59	4.24	1.93	7.43
KC753	36.35	19.31	3.49	0.58	9.80
KC754	32.10	20.54	1.16	0.42	7.89
KC1000	29.16	19.05	2.67	0.22	5.44
KC1001	28.53	15.75	3.32	1.70	5.67
KC1002	22.80	13.73	1.05	0.23	6.60
KC1003	26.72	15.61	1.29	0.46	7.71
KC1004	19.56	11.17	1.05	0.12	6.30

Table 3.8 and 3.9 show the results of the rest of the algorithm CD-TSP. The number of ascent iterations performed is given in the second column in each of the tables. An average of about 137 directed trees are evaluated by the ascent procedure. This figure is lesser than the number of iterations performed in the s-tree ascent (see Section 2.6.2). This is probably due to the manner in which we initialized the multiplier vector in the r-arborescence ascent to ensure quicker convergence. Although the

calculation of *s*-trees and *r*-arborescences have the same order of complexity, *s*-trees, on average, take less time to compute. Comparisons of the running times for the *r*-arborescence procedure (in Table 3.8 and Table 3.9) with that of the *s*-tree ascent (in Table 2.3 and Table 2.4) show that the former takes roughly the same time as the latter. The lower gaps obtained by the algorithm CD-TSP are however comparable to those obtained by the *s*-tree procedure. The average lower gap from algorithm CD-TSP for well-known problems is 0.57% while the average gap from the *s*-tree algorithm for the same set of problems is 0.56%. For the newly generated problems the average lower gaps produced by the two algorithms is 0.50% and 0.55% respectively.

Computational studies in the past have discouraged the use of the *r*-arborescence as a relaxation for the STSP. However, by using the complementary dual property of the AP and the *r*-arborescence relaxations, we are able to improve the lower bounds in a sequential manner. The result is that the final lower bounds are comparable to the tightest bounds possible for the STSP, ie., those from the *s*-tree relaxation. We attribute this result to the reduction tests which enable an increase in the lower bounds.

Table 3.8 and Table 3.9 also give the number of *cycles* of CD-TSP that are performed for each of the problems. A cycle is defined as the number of times either an AP (in Step 3) or an *r*-arborescence (in Step 6) are re-solved.

Table 3.8 Results for algorithm CD-TSP on well-known problems

Problem	# of *r*-arborescences	Number of cycles	CD-TSP Lower gap	% arcs superfluous	Cpu Secs for arborescence ascent	Cpu Secs for the AP subprocedure
DF42	88	1	0.34	93.75	43.49	0.62
HK48	116	1	0.16	96.15	62.94	0.86
GR48	120	0	1.76	83.30	70.61	0.85
KT57	149	1	0.41	89.92	99.14	1.34
ST481	128	0	1.88	86.19	71.18	0.87
ST482	112	1	0.18	95.52	60.89	0.87
ST483	108	1	0.06	96.33	66.95	0.83
ST484	124	1	0.66	91.99	77.20	0.82
ST485	112	1	0.16	95.70	62.52	0.82
ST600	144	1	0.22	94.97	127.96	1.56
ST601	164	0	0.96	91.41	149.05	1.54
ST602	125	0	Solved	_	139.06	_
ST603	162	0	1.04	88.02	130.75	1.55
ST604	144	1	0.16	95.66	101.22	1.56
ST605	156	1	0.85	91.58	103.61	1.52
ST606	132	1	0.03	97.79	104.46	1.45
ST607	150	1	0.43	93.42	139.68	1.53
ST608	160	0	1.12	89.74	137.28	1.52
ST609	156	1	0.79	91.58	129.96	1.54
NCE50	109	1	0.86	91.80	63.55	0.92
NCE75	162	0	0.32	89.85	188.54	2.82
NCE100	194	1	0.35	96.72	337.49	6.44

Finally, the tables also give the total number of available arcs declared superfluous by CD-TSP expressed as a percentage of the available arcs. The average number of arcs declared superfluous is 92.79% and 92.71% in the two problem sets respectively, thereby resulting in an average reduction of 92.75% over all the problems solved. This figure is marginally higher than the corresponding figure at the end of the branch-chord exchange procedure in the *s*-tree algorithm. This is a significant result that demonstrates the usefulness of the algorithm CD-TSP.

Table 3.9 Results for algorithm CD-TSP on newly generated problems

Problem	# of r-arborescences	Number of cycles	CD-TSP Lower gap	% arcs superfluous	Cpu secs for arborescence ascent	Cpu secs for the AP subprocedure
KC500	120	1	0.27	93.38	65.35	0.92
KC501	120	1	0.26	94.29	72.19	0.97
KC502	135	1	1.16	87.88	61.79	0.97
KC503	121	0	Solved	-	33.44	0.98
KC504	120	1	0.11	96.38	80.11	0.96
KC650	165	1	0.78	91.94	167.89	1.95
KC651	149	1	0.16	95.80	205.78	1.92
KC652	171	1	0.70	92.43	128.43	1.98
KC653	160	1	0.79	92.09	147.79	1.96
KC654	159	1	0.80	91.99	200.84	1.94
KC750	197	0	1.32	86.74	161.69	2.95
KC751	187	0	0.66	92.55	184.10	2.91
KC752	183	1	0.69	93.57	218.29	2.92
KC753	188	1	0.66	91.60	257.98	2.84
KC754	190	1	0.92	90.58	234.98	2.92
KC1000	239	1	0.34	95.68	597.97	6.39
KC1001	249	1	0.44	95.07	522.78	6.37
KC1002	219	1	0.13	97.13	891.62	6.56
KC1003	269	0	1.25	89.73	375.12	6.49
KC1004	279	0	1.56	85.38	516.63	6.29

3.6 Conclusions

In this chapter we have shown how the complementary nature of two substructures of the STSP - the AP and the r-arborescence - can be used in a sequential (lower-bound-augmenting) algorithm. Although both substructures are known to be weak for the STSP, we obtain lower bounds that are comparable to those produced by the best existing algorithm for solving STSPs. This encourages imbedding the bounds produced by this procedure in a branch and bound algorithm to solve STSPs.

In Section 3.4.3 we suggested, but did not implement, one possible avenue for further improvements. A procedure similar to the branch-chord exchange idea for the s-trees (explained in Section 2.5.1) can be used to identify indispensable arcs. Apart from reducing the size of the problem, this could also lead to an increase in the overall

lower bounds produced by CD-TSP. With a few modifications (due to asymmetric rather than symmetric graphs), the structural tests we used in the *s*-tree algorithm can be used to further decrease the size of the problem.

The algorithm is a general one in the sense that it is not confined for use on TSPs alone. Whenever complementary substructures are identified in a problem, the use of the complementary dual properties of these substructures can lead to an efficient bounding strategy within a branch and bound algorithm.

CHAPTER 4

A Branch and Bound Algorithm for the Symmetric Travelling Salesman Problem

4.0 Outline

In this chapter we describe a branch and bound algorithm for the STSP. At the root node of the search tree, we obtain lower bounds from the s-tree relaxation. Problem reduction tests identify superfluous and indispensable arcs. The graph G is then transformed to a graph G^T which is partially asymmetric. We give a formulation of the TSP defined on G^T. The complementary duality property of the AP and the r-arborescence is used to obtain lower bounds for the TSP defined on G^T. The depth-first branch and bound algorithm incorporates some new branching schemes. Throughout the algorithm, we perform simple tests on the structure of G^T to identify whether arcs in it could be declared superfluous or indispensable. Computational results are presented and the performance of the algorithm is discussed.

4.1 Lower bounds from the *s*-tree relaxation

Consider the complete symmetric graph $G=(N,A)$, where N is the set of n nodes and A is the set of m undirected arcs. The cost of an arc l is denoted by c_l, $\{l=1,\ldots,m\}$. An arc and its cost can also be represented as (i,j) and c_{ij} respectively. The STSP on the graph G is a tour consisting of n arcs from A such that the degree of every node is two. A formulation for this problem, denoted as P_{STSP}, is given by (1.1) and the constraints (1.2), (1.3), and (1.4).

In this chapter, we describe the stages of a branch and bound algorithm to solve problem P_{STSP}. Definitions and new notations will be introduced where necessary.

At the root node of the tree search, we perform an *s*-tree procedure. The various components of this procedure have been discussed in detail in Chapter 2. Below, we provide an outline of these components:

First, we use the *s*-tree substructure to obtain tight lower bounds on the value of the optimal solution of P_{STSP}. These lower bounds are obtained through a subgradient ascent. The bounds obtained at the end of the ascent are augmented by adjusting the values of the best multipliers obtained. An imbedded upper bound heuristic uses the *s*-trees generated in the ascent to produce tight upper bounds. The *s*-tree reduced costs obtained through branch-chord exchanges, the maximum *s*-tree lower bound and the upper bound enable a reduction in the size of the graph G. Further reductions in G are achieved using structural tests. As a result, on average, 90% of the available arcs are rendered superfluous, and 25% indispensable to the problem.

For problems that are not yet solved by the *s*-tree procedure, the graph that remains is sparse. We make use of this property in the reduced graph to apply a transformation which results in a partially asymmetric graph.

4.2 Transformation of the graph

Let $G_{SST}=(N, A_{SST})$ denote the subgraph of G which remains after the *s*-tree ascent procedure, where $A_{SST} \subset A$, is the set of available arcs. A few of the arcs in A_{SST} are indispensable to P_{STSP}. We transform the sparse (but still symmetric) graph G_{SST} into a partially asymmetric one. The transformed graph is denoted by $G^T=(N^T, A^T)$.

In the past, transformations have been applied to convert the STSP into ATSPs, which are easier to solve. The most effective of these is the transformation of Jonker, De Leve, Van der Velde & Volgenant [1980]. They suggest a method for obtaining lower bounds for the STSP by converting the symmetric graph into a partially asymmetric one. In their transformation, the STSP on $G=(N,A)$ is defined as the problem of determining the pair of paths of minimal total length from one to the other of two predetermined nodes $i, j \in N$, that contains all the other nodes $k \in N$, $k \neq i, j$. They call this problem the *two-routes problem.* Computational results show that an improvement of about 10% can be achieved using this transformation. However, since the AP lower bounds are known to be weak for the STSP, the lower gaps, even after the application of the two-routes transformation, are still approximately 10% away from optimal solution.

We apply a transformation which is based on the following proposition whose validity is obvious:

Proposition 4.1 The minimum cost Hamiltonian cycle in G, remains unchanged if the direction of one and only one arc in the graph G is fixed a priori.

In order to affect our transformation, we assume that the graph G_{SST} contains at least one chain of indispensable arcs. Let $C_i=\left\{x_1^i,\ldots,x_q^i\right\}$ with (arc) *cardinality* q_i, $\left(i=1,\ldots,p\right)$, denote the ith chain of indispensable arcs in G_{SST}. This chain can also be represented as a sequence of nodes $P_i=\left\{b_i,\ldots,e_i\right\}$, where b_i is an *entry node* and e_i is the *exit node* in the chain C_i.

The chain, whose cardinality q_1 is a maximum over all such chains of indispensable arcs, is denoted by $C_1=\left\{x_1^1,\ldots,x_r^1,\ldots,x_q^1\right\}$. From Proposition 4.1, we can fix the direction of at most one arc in G_{SST}. If we choose this arc to be $x_r^1 \in C_1$, the direction of all the arcs in C_1 can be fixed to the direction chosen for this arc, since all arcs in C_1 are indispensable. The longest chain of fixed arcs in G_{SST} is now a directed path.

The first stage of the transformation is the shrinking of this path to one *super node* representing both an *entry* (*beginning*) *node*, b_1 and an *exit node*, e_1. The *inner* indispensable arcs in the super node and the *inner nodes* that these arcs are incident at are ignored. As a consequence, the number of nodes in the graph G_{SST} is reduced to: $|N|-q_1$. All arcs in G_{SST} that leave the entry node b_1 and all arcs in G_{SST} that enter the exit node e_1 can be deleted.

For all the other chains of fixed arcs $C_i\ (i=2,\ldots,p)$ that have not been declared as super nodes we ignore the inner indispensable arcs and the inner nodes. The entire chain can be shrunk into two *special nodes*, b_i and e_i. An arc is introduced between these two special nodes. For a particular chain $C_i\ (i=2,\ldots,p)$, the choice of one of the terminal nodes to be designated as b_i is made arbitrarily. The other terminal node is denoted as e_i.

The remaining nodes of G_{SST} which are not a part of chains of fixed arcs are called *ordinary nodes.*

The graph that remains after the above changes are implemented to G_{SST} is the transformed graph, G^T. Due to the creation of a super node and special nodes, the size of G^T is less than that of G_{SST}. Moreover, it is also partially asymmetric. Initially, at the root node of the branch and bound search tree, only one super node is created. Subsequently, graphs that define subsets in the tree search consist of more such nodes. The degree of asymmetry in these graphs is reflected by the number of super nodes present in them. The method to increase the number of super nodes in subsets of the tree search will be presented in a later section.

We denote a super node $k \in N^T$, $k=1,\ldots,v$ as $V_k\langle b_k, e_k\rangle$, where b_k and e_k represent its entry and exit nodes. Over any subset of the tree search, the transformed graph, $G^T=(N^T, A^T)$, consists of subsets of nodes that are designated as:

- $N^v=\left\{V_1\langle b_1, e_1\rangle,\ldots, V_v\langle b_v, e_v\rangle\right\}$, the set of v super nodes;
- $N^{s1}=\left\{b_1,\ldots,b_s\right\}$ and $N^{s2}=\left\{e_1,\ldots,e_s\right\}$, two sets, each consisting of s special nodes. Given a chain of fixed arcs that is not represented by a

super node, one of the terminal nodes of this chain belongs to the set N^{s1}. The other terminal node is in N^{s2};

- N^{o}, the set of ordinary nodes.

The number of nodes in the graph G_{SST} is defined by:

$$|N^T| = |N| - \sum_{k=1}^{v} q_k - \sum_{i=2}^{p} (q_i - 1) \tag{4.1}$$

The arcs $(i,j) \in A^T$ and their costs, c_{ij}^T are defined as follows:

- The cost of an arc $(i,j) \in A^T$ such that node i is a super node, $i \equiv V_k\langle b_k, e_k \rangle \in N^v$, and j is not a super node, is given by the cost of an arc in G_{SST} from the exit node of i to j.

  For $i \equiv V_k\langle b_k, e_k \rangle \in N^v$,

$$c_{ij}^T = c_{e_k j} \qquad \forall\ i \in N^v \text{ and } j \in N^T \setminus N^v \tag{4.2}$$

- The cost of an arc $(i,j) \in A^T$ such that node i is not a super node and j is a super node, $j \equiv V_k\langle b_k, e_k \rangle \in N^v$, is given by the cost of an arc in G_{SST} from i to the entry node of j.

  For $j \equiv V_k\langle b_k, e_k \rangle \in N^v$,

$$c_{ij}^T = c_{i b_k} \qquad \forall\ i \in N^T \setminus N^v \text{ and } j \in N^v \tag{4.3}$$

- The cost of an arc $(i,j) \in A^T$ such that both i and j are super nodes, $i \equiv V_k\langle b_k, e_k \rangle \in N^v$ and $j \equiv V_l\langle b_l, e_l \rangle \in N^v$, is given by the cost of an arc in G_{SST} from the exit node of i to the entry node of j.

For $i \equiv V_k\langle b_k, e_k \rangle \in N^v$ and $j \equiv V_l\langle b_l, e_l \rangle \in N^v$,

$$c_{ij}^T = c_{e_k b_l} \qquad \forall\ i, j \in N^v \tag{4.4}$$

- The cost of an arc $c_{ij} \in G^T$ is zero if the nodes i and j are special nodes and if i and j are terminal nodes of the same chain of indispensable arcs.

For $i = b_k \in N^{s1}$, the kth element from the set N^{s1},

let $j = e_k \in N^{s2}$ be the corresponding kth element from the set N^{s2}

$$c_{ij}^T = c_{ji}^T = 0 \qquad \forall\ i \in N^{s1} \tag{4.5}$$

- The cost of all the other arcs $c_{ij} \in G^T$ is the same as their cost in G_{SST}.

For $i = b_k \in N^{s1}$ and $j = e_l \in N^{s2}, j \neq e_k$,

$$c_{ij}^T = c_{ji}^T = c_{ij} \qquad \forall\ i \in N^{s1}, j \in N^{s2} \tag{4.6}$$

For $i = b_k \in N^{s1}$ or $i = e_k \in N^{s2}$ and $j \in N^o$,

$$c_{ij}^T = c_{ji}^T = c_{ij} \qquad \forall\ i \in \{N^{s1}, N^{s2}\}, j \in N^o \tag{4.7}$$

For $i \in N^o$ and $j \in N^o$,

$$c_{ij}^T = c_{ji}^T = c_{ij} \qquad \forall\ i \in N^o \tag{4.8}$$

The transformation suggested above results from removing all the indispensable arcs from the graph G_{SST}. The cost of the indispensable arcs that have been removed from the graph G^T is given by:

$$C_F = \sum_{l=1}^{m} c_l \qquad \forall\ l \in A_{SST} : x_l = 1 \tag{4.9}$$

The cost C_F is subtracted from the upper bound U to produce an upper bound value on the TSP defined on G^T. Alternatively, any lower bound obtained for the TSP defined on G^T is augmented by the amount C_F.

4.2.1 An example

Consider the 15-node example of Figure 2.6. The figure, represents the graph, G_{SST}, that remains after the s-tree ascent and the reduction tests. G_{SST} consists of 19 arcs, 11 of which are indispensable. There are three chains of indispensable arcs:
$C_1 = \{x_{13,8}, x_{8,7}, x_{7,6}, x_{6,5}, x_{54}, x_{4,3}, x_{3,2}\}$, of cardinality $q_1 = 7$,
$C_2 = \{x_{10,9}, x_{11,10}\}$, of cardinality $q_2 = 2$, and
$C_3 = \{x_{1,15}, x_{15,14}\}$, of cardinality $q_3 = 2$.

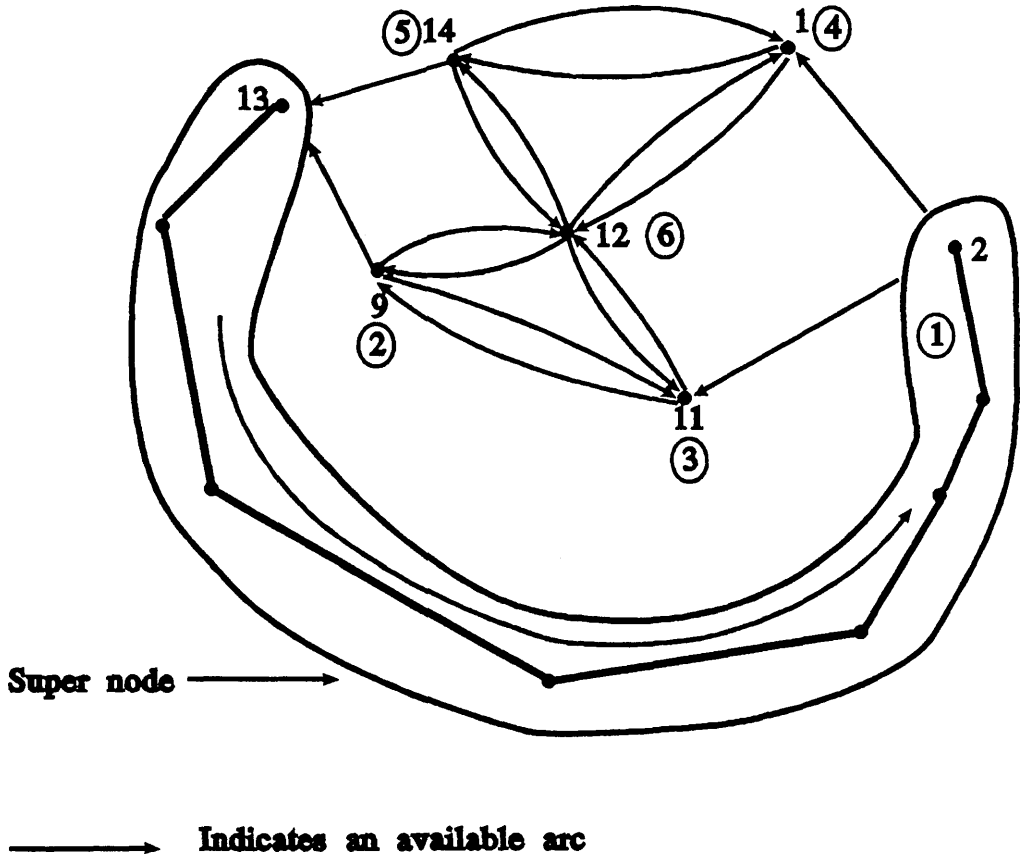
The chains can also be represented as the sequences of nodes: $P_1 = \{13, \ldots, 2\}$, $P_2 = \{9, \ldots, 11\}$ and $P_3 = \{1, \ldots, 14\}$.

The transformed graph G^T is constructed as follows: The longest chain of fixed arcs is C_1. We shrink this chain into a super node $V_1\langle 13, 2\rangle$, representing both an entry node, 13 and an exit node, 2. The inner indispensable arcs and the inner nodes of this chain are ignored. We delete all arcs that leave node 13, namely (13,14) and (13,9), and all arcs that enter node 2, namely (1,2) and (11,2). Two special nodes are created for each of the two remaining chains of fixed arcs. The nodes 9, 11, 14, and 1 are special nodes. The arcs (9,11), (11,9), (1,14) and (14,1), all of zero cost, are introduced between the special nodes (that are formed from the same chain). The

inner indispensable arcs and the inner nodes of these chains are ignored. Node 12 is an ordinary node.

The number of nodes in G^T is given by (4.1): $|N^T| = 15 - 7 - (1 + 1) = 6$. The sets of nodes that define the graph is given by: $N^v = \{V_1\langle 13, 2\rangle\}$, $N^{s1} = \{9, 1\}$, $N^{s2} = \{11, 14\}$ and $N^o = \{12\}$.

Figure 4.1 Transformed graph G_T of the 15-nodes example



The transformed graph G^T is shown in Figure 4.1. The encircled numbers represent the new node numbers of G^T.

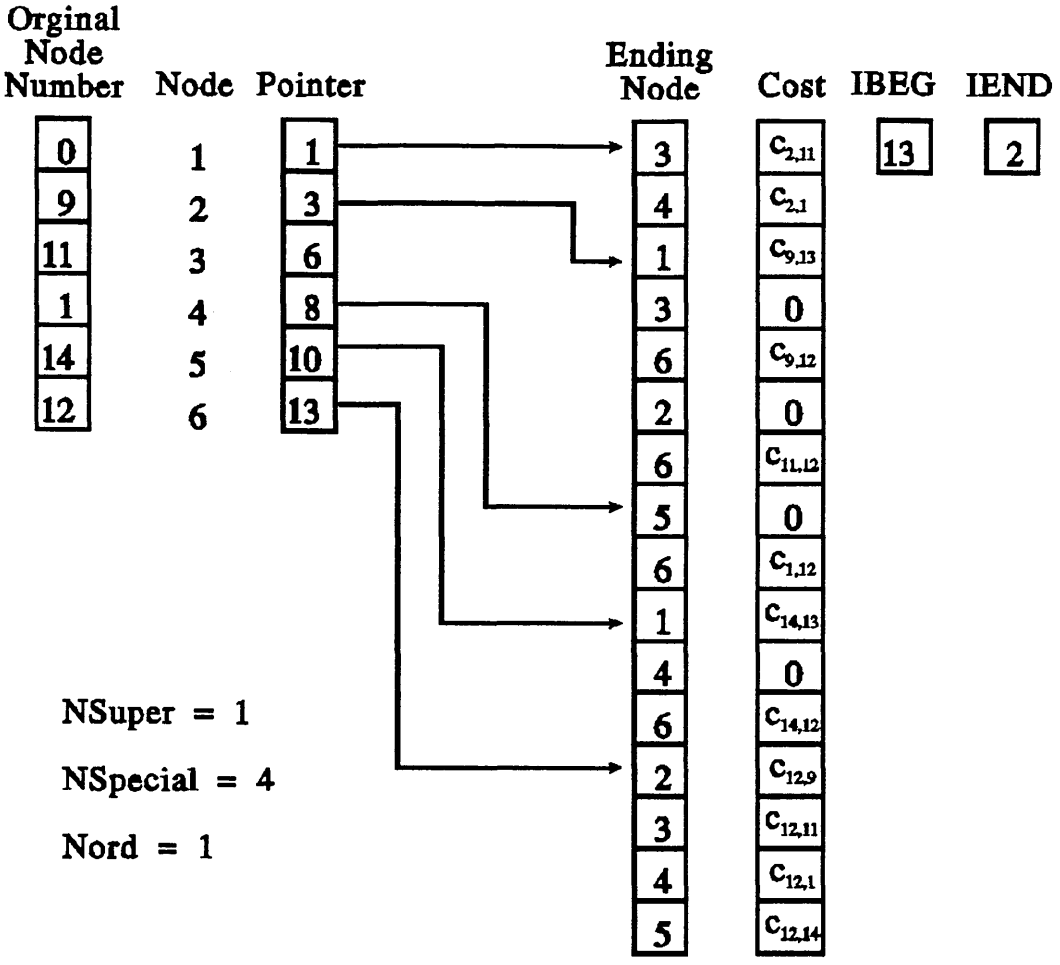
4.2.2 A simplified data structure

There are many conceivable advantages of the transformation described in Section 4.2.1. It results in a smaller graph. This in turn could lead to the generation of a fewer number of subproblems in a tree search algorithm. Moreover, the symmetric graph is transformed into a partially asymmetric one because of the direction of the fixed arcs in C_1. The transformation of the graph also paves the way for a simpler data structure to store the graph.

The sparse nature of transformed graphs enables the use of a *forward star structure* to store them. Dial, Glover, Karney & Klingman [1979] give a description of this method of storing sparse graphs (see also Gondran & Minoux [1984]). We illustrate this data structure through an example. Consider the transformed graph of Figure 4.1. The forward star structure of this graph is given in Figure 4.2.

The forward star of a node i is defined as the set of successors of i. The information corresponding to a node is stored by recording its forward star and the cost of the arcs it defines in two arrays (depicted in Figure 4.2 as the arrays Ending_Node and Cost). Associated with each node i is a pointer (contained in the array Pointer in Figure 4.2) which indicates the number of successive blocks of computer memory locations required for storing all the arcs in its forward star. The arcs in the forward star of node i appear immediately after the arcs in the forward star of node i-1. It follows that all the information relating to the node i is stored between the entries Pointer(i) and Pointer(i+1)-1 of the arrays Ending_Node(.) and Cost(.). Pointer(n+1) is initialized to be $|A^T|$.

Figure 4.2 Forward star data structure for the 15-nodes example



Three integers, *NSuper*, *NSpecial* and *NOrd* record the number of super, special and ordinary nodes in G^T. The original node indices (contained in the array Original_Node) are useful in the calculation of the cost of arcs in G^T. The cost of an arc is evaluated using an appropriate relation from (4.2) to (4.8).

If a node has index less than or equal to *Nsuper*, it is a super node in G^T. The entry and exit nodes associated with this super node are stored in two arrays IBEG and IEND whose lengths are equal to *Nsuper*. Initially, IBEG and IEND consist of only one element each. As more super nodes are created, the number of nodes, and the

size of the arrays Original_Node, Pointer, Ending_Node and Cost will decrease (as arcs and nodes will be deleted from G^T). The addition of each new super node will increase the dimension of IBEG and IEND by one.

The data structure is efficient and compact. Arc deletions and insertions, which will be necessary during the tree search, are easily achieved. Because of the large reductions achieved by the *s*-tree ascent, the amount of memory required to store the transformed graph is moderate.

4.2.3 Computational results for the graph transformation

In this section we highlight some of the characteristics of the transformed graphs of some of the test problems we used in the previous chapters. Table 2.3 and Table 2.4 show some of the characteristics of the reduced graphs G_{SST} of our test problems, in terms of the number of arcs declared indispensable. However, it is the number of arcs that form part of fixed chains which is of greater importance to the transformation.

Table 4.1 shows the number of undirected arcs in the complete graph G. The columns 5 to 8 of Table 4.1 indicate the main characteristics of the transformed graphs, namely, the cardinality of the longest chain of indispensable arcs, the number of chains of indispensable arcs, the number of nodes and the number of arcs in the transformed graph G^T.

Table 4.1 Results for the graph transformation

Problem	Original Size, $\|A\|$	Indispensable arc in G_{SST}	Available arcs in G_{SST}	q_1	p	$\|N^T\|$	$\|A^T\|$
DF42	861	28	57	13	6	20	65
HK48	1128	35	63	17	6	19	64
GR48	1128	5	197	2	3	46	380
KT57	1596	16	173	9	7	49	324
ST482	1128	31	61	19	4	21	64
ST484	1128	13	99	4	7	42	181
ST600	1770	43	76	13	11	28	84
ST601	1770	8	171	3	5	57	331
ST603	1770	11	233	1	10	59	454
ST605	1770	14	169	2	12	58	328
ST608	1770	12	175	3	9	57	339
NCE50	1225	4	117	2	2	48	224
NCE100	4950	24	199	6	13	89	372

q_1 : the cardinality of the longest chain of indispensable arcs in G_{SST}

p : number of chains of indispensable arcs

$|N^T|$: number of nodes in the transformed graph

$|A^T|$: number of arcs in the trasformed graph

4.3 Lower bounds from complementary duality

In Chapter 3 we gave a formulation for the TSP, defined on the graph $G=(N,A)$. Problem P_{TSP} is given by (3.1) and the constraints (3.2), (3.3), (3.4), (3.5) and (3.6). Constraints (3.7), (3.8) and (3.9) are alternate ways of formulating (3.4), the

connectivity constraints. With a few modifications, we can use the same formulation to define a TSP on the graph $G^T=(N^T,A^T)$. N, the set of nodes in that formulation, is replaced by N^T; A, the set of arcs, is substituted with A^T; c_{ij}, the cost of an arc (i,j), is replaced by c_{ij}^T, whose values are defined by one of the relations (4.2) to (4.8); n, the number of nodes, is replaced by $n^T=|N^T|$, whose value is given by (4.1). The node r in constraint (3.8) is taken to be the super node $V_1\langle b_1, e_1\rangle \in N^v$.

Consider an arc $(i,j)\in A^T$ such that $i=b_k\in N^{s1}$ and $j=e_k\in N^{s2}$. The nodes i and j are special nodes which are terminal nodes of C_k, the kth chain of indispensable arcs. The arc between the special nodes i and j in G^T represents a traversal along the C_k from b_k to e_k. Since, C_k represents a chain of indispensable arcs, at least one of the traversals, from b_k to e_k, or, from e_k to b_k should be present in an optimal solution to problem P_{TSP} defined on G^T. We ensure this by defining $c_{ij}^T=c_{ji}^T=0$. Thus, at least one of the arcs (i,j) or (j,i) is in the solution of problem P_{TSP} on G^T.

At the root node, after the application of the s-tree ascent and the graph transformation, lower bounds on the reduced problem are obtained from the procedure CD-TSP described in Chapter 3. If a feasible solution to the reduced problem is obtained by either the assignment subprocedure or the r-arborescence ascent, it is optimal for problem P_{TSP} on G. In this event, the set of arcs in the optimal solution is given by $\left\{\bar{x} \bigcup_{i=1}^{p} C_i\right\}$, if the feasible solution is obtained by the AP subprocedure, or, by $\left\{A_{R_r} \bigcup_{i=1}^{p} C_i\right\}$, if the feasible solution is produced by the r-arborescence ascent.

The value of the optimal solution is given by $Z\left(P_{AP}\right) + C_F$ in the former case and by $Z\ P_{R_r}\left(\delta^e\right)\Big) + C_F$ in the latter case.

If the problem is not solved at the root node, branching takes place. At any stage of the tree search, an appropriate branching scheme partitions the feasible set of the current subproblem into subsets. Lower bounds for each of the new subproblems that are generated are obtained from algorithm CD-TSP. Except for the starting set of multipliers associated with the nodes of the graph for the *r*-arborescence ascent, the procedure for obtaining lower bounds on the solution value of each of the subproblems in the tree search is exactly the same as that used at the root node. We do not distinguish, in any respect, between the *initial ascent* (at the root node) and the *general ascent* over any of the subproblems at a node of the tree search. The 1-tree based branch and bound algorithms of Smith & Thompson [1977] and Volgenant & Jonker [1982] use parameters to reduce the number of general ascent iterations. The reason for this is that if the initial ascent finds a good set of starting multipliers, a general ascent would require fewer number of ascent iterations to maximize the lower bounds on the subproblems. However, our branching schemes successively reduce the size of subproblems to be solved at nodes in the tree search (see the following section). Since the number of iterations of our ascent implementation is a function of problem size, we found that this causes the evaluation of a fewer number of *r*-arborescences in our general ascents.

At any node *k* of the tree search, the solution obtained from algorithm CD-TSP is either:

(a) feasible for the subproblem. In this case, the solution and its value are reported are reported. Again, the solution is represented either by $\left\{\bar{x} + \bigcup_{i=1}^{p^k} C_i^k\right\}$ or by $\left\{A_{R_r} + \bigcup_{i=1}^{p^k} C_i^k\right\}$, where $C_i^k \ (i = 1, \ldots, p^k)$ represents the chains of fixed arcs in the subproblem k. The value of this feasible solution is either $Z\left(\mathrm{P_{AP}}\right) + C_F^k$ or $Z\left(\mathrm{P_{R_r}}(\delta^e)\right) + C_F^k$, where C_F^k denotes the total cost of the fixed arcs in the subproblem k. If this value is less than the current best upper bound, U is updated accordingly.

(b) with an associated lower bound which is worse than U-1. In this case the subset is fathomed.

(c) with an associated lower bound less than U-1. In this case the current subset has to be further partitioned using an appropriate branching scheme.

Since all our test problems are integer valued, we use U-1 in the cases (b) and (c) above. The lower bounds, an upper bound, and the optimal reduced costs from the AP and r-arborescence substructures are used to reduce the size of all subproblems encountered during the tree search using (3.49) or (3.51). Again, the right hand sides of these relations are reduced to U-1. This may cause the remaining graph to be infeasible for the subproblem. As a result, the next computed bound could be greater than U-1. We then consider the subset fathomed. In our computational experiments, we observed that this occurs very frequently in all the problems that were solved.

The upper bounds we use throughout the tree search are those that are obtained at the root node using the heuristic described in Section 2.4. This is unless a better upper

bound is obtained during the tree search through the identification of a feasible solution whose value is less than U.

4.4 Computational results for the root node

In this section, we present the computational results for the application of the procedure CD-TSP to obtain lower bounds at the root node of the tree search. The procedure CD-TSP is applied to the transformed graph G^T. We apply the tree search to only some of the well-known problems from the literature. Table 4.2 shows the results obtained.

The number of ascent iterations performed is lesser than that performed on the complete graph, results for which are provided in Table 3.8. The lower gaps are very nearly the same. Some of the problems are infeasible because the branch-chord exchange tests within the s-tree ascent procedure or the problem reduction tests of CD-TSP removed arcs required to be in the optimal solution by taking $U \leftarrow U - 1$. In such cases the upper bound is the optimal solution. The table also gives the percentage of the available arcs in A^T declared superfluous. The time for the r-arborescence ascent is also given in this table.

Table 4.2 Computational results for CD-TSP at the root node of the tree search

Problem	# *r*-arborescences	AP Lower gap	Best *r*-arborescence Lower gap	% of available arcs removed	Cpu Secs for the *r*-arborescence subprocedure
DF42	49	6.39	0.63	9.23	1.38
HK48	43	7.13	0.33	3.125	1.12
GR48	119	4.86	1.76	1.05	69.89
KT57	16	4.70	0.00*	1.85	20.97
ST482	54	4.16	0.13	17.19	3.08
ST484	116	7.41	0.66	1.11	30.07
ST600	6	3.11	0.00*	11.90	3.8
ST601	158	7.10	0.95	10.88	128.97
ST603	146	8.46	0.99	1.32	124.29
ST605	147	9.42	0.85	7.62	89.27
ST608	_	0.00*	-	-	15.94
ST609	148	10.90	0.79	8.25	79.84
NCE50	104	3.91	0.87	1.78	99.35
NCE100	170	6.09	0.35	2.15	300.77

* : indicates that the problem is infeasible: Upper bound is the optimal solution in these cases

4.5 The branching schemes

The effectiveness of a branch and bound algorithm is assessed by the total number of subproblems that are solved. To this end, the use of effective branching schemes is essential. A "good" branching scheme is one that: (a) generates few successors from a node of the tree search, and (b) generates strongly constrained subproblems, ie., subproblems from which many solutions are excluded (see Balas & Toth [1985]).

Our branching schemes are used in conjunction with the solution of the AP obtained in Step 1 (or Step 3) of algorithm CD-TSP. Depending on the nature of the AP solution to the current subproblem, an appropriate scheme is selected from a hierarchy. Whenever a particular scheme fails, the next available scheme from the hierarchy is tried.

Traditionally, branching schemes for the TSP generate subproblems in the tree search that *require* or *forbid* certain arcs from the current subset of feasible solutions. For each node, k, of the tree search, two sets of arcs I_k and E_k denote the arcs that are respectively required in, and forbidden from, the solution of subproblem k. The symbol k which is used to represent a node of the tree search is, in general, a character string which represents the ancestry of the subproblem k. For example, if $k=(pqr)$, then k is the rth successor of (pq), which in turn is the qth successor to p. In terms of the variables x_{ij}, the subproblem at node k of the tree search is defined by (3.1), the constraints (3.2), (3.3), (3.4), (3.5), (3.6) and the following conditions:

$$x_{ij}=\begin{cases} 0, & \forall\ (i,j)\in E_k, \\ 1, & \forall\ (i,j)\in I_k, \end{cases} \tag{4.10}$$

At a node k, the AP and r-arborescence relaxations of the subproblem includes the constraints in (4.10) in addition to the constraints that define each of the substructures.

The subproblem at node k is denoted as $\mathrm{P}_{\mathrm{TSP}}^{k}$. Let $\mathrm{P}_{\mathrm{AP}}^{k}$ denote the AP relaxation of this subproblem and let $\bar{x}^{k}$ denote its optimal solution. If $\bar{x}^{k}$ is not a tour, S_k, the feasible subset of the current subproblem is partitioned into s subsets, $S_{i1},\ldots,S_{is}$, with the help of branching schemes such that $\bigcup_{j=1}^{s} S_{ij}=S_i$.

Most of the branching schemes used in AP based tree search algorithms for the solution of the TSP are based on the introduction of subtour patching or subtour breaking inequalities. Subproblems are generated by constraining the feasible space of the current subproblem in such a way that one of the subtours that appears in the solution to the current subproblem does not recur in any of its successors.

We now briefly describe the branching scheme of Bellmore & Malone [1971], which is one of the most effective schemes used in AP based tree search algorithms for solving the TSP.

Let $B_s=\{(i_1,i_2),\ldots,(i_t,i_1)\}$ be the arc set of a minimum cardinality subtour of $\bar{x}^k$. The set B_s can also be expressed as the node set $S=\{i_1,\ldots,i_t\}$. Define the set $B_s \setminus I_k=\{(i_1,i_2),\ldots,(i_s,i_{s+1})\}$, where $s \leq t$. The branching scheme of Bellmore & Malone [1971] is based on the following disjunction:

$$(x_{i_1 i_2}=0) \vee (x_{i_1 i_2}=1, x_{i_2 i_3}=0) \vee \ldots \vee (x_{i_1 i_2}=,\ldots, x_{i_{s-1} i_s}=1, x_{i_s i_{s+1}}=0),$$

where $s+1$ is taken to be modulo t.

Generate s successors of the tree node k, defined by the sets:

$$\left.\begin{aligned} E_{kr} &= E_k \cup \{(i_r,i_{r+1})\} \\ I_{kr} &= I_k \cup \{(i_1,i_2),\ldots,(i_{r-1},i_r)\} \end{aligned}\right\} \quad r=1,\ldots,s \qquad (4.11)$$

The first successor from the tree node k forbids the first arc from $B_s \setminus I_k$. The second successor requires the first arc but forbids the second arc, and so on until the last successor requires all but one arc but forbids the last arc in the subtour $B_s \setminus I_k$.

Carpaneto & Toth [1980] suggest a variation of this branching scheme by replacing the arc set B_s of a minimum cardinality subtour with that of a subtour containing the least number of *free* arcs (a free arc is one that does not belong to either I_k or E_k).

The branching schemes we use are also of the subtour breaking variety. However, we do not accomplish the subtour breaking strategy by merely requiring and forbidding arcs (by setting their values in the solution to the successor subproblems to 1 or 0 respectively). Our branching schemes can be seen as a system to increase the number of super nodes in graphs that define subproblems of the tree search. Denote the graph that define the subproblem at node k by $G_k^T = \left(N_k^T, A_k^T\right)$. N_k^T consists of the sets of super nodes, special nodes and ordinary nodes denoted as N_k^v, N_k^{s1}, N_k^{s2} and N^{k^o}.

At node k of the tree search, one of three schemes, BS1, BS2 or BS2 is selected. If $\bar{x}^k$, the optimal solution to problem P_{AP}^k satisfies certain properties, the first branching scheme is applied. If not, we identify whether $\bar{x}^k$ follows certain properties that facilitate the application of the second branching scheme. If not, the third branching scheme is applied. Arcs that are required to be in the solution to a subproblem are either converted to super nodes, with an entry and an exit point or, they are added to an existing super node. Thus, we only maintain the set E_k of arcs that are forbidden from the solution.

This hierarchy of schemes is described in the following sections.

4.5.1 Branching scheme BS1

This branching scheme identifies whether $\bar{x}^k$ contains any subtours of cardinality 2 between two special nodes. If such a subtour cannot be identified, this scheme cannot be applied.

Denote by $B_s^2 = \left\{\left(i_1, i_2\right), \left(i_2, i_1\right)\right\}$, a subtour of arc cardinality 2 in $\bar{x}^k$. If no such subtour exists in $\bar{x}^k$, this branching scheme cannot be applied. We implement the branching scheme BS3.

Of all the subtours of cardinality 2 that are present in $\bar{x}^k$, choose the one with $i_1 \equiv b_i$ and $i_2 \equiv e_i$. In other words, choose a subtour of cardinality 2 between the end (special) nodes, $b_i \in N^{s1}$ and $e_i \in N^{s2}$, of the same chain of fixed arcs, C_i.

If no such subtour exists in $\bar{x}^k$, we implement the branching scheme BS2.

If $\bar{x}^k$ consists of more than one such subtour, choose the one for which q_i, the inner arc cardinality of the fixed chain of arcs is a maximum. Ties are broken arbitrarily.

B_s^2 represents a traversal along a fixed chain of arcs, first along one direction and then back along the other. Only one such traversal can exist in the optimal solution to the TSP. Thus two successor subproblems can be generated from the node k. The first fixes the direction of traversal along C_i from b_i to e_i, while the second fixes the direction of traversal along C_i from e_i to b_i. The direction of traversal can be fixed through the creation of a super node.

Two successor subproblems are generated from node k. The sets of nodes that are altered in the graph that defines the respective subproblems are given by:

$$
\begin{aligned}
N_{k1}^{v+1} &= N_k^v \cup \left\{ V_{v+1} \left\langle b_i , e_i \right\rangle \right\} \\
N_{k1}^{s1} &= N_k^{s1} \setminus \left\{ b_i \right\} \\
N_{k1}^{s2} &= N_k^{s2} \setminus \left\{ e_i \right\}
\end{aligned}
$$

$$
\begin{aligned}
N_{k2}^{v+1} &= N_k^v \cup \left\{ V_{v+1} \left\langle e_i , b_i \right\rangle \right\} \\
N_{k2}^{s1} &= N_k^{s1} \setminus \left\{ b_i \right\} \\
N_{k2}^{s2} &= N_k^{s2} \setminus \left\{ e_i \right\}
\end{aligned}
$$

For the successor node $k1$, a super node is created with an entry in b_i and an exit in e_i. Thus, all arcs leading out of b_i and all arcs leading into e_i are removed from A_{k1}^T. Similar changes are made to A_{k2}^T. Since the two special nodes are shrunk into one super node, the number of nodes in the graph that defines the successor subproblems is reduced by one.

4.5.2 Branching scheme BS2

If $\bar{x}^k$ contains any subtours of cardinality 2 between ordinary nodes, this scheme is applied to convert the arc between the two nodes into a super node. A subtour, $B_s^2 = \left\{ \left(i_1 , i_2 \right), \left(i_2 , i_1 \right) \right\}$ is identified such that $\left\{ i_1 , i_2 \right\} \in N_k^o$. If $\bar{x}^k$ consists of more than one such subtour we choose the one which has the transformed arc cost $c_{i_1 i_2}$ maximum over all $\left(i_1 , i_2 \right)$.

Two successor subproblems are generated from node k. The first creates a super node with internal direction from $\left(i_1 , i_2 \right)$ while the second creates a super node with internal direction from $\left(i_2 , i_1 \right)$. The sets of nodes that are altered in the graph that defines the respective subproblems are given by:

$$N_{kl}^{v+1} = N_k^v \cup \left\{ V_{v+1} \left\langle i_1, i_2 \right\rangle \right\}$$
$$N_{kl}^{o} = N_k^o \setminus \left\{ i_1, i_2 \right\}$$

$$N_{k2}^{v+1} = N_k^v \cup \left\{ V_{v+1} \left\langle i_2, i_1 \right\rangle \right\}$$
$$N_{k2}^{o} = N_k^o \setminus \left\{ i_1, i_2 \right\}$$

This branching scheme is equivalent to fixing the variable $x_{i_1 i_2} = 1$ in the first successor subproblem from node k and the variable $x_{i_2 i_1} = 1$ in the second successor subproblem from node k. For the successor node $k1$, all arcs leading out of i_1 and all arcs leading into i_2 are removed from A_{kl}^T. Similar changes are made to A_{k2}^T. The number of nodes that defines the graph at each of these successor nodes is reduced by one.

4.5.3 Branching scheme BS3

Let $B_s^s = \left\{ \left(i_1, i_2 \right), \ldots, \left(i_t, i_1 \right) \right\}$ denote the arc set of minimum cardinality subtour of $\bar{x}^k$ such that the node $i_1 \equiv V_j$, $j \in \left\{ 1, \ldots, p^k \right\}$. Here $V_j \in N_k^v$ is one of the P^k super nodes in the subproblem at node k. We represent the set as $B_s^s = \left\{ \left(V_j, i_2 \right), \ldots, \left(i_t, V_j \right) \right\}$, where the node i_1 has been replaced by V_j. B_s^s can also be represented as the set of nodes, $S_s^s = \left\{ V_j, \ldots, i_t \right\}$. B_s^s represents a subtour of minimum cardinality which also contains at least one super node in it. At least one such subtour must exist in $\bar{x}^k$. Let $B_s^{s-} = \left\{ \left(V_j, i_2 \right), \ldots, \left(i_s, V_j \right) \right\}$ denote the *s free arcs* in the subtour, where $s \leq t$. The set of free arcs is obtained by removing from B_s^s arcs between special nodes that may be present in B_s^s. The branching scheme BS3 is based on the following disjunction:

$$\left(x_{V_j i_2}=0\right)\vee\left(x_{i_1 i_2}=1, x_{i_2 i_3}=0\right)\vee,\ldots,\vee\left(x_{i_1 i_2}=,\ldots,x_{i_{s-1} i_s}=1, x_{i_s V_j}=0\right).$$

Generate s successors of the node k.

The first successor is generated by removing arc $\left(V_j, i_2\right)$ from the graph that defines the current subproblem.

The second successor is obtained by increasing the cardinality q_j of the super node by one. The super node is altered to $V_j\left\langle b_j, i_2\right\rangle$, as the node i_2 is the new exit node of the super node. The node sets in N_{k2}^T are appropriately augmented. The next arc from B_s^{s-} is removed from the subproblem.

The process of generating any of the remaining successors is exactly the same. The sth successor consists of one super node which is denoted by $V_j\left\langle b_j, i_s\right\rangle$. The arc $\left(i_s, V_j\right)$ is removed from the graph defining the sth subproblem.

Consider the construction of the rth successor. The super node, $V_j\left\langle b_j, i_{r-1}\right\rangle$, constructed for the ($r$-1)th successor has to be augmented due to the requirement that $x_{i_{r-1} i_r}=1$. Two particular cases occur depending on the nature of the node i_r:

Node $i_r \equiv V_l \in N^v$: The node i_r is a super node defined by an entry node b_l and an exit e_l. It is easy to verify that the super node for the rth successor is given by $V_j\left\langle b_j, e_l\right\rangle$.

Node $i_r \equiv b_l \in N^{sl}$: The node i_r is the entry node of a fixed chain of arcs which is not yet designated as a super node. It is easy to verify that the super node for the rth successor is given by $V_j\left\langle b_j, e_l\right\rangle$. A similar result is true if the node

i_r is an exit node of a fixed chain of arcs which is not yet designated as a super node.

This branching scheme is similar to the branching scheme of Bellmore & Malone [1971]. The difference is that the super nodes successively reduce the size of the subproblem to be solved. Moreover, when the branch constraint for the creation of a subproblem involves forcing an arc between a super node to a terminal node of a chain of indispensable arcs (which is not yet designated as a super node) the entire chain is forced into the solution of the subproblem.

4.6 Structural tests and infeasible subproblems

Throughout the computations we perform simple structural tests to reduce the size of the subproblems:

T4.1: If a node i is connected to only one other node j, the two nodes can be shrunk into a super node $V_{p^{k+1}}\langle i, j \rangle$.

If node j (for example) is already a super node, the appropriate terminal nodes of the super node are altered to reflect a new terminal node and node i is removed from the graph. If node j (for example) is an end node (say b_l) of a chain of indispensable arcs, a new super node $V_{p^{k+1}}\langle i, b_l \rangle$ is created.

Apart from T4.1, we perform another test, similar to the test T2.4 of Chapter 2, to reduce the size of subproblems.

T4.2: Consider two super nodes $j \equiv V_j\langle b_j, e_j \rangle$ and $l \equiv V_l\langle b_l, e_l \rangle$. If the arc (i,j) is available, we test the effects of forcing it into the solution of the subproblem. A new super node is created. All arcs leaving node j and all arcs entering node l are deleted. If the remaining graph is rendered disconnected, the arc (j,l) must be in the optimal solution to the subproblem. The two super nodes are shrunk into one with the appropriate terminal nodes.

If these reductions result in a disconnected graph the subproblem is infeasible and the branch is considered fathomed.

4.7 Computational results for the tree search

In this section we report some initial results for the tests we carried out on the effectiveness of the tree search algorithm. Not all problems were tested. We only use the well-known problems from the literature to test our tree search algorithm. Table 4.3 reports the results. The tree search algorithm was coded in FORTRAN 77 and run on a CYBER/930.

A few of the problems were solved at the root node itself by either the *s*-tree procedure or the procedure CD-TSP applied to the transformed graph. Branching was unnecessary for these problems. For a few of the problems the procedure CD-TSP aplication at the root node lead to a lower bound that is greater than the upper bound minus one. This is because, we use U-1 as the value of the upper bound in the problem reduction tests that are carried out in the *s*-tree ascent procedure and in the

procedure CD-TSP. As a result, some of the arcs required in the solution may have been declared superfluous. In such cases the optimal solution. This vindicates our conclusion to Chapter 2 that, by re-calculating an s-tree at the end of the ascent procedure, more problems could be declared as solved if the newly calculated lower bound is greater than U-1.

Included in the table are the number of subproblems of the tree search that were needed to find the optimal solution. It is quite clear that the results are encouraging. Only 3 nodes were required to solve the 42-node road map problem, DF42. HK48, the easier of the two 48-node road map problem required only 5 tree nodes to find the optimal solution. GR48, the more difficult of the 48-node road map problem required the generation of only 55 subproblems in the tree search.

The table also provides figures for the number of subproblems that did not require the application of the procedure CD-TSP to identify lower bounds, because the structural tests identified them as being infeasible. The table also provides the total number of r-arborescences evaluated and the total Cpu secs required for the optimal solution to be found.

Table 4.3 Computational results for the tree search

Problem	# nodes in the tree search	# nodes infeasible through the structural tests	# of r-arborescences evaluated	Cpu Secs
DF42	3	0	64	2.71
HK48	5	2	85	2.79
GR48	55	11	980	590.76
KT57	Infeasible graph G_{SST}. Upper bound optimal.			
ST482	5	2	128	5.34
ST484	11	2	571	218.69
ST485	Solved at the root node			
ST600	Solved at the root node			
ST601	53	9	525	636.59
ST602	Solved by the s-tree ascent			
ST603	39	7	313	385.91
ST604	Solved by the s-tree ascent at the root node			
ST606	Solved by the s-tree ascent at the root node			
ST608	Infeasible graph G_{SST}. Upper bound optimal.			
NCE50	17	2	103	239.79
NCE100	11*	3	395	700.00*

* : Cpu Time Limit exceeded. Best lower bound 0.09% away from optimal solution

4.8 Conclusions

In this chapter we gave a simple transformation for a symmetric TSP into a partially asymmetric TSP. Bounds on this transformed problem are obtained from the complementary dual procedure described in Chapter 3. A new branching scheme which is a departure from established and often used ones was introduced. This, together with modifications of one other existing branching scheme, forms a hierarchical system of branching schemes which seems to work well in creating well constrained subproblems in the tree search procedure. Limited computational experience suggests that this scheme for branching, together with the tight bounds obtained from the procedure CD-TSP, provides promise for a good algorithm to solve STSPs.

CHAPTER - 5

A Heuristic Solution to the Travelling Salesman Problem in the plane using Fractal Geometry

5.0 Outline

Plane-filling fractal curves are mappings that take the unit interval into the plane. In this chapter we present fast and simple heuristic algorithms based on the Peano-Cesaro plane-filling fractal curve for solving large scale TSPs in the plane. The basic algorithm is enhanced by "generative" improvement techniques. Computational results are presented and the performance of the algorithms is discussed.

5.1 Introduction

The *Euclidean travelling salesman problem* (TSP) is the problem of constructing a minimal length tour through a given set of n points distributed anywhere in d-dimensional Euclidean space. In this chapter, we deal with the TSP in the plane where the n points are distributed in the 2-dimensional domain. The Euclidean TSP has been shown to be *NP*-complete (see Papadimitriou [1977]). Although recent approaches to solving the TSP have increased the size of problems that can be solved optimally, many problems of large size rely on heuristic techniques that provide near-optimal solutions. Karp and Steele [1985] give a probabilistic analysis of heuristics for the Euclidean TSP.

For the TSP in the plane Karp [1977] gave an effective patching heuristic. In this algorithm the unit square is partitioned into rectangular regions, each containing a few points. Optimal tours over the points in each region form subtours which are then patched to give a tour through all the points. Christofides [1976] gave a heuristic (that also applies to TSPs with the more general metric satisfying the triangle-inequality) which has a worst-case error of less than 50% from the optimal solution.

Consider the set of points $P=\{i \,|\, i \in \mathrm{U},\ i=1,\ldots,n\}$, defined over the unit square, $\mathrm{U}=\{(x,y) \,|\, 0 \le x \le 1;\ 0 \le y \le 1\}$. Define d_{kt} as the Euclidean distance and (k,t) as the link between the points k and t. Let $Q_{be}=\{b,\ldots,e\}$ denote a permutation of a chain commencing at point b and ending at point e.

In this chapter we present fast and simple heuristic algorithms based on a spacefilling fractal curve for solving large scale TSPs in the plane. The basic algorithm is enhanced by "generative" improvement techniques. Computational results are presented and the performance of the algorithms is discussed. The results show that the algorithms developed here can be quite useful in obtaining reasonably good upper bounds on the optimal solution value of very large-scale TSPs in a very short time.

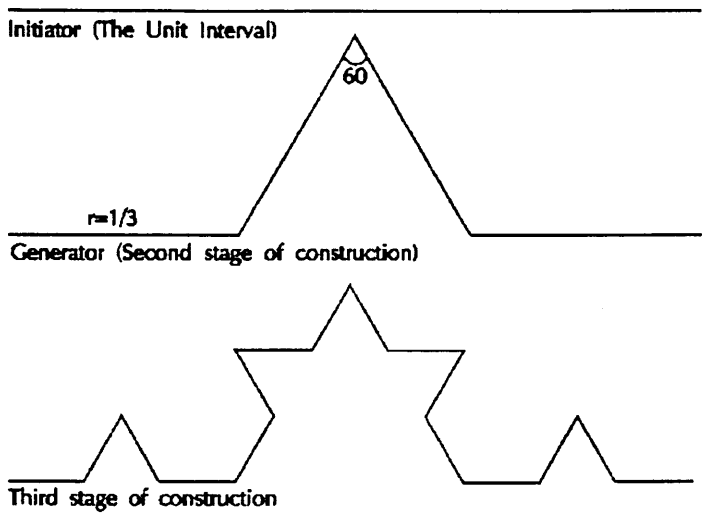
Spacefilling curves have been used in the past for the design of heuristics for combinatorial problems in the plane. Bartholdi and Platzman [1982] provide an $O(n \log n)$ heuristic for the planar TSP using the Sierpinski spacefilling curve, a curve that maps the unit interval continuously and recursively into the plane (see Abelson *et al.* [1982] for a description of this curve). Mandelbrot [1983] describes a whole family of such curves. Spacefilling curves have also been used in algorithms for weighted perfect matching, simple location problems, Steiner tree problems, etc. (see Iri *et al.* [1983], Imai [1986], Avis [1983], Bartholdi and Platzman [1983]). More recently, Platzman and Bartholdi [1989] produced worst-case bounds on the heuristic tour lengths produced by spacefilling curves. They also prove that the worst-case ratio of the heuristic tour length to the optimal tour length is $O(\log n)$. Bertsimas and Grigni [1989] show that this is a tight upper bound.

5.2 Some fractal definitions

The central idea behind a fractal is to find a recursive mapping that takes the unit interval into the plane. The key concepts in such constructions are *Initiator,*

Generator, and Sweep. An example of a fractal (the Koch Triadic Teragon) is illustrated in Figure 5.1.

Figure 5.1 The Koch-Triadic Fractal Pattern



If production of a fractal proceeds indefinitely, a trace that is everywhere continuous but nowhere differentiable would be obtained. The generator of a fractal consists of components, each of which are treated as scaled-down initiators for the next stage of construction. Patterns that replicate the generator as recursion deepens are called *self-similar* (Mandelbrot [1983]). For such fractals, *similarity Ratio* is defined as the ratio of the length of a component of the generator to that of the initiator. Self-similar fractals satisfy Mandelbrot's notion of *dimensionality* defined as $D = \frac{\log C}{\log (1/r)}$, where C is the number of components of the generator and r is the similarity ratio. Thus, the Koch curve, in which the generator consists of 4 components (C=4), each of which is a third in length of the initiator (r=1/3), has dimensionality D=log4/log3≈1.26186.

Fractals over the plane with D close to 1 are more smooth and well behaved than those with D close to 2 which are more plane-filling in nature.

Fractals with $D \leq 2$ do not cross themselves, a property that is also shared by the optimal TSP solution. Since the TSP in the plane involves n points distributed anywhere in U, an entirely plane filling fractal, ie., one with D=2 is required to "reach" all the points. We can now consider the Peano-Cesaro plane filling fractal pattern which we employ in this chapter for the TSP.

5.3 The Peano-Cesaro sweep: A plane filling fractal

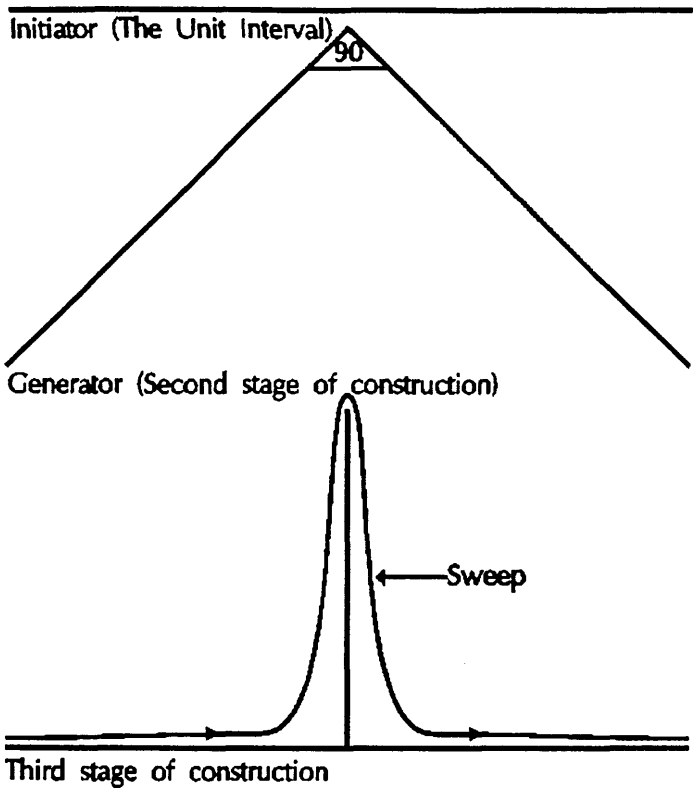
A *sweep* is a walk from the start to the end of the initiator along a definite path and this sweep defines the fractal. Taking the sides of the unit square as initiators (and the first stage of construction) the Peano-Cesaro sweep can be developed using the generator as shown in Figure 5.2 and the following rule of placement :

> At every even [odd] stage k>1, walk along the (k-1)th sweep and place the generator to the right [left] of each and every component.

Figure 5.3 shows four stages of development of the Peano-Cesaro fractal pattern. Sweeps in Figure 5.3 are indicated by directed paths all beginning and ending at a corner of the unit square. Since C=2 and $r = 1/\sqrt{2}$, using the above definition, we have D=2. Hence the plane filling property of the Peano-Cesaro fractal.

The Peano-Cesaro fractal *tiles* the unit square, U, with right angled isosceles triangles. At stage k a triangular tile is composed of the current initiator (forming the hypotenuse) and the components of the corresponding generator (as the other two sides).

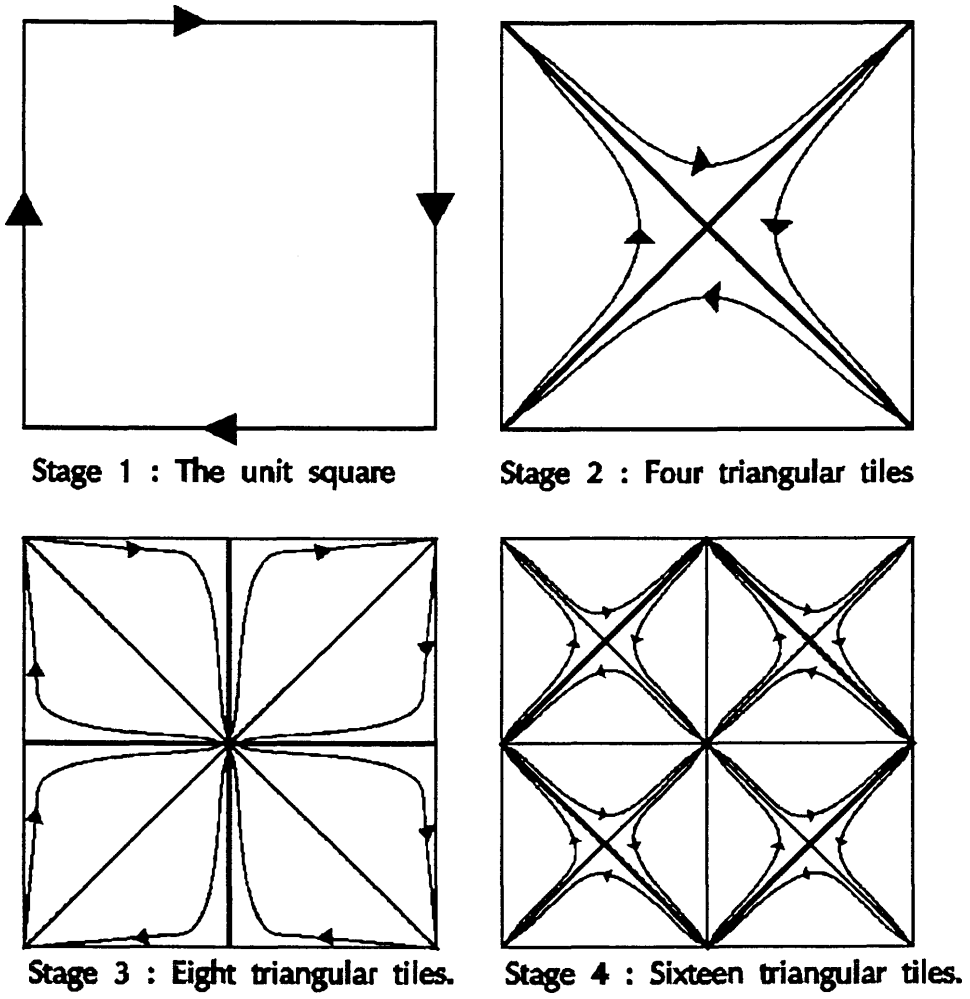
Figure 5.2 The Peano-Cesaro Fractal Sweep



The number of tiles is multiplied by a factor of 2 each time the sweep is advanced by one recursive stage, creating 2^k triangular tiles at the kth stage ($k>1$).

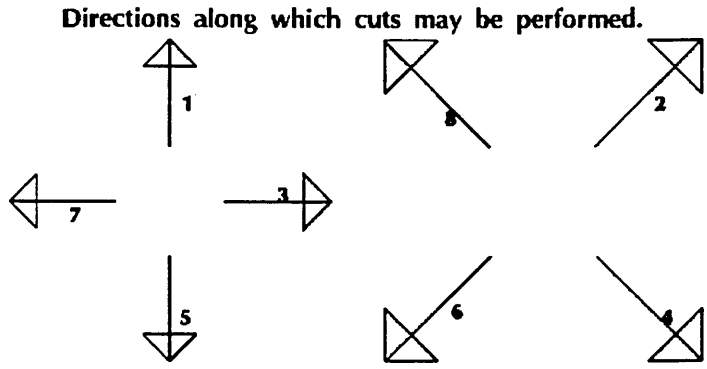
At an even [odd] stage of recursion the sweep cuts tiles U (initially) and the triangular tiles (subsequently) in one of the four possible directions shown in Figure 5.4a [Figure 5.4b]. The domain U is segmentable to any desired degree. The tiling is everywhere regular and isotropic and at every level of fragmentation each tile is visited by the sweep only once (in the sense that the sweep traverses the two sides of the tile defined by the components of the generator). These are properties that make the Peano-Cesaro sweep suitable (as a *divide and rule* strategy) for solving Euclidean TSPs.

Figure 5.3 A few stages of the Peano-Cesaro sweep



The faint lines indicate the previous sweeps
The arrows indicate the direction of the sweep
The thickened lines are the fractal cuts

Figure 5.4 Cuts at odd and even stages
Figure 5.4a **Figure 5.4b**



5.4 Description of the basic algorithm: TSPB

In this chapter we develop a basic algorithm called TSPB and three generative improvement procedures that have been added to enhance its performance.

The algorithm sets out to identify the sequence of points "collected" by the fractal sweep. It proceeds by considering one tile at a time. The tile under consideration at any stage is referred to as the *current tile.* The main step in the algorithm involves identification of points that belong to the current tile. Three possibilities occur: The current tile either contains no points, exactly one point, or more than one point. If it contains exactly one point, the sweep collects and adds it to the sequence of points that have already been collected. If it contains more than one point, further tiling results in 2 *sibling tiles.*

Initially, the unit square is cut along the diagonal (using the cut 2 in Figure 5.4b). Subsequently, at any step of the algorithm, if further tiling is required, cuts in one of eight possible directions shown in Figure 5.4a or Figure 5.4b are performed along the median (to

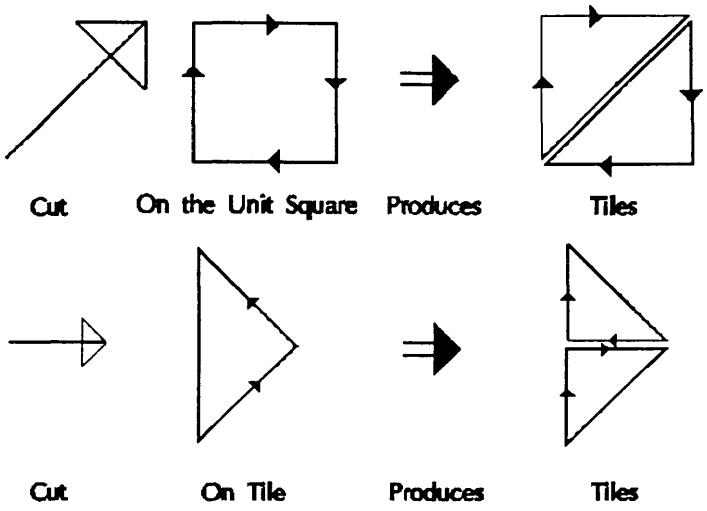
the hypotenuse) of triangular tiles. In these tiles the sweep has an entry and an exit point and its traversal of the sweep is along the two sides that subtend the right angle. Figure 5.5 shows the results of (i) the cut performed on the unit square and, (ii) a cut performed on a triangular tile at an intermediate step in the algorithm.

Let n_0 be a parameter that serves as a threshold on the minimum number of points contained in a tile before the sweep proceeds to the next tile. The algorithm can then be thought of as the sequencing of tiles having at most n_0 points in them. For large-scale TSPs, an open-ended TSP can then be solved for the points in a tile along with the entry and exit points of the sweep in the tile. The final tour is then constructed by patching together chains of at most n_0 points according to the order in which the chains were reached by the sweep.

The algorithms described in this chapter are for $n_0 = 1$. They can be generalized for $n_0 > 1$. Algorithm TSPB can be formally stated as follows:-

```
Step 0  The LIST of points to be collected by the sweep is empty
Step 1  If the fractal sweep has collected all n points
                STOP
Step 2  If the  current tile contains more than 1 point
                Cut the tile into two new sibling tiles
        Else If the current tile contains exactly 1 point
                Record the point in LIST
                Advance sweep to the next tile
        Else                                    /* No points in the tile */
                Advance sweep to the next tile
Step 3  Return to Step 2
```

Figure 5.5 Rules for Tiling



The algorithm can be seen as the processing of a stack that initially contains the unit square with n points. Each step of the computation involves checking for the number of points contained in the tile at the top of the stack. If there is exactly 1 point in the tile, record it in an array (which describes the TSP solution sequence) and remove the tile from the stack. If there are no points in the tile, remove the tile from the stack. If the tile contains more than one point, segment it into two and replace it in the stack with the two newly created tiles in such a way that the tile at the top is the one first reached by the sweep. Computation is halted when the stack is empty. Alternatively, the algorithm can be seen as a depth-first binary-tree. A parent tile at the level (k-1) gives rise to two siblings which are considered at level k. The leaf nodes of the tree consist of either empty tiles or tiles with exactly 1 point.

When a point (say t) is collected by the fractal the resulting chain of points already collected is represented as $Q_{bt}=(b,\ldots,t)$, where b is the first point collected. Q_{bt}

is referred to as the *current chain.* When the number of points in the chain is n, computation is halted. The TSP tour is then given by $Q_{bb}=(b,\ldots,s,b)$.

5.5 Generative improvements to algorithm TSPB

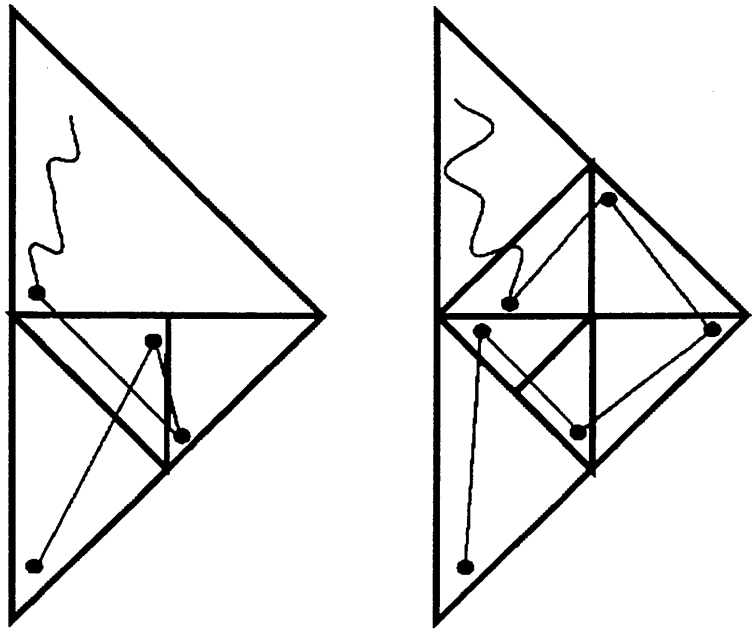
We tested the algorithm TSP on many randomly generated problems in the unit square. We observed that tours generated by TSPB contained, in the main, two undesirable features: (a) *Link crossings* and (b) *Expensive sub-chains.*

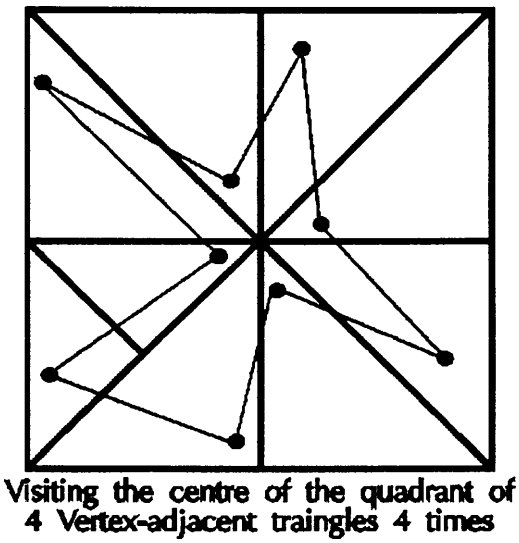
Although it is true that the fractal does not cross itself, tours that it produces could. Moreover, analysis of the tours produced by the sweep indicate that the detracting features stem from the predetermined nature of the cuts which have fixed orientations. We also noted that, at any stage in the algorithm, TSPB can produce a chain that visits the centre of the quadrant of 4 vertex-adjacent triangles as many as 4 times, thereby giving rise to expensive star-shaped tour patterns. Figure 5.6 illustrates these shortcomings. To remove them, we augment TSPB with a few generative improvement procedures which we now describe.

5.5.1 Procedure α: Intertile exchange of points that are close to the cuts

This procedure sets out to remove the "blindness" of the fractal cuts that could separate points close to each other, thereby leading to the formation of expensive sub-chains. Such close points can be grouped together so that they are swept collectively and not at different stages of the fractal sweep.

Figure 5.6 Shortcomings of Algorithm TSPB





We employ ideas from clustering to achieve this. At any level of recursion say $k-1$ ($k \geq 1$), let K_{k-1} denote the set of points contained in the current tile. If $|K_{k-1}| > n_0$, the tile has to be segmented into two sibling tiles at the next stage of recursion with a cut applied in the appropriate direction. (In TSPB, we took n_0 to be equal to 1). Let K_k^1 and $\bar{K}_k^1$ represent the sets of points contained in the two sibling tiles. Lay a

strip $D(\tau_k)$ of width τ_k around the cut. Construct the sets $M_k \left\{ i \middle| i \in K_k^1,\ i \in D(\tau_k) \right\}$ and $\bar{M}_k = \left\{ j \middle| j \in \bar{K}_k^1,\ j \in D(\tau_k) \right\}$. M_k and $\bar{M}_k$ represent the sets of points on either sides of the cut that fall within the strip in the two sibling tiles respectively. If either $M_k \neq \varnothing$ or $\bar{M}_k \neq \varnothing$, we evaluate either of the four possibilities:

1 Leave points within the strip intact. Compute the spread, ω_1 (see below for a definition of spread).

2 Transfer points $i \in M_k$ to $\bar{K}_k^1$ to produce the sets $K_k^2 = K_k^1 \backslash M_k$ and $\bar{K}_k^2 = \bar{K}_k^1 \cup M_k$. Compute the spread, ω_2.

3 Transfer points $j \in \bar{M}_k$ to K_k^1 to produce the sets $K_k^3 = K_k^1 \cup \bar{M}_k$ and $\bar{K}_k^3 = \bar{K}_k^1 \backslash \bar{M}_k$. Compute the spread, ω_3.

4 Exchange points $i \in M_k$ and $j \in \bar{M}_k$ to produce the sets $K_k^4 = K_k^1 \cup \bar{M}_k \backslash M_k$ and $\bar{K}_k^4 = \bar{K}_k^1 \cup M_k \backslash \bar{M}_k$. Compute the spread, ω_4.

Find the minimum of $\omega_1, \omega_2, \omega_3$, and ω_4 and choose the corresponding configuration as the one to be processed.

The **Spread** referred to in the four possibilities above is defined by:

$$\omega_i = \prod_{s \in K_k^l} \left\{ \left(X_c^l - x_s \right)^2 + \left(Y_c^l - y_s \right)^2 \right\}^{1/2} + \prod_{t \in \bar{K}_k^l} \left\{ \left(\bar{X}_c^l - x_t \right)^2 + \left(\bar{Y}_c^l - y_t \right)^2 \right\}^{1/2} \quad (5.1)$$

where $l = \{1,2,3,4\}$ represents the four possibilities set out above. For each of these possibilities $\left(X_c^l, Y_c^l \right)$ and $\left(\bar{X}_c^l, \bar{Y}_c^l \right)$ are the coordinates of the centres of gravity of points in K_k^l and $\bar{K}_k^l$ respectively. The spread in (5.1) is described as the sum of the

products of distances of the points that belong to a tile from their respective centres of gravity.

Transfer of a point, say t, from one tile to the other sibling tile involves the creation of a virtual point, $\hat{t}$, which is merely a mirror image of the point t about the cut. The introduction of $\hat{t}$ produces a chain that excludes t (by ignoring it) till the time when $\hat{t}$ is collected by the sweep.

Two criteria are used to estimate τ_k, the strip-width:

(i) The greater the recursion level k (ie., the smaller the tile size) the smaller the strip-width,

(ii) The lesser the number of points in the tile, the greater the strip-width.

After some experimentation, the following definition was found to be suitable: $\tau_k = \frac{2}{k} * \frac{\sqrt{A_k}}{m_{k-1}}$, where m_k and A_k respectively denote the number of points in and the area of the tile at the kth level respectively. Since $A_k = \frac{1}{2^k}$, the above expression can be re-written as: $\tau_k = \frac{1}{2^{k/2}} * \frac{2}{m_{k-1}} * \frac{1}{k}$.

The basic algorithm with this procedure added to it is called TSPB(α).

5.5.2 Procedure β: Generative improvements to the current chain

The objective of this recursive procedure, composed of two parts, is to investigate points in U that are spatially close to each other yet separated by the chain that is constructed by TSPB. The procedure is invoked each time a point is added to the current chain.

The first part investigates whether points that have not yet been collected by the fractal can be added to the current chain. When a point that is not in the chain is inserted between the two points that form the last link, an additional distance (an *extra mileage*) is incurred. Points that produce an extra mileage within a specific tolerance are examined as candidates for insertion. The point that produces the least extra-mileage within the allowed tolerance is inserted in the chain. This process is carried out for each of the new links created till no more insertions are possible.

The second part looks at points already belonging to the chain that fall within a neighbourhood around a new link in the current chain. A small family of links are exchanged to see if improvements (ie., shorter chain lengths) are possible. A greedy approach leads to accepting the configuration that corresponds to the maximal improvement.

The basic algorithm with this procedure added to it is called TSPB(β). Below, we present an algorithmic description of procedure β.

Step A: Let e be a point obtained in Step 2 of TSPB (Section 5.4). Denote the preceding point in the current chain Q_{be} by $e\_$. Insert the last link ($e\_$,e) into a stack consisting of links that need to be examined.

Step B: If the stack is empty, STOP.

Step C: Let (i,j) represent the first link in the stack. Q_{bi} is the current chain. We now examine the points in U that are *close* to (i,j) but not in Q_{bi}. A point $t \in$ U is defined to be close to (i,j) if: $d_{it} + d_{tj} \leq \delta d_{ij}$, where δ is an extra mileage tolerance.

After some statistical analysis of a sufficient number of problems we found that the best results were obtained for δ=1.185.

Form $I_j = \left\{ t \middle| t \in P,\ t \notin Q_{bi},\ d_{it} + d_{tj} \leq \delta d_{ij} \right\}$; the set of points that do not belong to the current chain but satisfy the extra mileage inequality.

Step D: If $I_j = \varnothing$, Go To Step G.

Step E: Identify the point $c \in I_j$ such that, $\left(d_{ic} + d_{cj}\right) = \min_{t \in I_j} \left\{ d_{it} + d_{tj} \right\}$. c is the point that produces the least extra mileage.

Two new links, (i,c) and (c,j) are created and inserted into the stack so that (i,c) is now at the top.

Step F: Go To Step C.

Step G: There exists no points $t \in P$, $t \notin Q_{bi}$ close to the link (i,j). The current chain $Q_{bi} = (b, \ldots, i)$ is updated to $Q_{bj} = (b, \ldots, i, j)$.

Step H: Let $d_{t(i,j)}$ be the euclidean distance between point t and the link (i,j) defined as $d_{t(i,j)} = \min_{q \in (i,j)} \left\{ d_{tq} \right\}$, where $q \in (i,j)$, $q \notin P$ represents all points in U that form the link (i,j).

Let $N' = \left\{ t \middle| t \in P,\ d_{t(ij)} \leq \varepsilon_{ij} \right\}$, for some $\varepsilon_{ij} > 0$.

We can simplify this by using the *chess-board* distance metric expressed as: $d^c_{t(ij)} = \min_{q' \in \{i,j\}} \left\{ \max \left\{ \left| x_{q'} - x_t \right|, \left| y_{q'} - y_t \right| \right\} \right\}$ to define the distance between a point t and link (i,j).

Define the set $N = \left\{ t \middle| t \in P,\ d^c_{t(ij)} \leq \varepsilon_{ij} \right\}$. N is the set of points in a rectangular neighbourhood around the last link (i,j) whose major and minor axes are of lengths $3\varepsilon_{ij}$ and $2\varepsilon_{ij}$ respectively. Clearly, $N' \subseteq N$. Let l^* denote the length of the optimal tour through n points distributed in the unit square. Beardwood *et al.* [1959] have proved (see also Eilon *et al.* [1971]) that $\lim_{n \to \infty} \frac{l^*}{\sqrt{n}} \approx 0.749$ with probability one. Hence, if l^*_n is the expected distance between two points in the optimal tour, then for large n, $l^*_n = \frac{l^*}{n} \approx \frac{0.749}{\sqrt{n}}$. We use this to define $\varepsilon_{ij} = \max \left(l^*_n,\ d_{ij} \right)$, thereby ensuring that all points in P at a distance of l^*_n or d_{ij} away from (i,j) are included in the set N.

Form $E_j = \left\{ t \middle| t \in Q_{bj},\ t \in N \right\}$; the set of points that belong to the current chain and the neighbourhood N.

Step I: If $E_j = \varnothing$, Go To Step B.

Step J: For any point $c \in E_j$, define c_ and c_+ as its preceding and succeeding points in the chain. Also define i_- as the point that precedes i. The chain can then be expressed as a sequence

$Q_{bj} = \left(b, \ldots, c_-, c, c_+, \ldots, i_-, i, j \right)$ which is permuted to obtain:

$Q^1_{bj} = \left(b, \ldots, c_-, c, i, i_-, \ldots, c_+, j \right)$,

$Q^2_{bj} = \left(b, \ldots, c_-, c, i, c_+, \ldots, i_-, j \right)$,

$Q^3_{bj} = \left(b, \ldots, c_-, c_+, \ldots, i_-, i, c, j \right)$,

$Q_{bj}^4=\left(b,\ldots,c_-,i,i_-,\ldots,c_+,c,j\right)$, and

$Q_{bj}^5=\left(b,\ldots,c_-,i,c,c_+,\ldots,i_-,j\right)$.

If $c=i_-$, a sixth permutation, $Q_{bj}^6=\left(b,\ldots,c_-,i,i_-,j\right)$ helps identify link crossings between (c_-,i_-) and (i,j).

In all these permutations the only points that undergo a change of position in the chain are i, i_-, and c_+.

Identify the permutation $\hat{Q}_{bj}\in\left\{Q_{bj},\ Q_{bj}^i,\ i=1,\ldots,6\right\}$ with minimum total chain distance.

Step K: If $\hat{Q}_{bj}=Q_{bj}$, Go To Step B.

Step L: Replace Q_{bj} by $\hat{Q}_{bj}$. Go To Step B.

5.5.3 Procedure γ: Eliminating link crossings

Each time a point is added to the current chain, a check can be carried out to identify if the last link results in crossings. If the number of crossings caused by the last link is greater than one, a greedy approach leads to the exchange of that pair of crossed links which produces maximum improvement. If the last link (e,e_-) crosses (i,i_+) in the chain $Q_{be}^1=\left(b,\ldots,i,i_+,\ldots,e_-,e\right)$, the positions of points i_+ and e_- are exchanged to produce $Q_{be}^2=\left(b,\ldots,i,e_-,\ldots,i_+,e\right)$. The length of Q_{be}^2 is then smaller than that of Q_{be}^1.

This process of investigating the last link for crossings is continued till no further improvements are obtained thereby serving to remove the crossings.

The basic algorithm with this procedure added to it is called TSPB(γ).

5.5.4 Algorithm TSPB($\alpha\beta\gamma$): The composite algorithm.

A composite algorithm includes all three procedures mentioned in this section and is called TSPB($\alpha\beta\gamma$). The basic algorithm TSPB produces the same solution irrespective of the vertex of U used as the starting point of computation. This is not the case with the generative improvement algorithms. For these algorithms, the minimal length tour obtained by four runs (each starting at one vertex of U) is declared as the best heuristic solution obtained.

5.6 Performance Analysis

In this section we analyze the complexity of the algorithms described. We also discuss the worst-case and probabilistic behaviour of the algorithms. We denote L and A as the worst-case and average-case tour lengths respectively and obtain upper bounds on their values.

These results have also been proved by Bartholdi and Platzman [1982], although in a different manner.

Theorem: The complexity of TSPB is $O\left(n \log_2 n\right)$.

Proof. If the n points are uniformly distributed in the plane, the maximum level of recursion required to get tiles with exactly one point in them is $\lceil \log_2 n \rceil$. At the kth level, $1 \le k \le \lceil \log_2 n \rceil$, there are 2^k tiles on average, each containing $n/2^k$ points. Thus, the order of complexity of TSPB is $\approx \sum_{k=1}^{\lceil \log_2 n \rceil} \sum_{j=1}^{2^k} \frac{n}{2^k} = n \lceil \log_2 n \rceil$.

Theorem: The complexity of TSPB(α) is $O\left(n \log_2 n\right)$.

Proof. If we consider strips of width $\tau_k = \frac{1}{2^{k/2}} * \frac{2}{m_{k-1}} * \frac{1}{k}$ along each of the fractal cuts at the kth level of recursion, the area of the strip is then given by $\sqrt{2}/nk$ and the expected number of points therein is $\sqrt{2}/k$. The order of complexity of the exchange algorithm at level k can then be expressed by: $\left(\frac{8n}{2^k} + \frac{\sqrt{2}}{k} \right) * 2^{k-1}$. The first term is the maximal order of complexity for the four possible exchange rules, each involving the calculation of the spread (defined earlier), applied to both tiles sharing the cut. The second expression is the maximal order of complexity for the transfer of points across the cut. Hence, the complexity of algorithm TSPB(α) is $\approx \sum_{k=1}^{\lceil \log_2 n \rceil} \left(4n + \frac{2^k}{k\sqrt{2}} \right) \approx C_1 \, n \log_2 n$, where C_1 is of order of magnitude 10.

Theorem: It can be shown that the complexity of the generative improvement algorithms TSPB(β) and TSPB(γ) are $O\left(n \log_2 n\right)$.

Corollary: The complexity of the composite algorithm TSPB(αβγ) is $O\left(n \log_2 n\right)$.

Theorem: In the worst-case, $L = 2\sqrt{n}$.

Proof. The largest tour that visits every fractal tile at the kth level is precisely the Peano-Cesaro sweep, where the points in each of the tiles visited by the sweep are assumed to be infinitesimally close to the vertices of the tiles. For such a distribution of points it can be shown that the worst-case tour length is:

$L = 2 * \left(2^{\lceil \log_2 n \rceil / 2}\right) = 2\sqrt{n}$. Note that here we assume that the length of the square enclosing all the points is 1.

Theorem: For large values of n, the average tour length, $A \rightarrow a\sqrt{n}$, where $0<a<1$.

Proof. If we assume that the n points are uniformly distributed over the unit square, the average tour connects the centres of each of the tiles produced. At the kth level one of the following recursions hold: $A_k = 2A_{k-2} - \frac{8}{3} * 2^{-k/2}\left(\sqrt{2}-1\right)$, where k is even ($k>2$), or $A_k = 2A_{k-2} + \frac{8}{3} * 2^{-k/2}\left(\sqrt{2}-1\right)$, where k is odd ($k>3$). For large k, using either of these results, the average tour length is the sum of a series and can be expressed as $A \approx \sqrt{2}\frac{\left(4+\sqrt{2}\right)}{9}2^{k/2}$. In the average case, since $k_{max}=\log_2 n$, we have $A \approx \sqrt{2}\frac{\left(\sqrt{2}+4\right)}{9}\sqrt{n}$ for $n>>2^4$.

5.7 Computational Experience

The algorithms described in Sections 5.4 and 5.5 were coded in FORTRAN 77 and run on a CYBER/960. The algorithms were tested on random problems in the plane and on a few well known problems for which the optimal solutions are known.

Table 5.1 summarizes the results for the randomly generated problems ranging from 50 to 5000 points. The entries in this table represent averages of 5 problems for each size. The results shown are for the basic algorithm, the three generative algorithms and the composite algorithm. The last column shows the performance of a three-opt algorithm with insertions. This is an algorithm with complexity less than the complexity of the three-opt heuristic (Lin [1965]). We call this heuristic TOWI. The

two values under each of the algorithms show the average value of the normalized tour lengths $l/\sqrt{n}$, and CPU times respectively for each problem. Since the optimal solutions are not available for these problems, we use the normalized lengths as a measure of comparison, where l is the tour length.

We note that TSPB(α) performs worse than TSPB(β), TSPB(γ) and even TSPB. A variety of compositions of the improvement procedures were considered including some that excluded procedure α. The overriding result is that the presence of procedure α enhances the performance of the composite algorithm.

TSPB(β) consistently outperforms the other improvement algorithms. The composite algorithm, however, dominates TSPB(β) - for the problems tested here an average improvement of at least 1.02% is obtained. This is obtained at an additional computational effort of $1\frac{1}{2}$ to twice as much. We therefore conclude that TSPB(β) is a reasonably efficient algorithm although a small improvement can be obtained at an extra computational effort.

We observe from the results that the normalized tour lengths produced by algorithm TOWI are better than those produced by TSPB($\alpha\beta\gamma$). However, the computational effort is considerably higher. Moreover, this algorithm does not lend itself to large-sized problems as the computational time becomes prohibitive.

The algorithms were also tested on some classical problems for which optimal solutions are known. The problems ST48 and ST60 represent the five 48-city examples and the ten 60-city examples respectively of Smith & Thompson [1977]. NCE50, NCE75, and NCE100 are the 50-city, 75-city, and 100-city examples

respectively of Christofides & Eilon [1969]. KFM100 represents three 100-city examples of Krolak, Felts, & Nelson [1972], known in previous literature (Lawler *et al* [1985] for example) as problems 24, 26, and 28 respectively. VJ100 stands for the ten 100-city examples of Volgenant & Jonker [1983]. LK318 is the 318-city example of Lin & Kernighan [1973] and LK105 is the 105-city problem extracted from it. GJ249 is the 249-city example of Gillett & Johnson [1976]. The optimal solutions for each of these problems are from the literature. For comparison purposes the algorithms developed here are tested against the performance of a three-opt algorithm with insertions. Table 5.2 presents these results.

Again we observe that TSPB(α) does not perform well in comparison to the other algorithms. However, as before, the composite algorithm, with procedure α included, produces best results. We also observe that procedure β is the best of the improvement ideas that we have suggested. The composite algorithm produces an average 1.03% improvement over TSPB(β) at an extra computational effort of about $1\frac{1}{2}$ times as much. The ratios that are produced by TOWI are superior to those produced by TSPB($\alpha\beta\gamma$) by about 1.05%. The running times, however, are quite large in comparison to TSPB($\alpha\beta\gamma$) and only medium-sized problems can be handled by algorithm TOWI.

Table 5.1 Results for randomly generated problems of 50 to 5000 points in the unit square

Problem	TSPB		TSPB(α)		TSPB(β)		TSPB(γ)		TSPB(αβγ)		TOWI	
n	$l/\sqrt{n}$	Cpu secs[a]	$l/\sqrt{n}$	Cpu secs[a]	$l/\sqrt{n}$	Cpu secs[a]	$l/\sqrt{n}$	Cpu secs[a]	$l/\sqrt{n}$	Cpu secs[a]	$l/\sqrt{n}$	Cpu secs[a]
50	0.970	0.073	0.944	0.094	0.875	0.228	0.907	0.169	0.841	0.327	0.824	7.561
75	0.943	0.085	0.974	0.121	0.841	0.411	0.854	0.313	0.837	0.637	0.795	23.652
100	0.917	0.103	0.929	0.153	0.830	0.639	0.843	0.510	0.820	1.027	0.787	73.728
250	0.941	0.216	0.949	0.458	0.830	4.680	0.852	4.354	0.822	8.309	0.776	1814.096
500	0.960	0.337	0.970	0.814	0.837	11.278	0.854	11.306	0.831	20.301	–	–
750	0.961	0.490	0.977	1.410	0.840	26.372	0.870	24.814	0.833	46.353	–	–
1000	0.955	0.648	0.965	2.088	0.829	45.400	0.865	44.419	0.823	80.959	–	–
2000	0.958	1.279	0.972	5.779	0.827	173.026	0.861	177.409	0.823	314.694	–	–
3000	0.962	1.955	0.974	11.029	0.827	384.524	0.863	399.418	0.824	702.501	–	–
4000	0.959	2.623	0.971	17.824	0.825	680.829	0.860	712.021	0.821	1580.637	–	–
5000	0.966	3.277	0.981	26.467	0.827	1887.858	0.868	1659.009	0.820	3445.789	–	–

[a] Seconds of CPU time on a CYBER/960.

– These problems could not be solved in reasonable computing time.

Table 5.2 Results for some well-known problems.

Problem		TSPB		TSPB(α)		TSPB(β)		TSPB(γ)		TSPB(αβγ)		TOWI	
Name	*n*	Ratio[b]	Cpu secs[a]	Ratio[b]	Cpu secs[a]	Ratio[b]	Cpu secs[a]	Ratio[b]	Cpu secs[a]	Ratio[b]	Cpu secs[a]	Ratio[b]	Cpu secs[a]
ST48*	48	1.195	0.070	1.217	0.093	1.059	0.258	1.078	0.158	1.045	0.359	1.007	5.972
NCE50	50	1.121	0.068	1.132	0.917	1.047	0.265	1.073	0.149	1.039	0.368	1.019	6.659
ST60*	60	1.225	0.078	1.215	0.106	1.075	0.360	1.127	0.217	1.066	0.508	1.018	27.025
NCE75	75	1.190	0.079	1.190	0.121	1.094	0.511	1.136	0.277	1.084	0.768	1.042	28.666
NCE100	100	1.181	0.091	1.184	0.149	1.073	0.848	1.132	0.489	1.073	1.224	1.040	68.973
KFM100*	100	1.399	0.099	1.350	0.157	1.194	0.792	1.253	0.502	1.108	1.169	1.005	58.853
VJ100*	100	1.250	0.099	1.239	0.157	1.102	0.793	1.432	0.514	1.084	1.184	1.022	66.541
LK105	105	1.474	0.114	1.426	0.183	1.159	0.973	1.314	0.619	1.086	1.369	1.001	84.318
GJ249	249	1.272	0.180	1.276	0.379	1.114	3.630	1.157	2.927	1.097	5.977	1.001	1562.059
LK318	318	1.479	0.267	1.484	0.547	1.217	6.555	1.255	5.187	1.165	10.298	–	–

[a] Seconds of CPU time on a CYBER/960.

[b] Ratio of the value obtained by the heuristic to optimal value

* Figures in these rows represent averages for several problems

– This problem could not be solved in reasonable computing time

5.8 Conclusions.

In this chapter we described how fractal curves can be used in an algorithm for the TSP. The improvement techniques deliver better solutions than those produced by the basic fractal algorithm. The results presented show that this method can be used for obtaining reasonable (but not excellent) solutions for very large problems in a very fast time.

We feel that the quality of the solutions produced by this algorithm can be greatly enhanced by improving procedure α. Currently, all points close to the fractal cuts are treated by a crude exchange mechanism. A more robust intertile exchange procedure ought to handle points that are close to the cuts on an individual basis. We feel that such a modified procedure will improve the quality of solutions produced by TSPB(α) and hence, TSPB($\alpha\beta\gamma$). The fractal algorithm is extremely fast. We can exploit this feature and use it as the basis for designing an algorithm that enables many runs, each yielding a different solution, to be performed in reasonable computational time.

CHAPTER 6

Conclusions

In this dissertation we developed exact and heuristic algorithms for the symmetric travelling salesman problem (STSP). We used three different substructures of this problem to obtain lower bounds on its solution. Two heuristic algorithms are developed. The first is based on fractal spacefilling curves. The second is based on extensions of two well-known heuristics for the STSP.

In Chapter 2 we considered the shortest spanning tree substructure within the STSP formulation. This relaxation produces very tight lower bounds for the problem and is, hence, widely used in exact algorithms for the solution of STSPs. Lower bounds on the solution are derived by maximizing the solution of the corresponding lagrangean dual problem using a subgradient ascent procedure. In our ascent, we implement the best features of existing approaches. These result in lower bounds that are better than those that are obtained through other existing approaches. Efficient problem reduction tests are used to yield considerable reductions in problem size. These tests are simple to apply and significant reductions are achieved.

We also used the assignment problem and the r-arborescence substructures to obtain lower bounds on the optimal solution to the STSP. We exploited the complementary nature of the two substructures to obtain very good lower bounds, obtained through

a sequential, lower-bound-augmenting procedure. Apart from the lower bounds, the procedure also yields significant reductions in problem size. Although both the complementary substructures used in this procedure (the assignment problem and the r-arborescence problem) are known to be weak relaxations for the STSP, the lower bounds obtained by the complementary dual procedure are comparable to those produced by even the s-tree relaxation. This, therefore points to a promising future for the use of this method to solve STSPs.

The complementary dual algorithm is a general one, in the sense that its use need not necessarily be confined to the TSP. Whenever complementary substructures are identified in a problem, we can exploit the complementary dual properties of these substructures to produce an efficient bounding strategy within a branch and bound algorithm.

In Chapter 4 we used the complementary dual algorithm in a branch and bound algorithm for the STSP. We suggested a transformation of the symmetric graph into an asymmetric one. We also used some new branching schemes to partition the set of feasible solutions of the subproblems. Initial results are encouraging.

Two heuristic ideas were investigated. One of the heuristics was described in Chapter 2. This algorithm is easily imbedded into the s-tree ascent procedure and produces very good upper bounds on almost all tested problems. A heuristic algorithm that uses the geometry of fractal spacefilling curves is also described (Chapter 5). The main advantages of this algorithm is that reasonable (although, not excellent) solutions are produced quickly for very large problems. The area offers wide scope for further research.

References

Aarts, E.H.L., and Korst, J.H.M. (1988). *Simulated annealing and Boltzman machines*. John Wiley, Chichester.

Abelson, H., and diSessa, A. (1982). *Turtle Geometry: The Computer as a Medium for Exploring Mathematics*, MIT Press, Cambridge, MA.

Adrabinski, A and Syslo, M.M. (1983). Computational experiments with some approximation algorithms for the travelling salesman problem, *Zastos Mat. 18*, 91-95.

Aho, A.V., Hopcroft, J.D., and Ullman, J.D. (1974). *The design and analysis of computer algorithms*, Addison-Wesley, Reading MA.

Akl, S. (1980). The minimal directed spanning subgraph for combinatorial optimization. *Australian Computing Journal 12*, 132-136.

Avis, D. (1983). A survey of heuristics for the weighted matching problem. *Networks 4*, 13, 475-493.

Balas, E. (1975). Bivalent programming by implicit enumeration. Belzer, J., Holzman, A.G., Kent, A. (eds). *Encyclopedia of Computer Science and Technology 2*, Dekker, New-York, 479-494.

Balas, E., and Christofides, N. (1981). A restricted Lagrangean approach to the travelling salesman problem. *Mathematical Programming 21*, 19-46.

Balas, E., and Guignard, M. (1979). Branch and bound/implicit enumeration. *Annals of Discrete Mathematics 5*, 185-191.

Balas, E., and Toth, P (1985). Branch and bound methods. Lawler, E.L., Lenstra, J.K., Rinooy Kan, A.H.G., and Shmoys, D.B. (eds.) (1985). *The Travelling Salesman Problem: A Guided Tour of Combinatorial Optimization*, Wiley, Chichester.

Bartholdi, J.J., and Platzman, L.K. (1982). An O(N log N) planar travelling salesman heuristic based on spacefilling curves. *Operations Research Letters 1*, No. 4.

Bartholdi J.J., and Platzman, L.K. (1983). A fast heuristic based on spacefilling curves for minimum-weight matching in the plane. *Information Processing Letters 17*, 177-180.

Bartholdi, J.J., and Platzman, L.K. (1988). Heuristics based on spacefilling curves for combinatorial problems in euclidean space. *Management Science 34*, 3, 291-305.

Bazaraa, M.S and Goode, J.J. (1977). The travelling salesman problem: A duality approach. *Mathematical Programming 13*, 1-13.

Beale, E.M.L. (1979). Branch and bound methods for mathematical programming systems. *Annals of Discrete Mathematics 5*, 201-219.

Beardwood, J., Halton, J.H., and Hammersley.J.M. (1959). The shortest path through many points. *Proceedings of the Cambridge Philosophical Society 55*, 299-327.

Beasley, J. (1984). An algorithm for the Steiner problems in graphs. *Networks 14*, 147-159.

Bellmore, M., and Malone, J.C. (1971). Pathology of travelling-salesman subtour-elimination algorithms. *Operations Research 19*, 278-307.

Berge, C. (1973). *Graphs and Hypergraphs*, North-Holland, Amsterdam.

Bertsimas, D., and Grigni, M. (1989). On the spacefilling curve heuristic for the euclidean travelling salesman problem. *Operations Research Letters 8*, 241-244.

Bock, F. (1971). An algorithm to construct a minimum directed spanning tree in a directed network, *Developments in Operations Research*, Gordon and Breach, New York, 29-44.

Bonomi, E and Lutton, J-L. (1984). The *N*-city travelling salesman problem: Statistical mechanics and the metropolis algorithm. *SIAM Review 26*, 4, 551-568.

Camerini, P.M., Fratta, L. and Maffioli, F. (1979). A note on finding optimum branchings. *Networks 9*, 309-312.

Carpaneto, G., and Toth, P. (1980). Some new branching and bounding criteria for the asymmetric travelling salesman problem. *Management Sci. 26*, 736-743.

Carpaneto, G., Dell'Amico. M., Fischetti, M. and Toth, P. (1988). *A branch and bound algorithm for the multiple depot vehicle scheduling problem*, Technical Report OR/87/2, DEIS, University of Bologna. (appeared in Networks, 1988)

Carpaneto, G., Fischetti, M. and Toth, P. (1989). New lower bounds for the symmetric travelling salesman problem. *Mathematical Programming 45B*, 233-254

Christofides, N. (1970). The shortest Hamiltonian chain of a graph. *SIAM Journal of Applied Mathematics 19*, 689-696.

Christofides, N., (1972). Bounds for the travelling salesman problem. *Operations Research 20*, 1044-1056.

Christofides, N., (1975). *Graph theory - An algorithmic Approach*, Academic Press, London.

Christofides, N. (1976). *Worst-Case Analysis of a New Heuristic for the Travelling Salesman Problem*, Report 388, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, PA.

Christofides, N. (1979). The travelling salesman problem. Christofides, N., Mingozzi, A., Sandi, C. (eds). *Combinatorial Optimization*, Wiley, Chichester, 131-149.

Christofides, N., and Eilon, S. (1969). An algorithm for the vehicle-dispatching problem. *Operations Research Quarterly 23*, 511-518.

Christofides, N., Mingozzi, A., Sandi, C. (eds.). *Combinatorial Optimization*, Wiley, Chichester, 131-149.

Chu, Y.J. and Liu, T.H. (1965). On the shortest spanning arborescence of a directed graph, *Sci. Sinica 14*, 1396-1400.

Clarke, G., and Wright, J.W. (1964). Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research 12*, 568-581.

Cook, S.A. (1971). On the complexity of theorem-proving procedures. *Proceedings of the 3rd Annual ACM Symposium on the Theory of Computing*, 151-158.

Crama, Y., Kolen, A.W.J., Oerlmans, A.G. and Spieksma, F.C.R. (1989). Throughput rate optimization in the automated assembly of printed circuit boards. *Technical Report RM89.034*, Faculty of Economics, Limburg University, Maastricht.

Crowder, H., and Padberg, M.W. (1980). Solving large-scale symmetric travelling salesman problems to optimality. *Management Science 26*, 495-509.

Crowder, H., Johnson, E.L., and Padberg, M.W. (1983). Solving large-scale zero-one linear programming problems. *Operations Research 31*, 803-834.

Dantzig, G.B., Fulkerson, D.R., and Johnson, S.M. (1954). Solution of a large-scale travelling salesman problem. *Operations Research 2*, 393-410.

Dial, R., Glover, F., Karney, D. and Klingman, D. (1979). A computational analysis of alternative algorithms and labelling techniques for finding shortest path trees. *Networks 9*, 215-248.

Dijkstra, E.W. (1959). A note on two problems in connection with graphs. *Numer. Math. 1*, 269-271.

Eastman, W.L. (1956). *Linear Programming with Pattern Constraints.*, Ph.D. thesis, Harvard University, Cambridge, MA.

Edmonds, J. (1965a). Paths, trees, and flowers. *Canadian Journal of Mathematics 17*, 449-467.

Edmonds, J. (1965b). Minimum partition of a matroid into independent subsets. *J. Res. Nat. Bur. Standards 69B*, 67-72.

Edmonds, J. (1967). Optimum branchings. *J. Res. Nat. Bur. Standards 71B*, 233-240.

Eilon, S., Watson-Gandy, C.D.T., and Christofides, N. (1971). *Distribution Management: Mathematical Modelling and Practical Analysis*, Griffin, London.

Fischetti, M. and Toth, P. (1988a). An additive approach for the optimal solution of the prize-collecting travelling salesman problem. Golden, B.L. and Assad, A.A (eds.) *Vehicle Routing: Methods and Studies*, North-Holland, 319-343.

Fischetti, M. and Toth, P. (1988b). An efficient algorithm for the min-sum arborescence problem. *Technical Report OR-88-4, DEIS*, University of Bologna. (*appeared in ACM Transactions on Mathematical Software*).

Fischetti, M. and Toth, P. (1988c). An additive bounding procedure for the asymmetric travelling salesman problem. *Technical Report, DEIS*, University of Bologna (appeared in *Mathematical Programming*).

Fisher, M.L. (1981). The lagrangean relaxation method for solving integer programming problems. *Management Science 27*, 1-18.

Fisher, M.L., and Jaikumar, R. (1981). A generalized assignment heuristic for vehicle routing. *Networks 11*, 109-124.

Fleischmann, B. (1982). *Linear Programming Approaches to Travelling Salesman and Vehicle Scheduling Problems.* Presented at the XIth International Symp. Mathematical Programming, Bonn, August 1982.

Ford, L.R., and Fulkerson, D.R. (1962). *Flows in Networks*, Princeton University Press, Princeton, NJ.

Frieze, A.M., Galbiati, G., and Maffioli, F. (1982). On the worst-case performance of some algorithms for the asymmetric travelling salesman problem. *Networks 12*, 23-39.

Fulkerson, D.R. (1974). Packing rooted cuts in a weighted graph. *Mathematical Programming 6*, 1-13.

Gabow, H.N. (1977).. Two algorithms for generating weighted spanning trees in order, *SIAM Journal on Computing 6*, 139-150.

Gabow, H.N., Galil, Z., and Spencer, T.H. (1984). Efficient implementation of graph algorithms using contraction. Proc. *25th Annual IEEE Symp. Foundations of Computer Science*, 347-357.

Gabow, H.N., Galil, Z., Spencer, T. and Tarjan, R.E. (1986). Efficient algorithms for finding minimum spanning trees in undirected and directed graphs. *Combinatorica 6*, 109-122.

Garey, M.R., and Johnson, D.S. (1979). *Computers and intractability: A guide to the theory of NP-completeness*, Freeman, San Francisco.

Garfinkel, R.S. (1979). Branch and bound methods for Integer Programming. Christofides, N., Mingozzi, A., Sandi, C. (eds.). *Combinatorial Optimization*, Wiley, Chichester.

Garfinkel, R.S. (1985). Motivation and modeling. Lawler, E.L., Lenstra, J.K., Rinooy Kan, A.H.G., and Shmoys, D.B. (eds.) (1985). *The Travelling Salesman Problem: A Guided Tour of Combinatorial Optimization*, Wiley, Chichester.

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

Gavish, B., and Srikanth, K.N. (1983). *Efficient Branch and Bound Code for Solving Large Scale Travelling Salesman Problems to Optimality*, Working paper Qm8329, Graduate School of Management, University of Rochester.

Geoffrion, A.M. (1974). Lagrangean relaxation for integer programming. *Mathematical Programming Studies 2*, 82-114.

Gillett, B.E., and Johnson, J.G. (1976). Multi-terminal Vehicle-dispatch algorithm. *Omega, 4*, 711-718.

Gilmore, P.C., and Gomory, R.E. (1964). Sequencing a one state-variable machine: a solvable case of the travelling salesman problem. *Operations Research 12*, 655-679.

Golden, B.L. (1977). Evaluating a sequential vehicle routing algorithm. *AIIE Transactions 9*, 204-208.

Golden, B.L., Bodin, L.D., Doyle, T., and Stewart, W. (1980). Approximate travelling salesman algorithms. *Operations Research 28*, 694-711.

Golden, B.L. and Stewart, W.R. (1985). Empirical analysis of heuristics. Lawler, E.L., Lenstra, J.K., Rinooy Kan, A.H.G., and Shmoys, D.B. (eds.) (1985). *The Travelling Salesman Problem: A Guided Tour of Combinatorial Optimization*, Wiley, Chichester.

Golden, B.L. and Skiscim, C. (1986). Using simulated annealing to solve routing and location problems.

Gondran, M. and Minoux, M. (1984). *Graphs and Algorithms*, Wiley, Chichester.

Grötschel, M. (1977). The monotone 2-matching polytope on a complete graph. *Operations Research Verfahren 24*, 72-84.

Grötschel, M. (1980). On the symmetric travelling salesman problem: solution of a 120-city problem. *Mathematical Programming Studies 12*, 61-77.

Grötschel, M., and Padberg, M.W. (1979a). On the symmetric travelling salesman problem I: inequalities. *Mathematical Programming 16*, 265-280.

Grötschel, M., and Padberg. M.W. (1979b). On the symmetric travelling salesman problem II: lifting theories and facets. *Mathematical Programming 16*, 281-302.

Grötschel, M., and Padberg, M.W. (1985). Polyhedral Theory. Lawler, E.L., Lenstra, J.K., Rinooy Kan, A.H.G., and Shmoys, D.B. (eds.) (1985). *The Travelling Salesman Problem: A Guided Tour of Combinatorial Optimization*, Wiley, Chichester.

Helbig Hansen, K., and Krarup, J. (1974). Improvements of the Held-Karp algorithm for the symmetric travelling salesman problem. *Mathematical Programming 7*, 87-96.

Held, M., and Karp, R.M. (1962). A dynamic programming approach to sequencing problems. *SIAM Journal of Applied Mathematics 10*, 196-210.

Held, M., and Karp, R.M. (1970). The travelling-salesman problem and minimum spanning trees. *Operations Research 18*, 1138-1162.

Held, M., and Karp, R.M. (1971). The travelling-salesman problem and minimum spanning trees: part II. *Mathematical Programming 1*, 6-25.

Held, M., Wolfe, P., and Crowder, H.P. (1974). Validation of subgradient optimization. *Mathematical Programming 6*, 62-88.

Hoffman, A.J. and Wolfe, P. (1985). History. Lawler, E.L., Lenstra, J.K., Rinooy Kan, A.H.G., and Shmoys, D.B. (eds.) (1985). *The Travelling Salesman Problem: A Guided Tour of Combinatorial Optimization*, Wiley, Chichester.

Hong, S. (1977). The asymmetric travelling salesman problem and spanning arborescences of directed graphs. *Operations Research verfahren, 25*, 1920-204.

Houck, D.J., Picard, J.-C., Queyranne, M., and Vemuganti, R.R. (1980). The travelling salesman problem as a constrained shortest path problem: theory and computational experience. *Opsearch 17*, 93-109.

Imai, H. (1986). Worst-case analysis for planar matching and tour heuristics with bucketing techniques and spacefilling curves. *Journal of the Operations Research society of Japan 29*, 43-67.

Iri, M., Murota, K., and Matsui, S. (1983). Heuristics for planar minimum-weight perfect matchings. *Networks 13*, 67-92.

Johnson, D.S., and Papadimitriou, C.H. (1985). Performance guarantees for heuristics. Lawler, E.L., Lenstra, J.K., Rinooy Kan, A.H.G., and Shmoys, D.B. (eds.) (1985). *The Travelling Salesman Problem: A Guided Tour of Combinatorial Optimization*, Wiley, Chichester.

Jonker, R (1986). *Travelling salesman and assignment algorithms: design and implementation.* PhD Thesis, University of Amsterdam.

Jonker, R., De Leve, G., Van der Velde, J and Volgenant, T. (1980). Bounding symmetric travelling salesman problems with an asymmetric assignment problem. Operations Research, 28, 623-627.

Karg, R.L., and Thompson, G.L. (1964). A heuristic approach to solving the travelling salesman problems. *Management Science 10*, 225-248.

Karp, R.M. (1971). A simple derivation of Edmonds' algorithm for optimum branchings. *Networks 1*, 265-272.

Karp, R.M. (1972). Reducibility among combinatorial problems. Miller, R.E., Thatcher, J.W. (eds). *Complexity of Computer Computations*, Plenum Press, New York, 85-103.

Karp, R.M. (1977). Probabilistic analysis of partitioning algorithms for the travelling salesman problem in the plane. *Mathematics of Operations Research 2*, 209-224.

Karp, R.M. (1979). A patching algorithm for the nonsymmetric travelling-salesman problem in the plane. *SIAM J. Comput. 8*, 561-573.

Karp, R.M., and Steele, J.M. (1985). Probabilistic analysis of heuristics. Lawler, E.L., Lenstra, J.K., Rinooy Kan, A.H.G., and Shmoys, D.B. (eds.). *The Travelling Salesman Problem: A Guided Tour of Combinatorial Optimization*, Wiley, Chichester.

Katoh, N., Ibaraki, T. and Mine, H (1981), An algorithm for finding K minimum spanning trees. SIAM Journal on Computing 10, 247-255.

Kirkpatrick, S., Gelatt, C.D., and Vecchi, M.P. (1982). *Optimization by Simulated Annealing*, IBM Computer Science/Engineering report, IBM Thomas J. Watson Research Center, Yorktown Heights, NY.

Kirkpatrick, S., Gelatt, C.D., and Vecchi, M.P. (1983). Optimization by simulated annealing. *Science 220*, 671-680.

Krolak, P.D., Felts, W., and Marble, G. (1971). A man-machine approach toward solving the travelling salesman problem. *Communications ACM 14*, 327-334.

Krolak, P.D., Felts, W., and Nelson, J.H. (1972). A man-machine approach toward solving the generalized truck dispatching problem. *Transportation Science 6*, 149-170.

Kruskal, J.B. (1956). On the shortest spanning subtree of a graph and the travelling salesman problem. *Proceedings of the American Mathematical Society 7*, 48-50.

Kuhn, H.W. (1955). The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly 2*, 83-97.

Land, A.H. (1979). *The solution of 100-city symmetric travelling salesman problems*, Research report, London School of Economics.

Lawler, E.L. (1976). *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart and Winston, New York.

Lawler, E.L., Lenstra, J.K., Rinooy Kan, A.H.G., and Shmoys, D.B. (eds.) (1985). *The Travelling Salesman Problem: A Guided Tour of Combinatorial Optimization*, Wiley, Chichester.

Lenstra, J.K. (1974). Clustering a data array and the travelling salesman problem. *Operations Research 22*, 413-414.

Lenstra, J.K. (1976). *Sequencing by Enumerative Methods*, Mathematisch Centrum, Amsterdam.

Lenstra, J.K., and Rinnooy Kan, A.H.G. (1975). Some simple applications of the travelling salesman problem. *Operations Research Quarterly 26*, 717-733.

Lin, S. (1965). Computer solutions of the travelling salesman problem. *Bell System Technical Journal 44*, 2245-2269.

Lin, S., and Kernighan, B.W. (1973). An effective heuristic algorithm for the travelling salesman problem. *Operations Research 21*, 498-516.

Little, J.D.C., Murty, K.G., Sweeny D.W., and Karel, C. (1963). An algorithm for the travelling salesman problem. *Operations Research 11*, 972-989.

Mandelbrot, B.B. (1983). *The Fractal Geometry of Nature*. W.H.Freeman and Co., San Francisco.

McCallum, C.J. Jr. (1986). Operations research in manufacturing. *AT&T Technical Journal 65*, 4, 4-16.

Miliotis, P. (1976). Integer programming approaches to the travelling salesman problem. *Mathematical Programming 10*, 367-378.

Miliotis, P. (1978). Using cutting planes to solve the symmetric travelling salesman problem. *Mathematical Programming 15*, 177-188.

Mohan, J. (1983). Experience with two parallel programs for solving the travelling salesman problem. *Proceedings of the 1983 International conference on Parallel Processing*, 191-193.

Nemhauser, G.L. and Wolsey, L.A. (1988). *Integer and Combinatorial Optimization*. John Wiley, Chichester.

Norback, J.P., and Love, R.F. (1977). Geometric approaches to solving the travelling salesman problem. *Management Science 23*, 1208-1223.

Norback, J.P., and Love, R.F. (1979). Heuristic for the Hamiltonian path in Euclidean two space. *Journal of the Operational Research Society 30*, 363-368.

Or, I. (1976). *Travelling Salesman-Type combinatorial Problems and their relation to the Logistics of regional Blood Banking*, Ph.D. thesis, Northwestern University, Evanston, IL.

Padberg, M.W., and Hong, S. (1980). On the symmetric travelling salesman problem: a computational study. *Mathematical Programming Studies 12*, 78-107.

Padberg, M.W., and Grötschel, M. (1985). Polyhedral Computations. Lawler, E.L., Lenstra, J.K., Rinooy Kan, A.H.G., and Shmoys, D.B. (eds.) (1985). *The Travelling Salesman Problem: A Guided Tour of Combinatorial Optimization*, Wiley, Chichester.

Padberg, M.W., and Rinaldi, G. (1987). Optimization of a 532-city travelling salesman problem by branch and cut. *Operations Research Letters 6*, 1-8.

Papadimitriou, C.H. (1977). The euclidean travelling salesman problem is NP-Complete. *Theoretical Computer Science 4*, 237-244.

Papadimitriou, C.H., and Steiglitz, K. (1982). *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, NJ.

Pekny, J.F., and Miller, D.L. (1988). *A parallel branch and bound algorithm for solving large asymmetric travelling salesman problems*. Technical Report EDRC 05-27-88, Carnegie-Mellon University, Pittsburgh, PA 15212.

Platzman,L.K., and Bartholdi, J.J. (1989). Spacefilling curves and the planar travelling salesman problem. *JACM 36*, 4, 719-737.

Prim, R.C. (1957). Shortest connection networks and some generalizations. *Bell System Technical Journal 36*, 1389-1401.

Raymond, T.C. (1969). Heuristic algorithms for the travelling salesman problem. *IBM Journal for Research and Development 13*, 400-407.

Rosenkrantz, D.J., Stearns, R.E., and Lewis, P.M. II (1977). An analysis of several heuristics for the Travelling salesman problem. *SIAM Journal of Computing 6*, 563-581.

Sandi, C. (1979). Subgradient optimization. Christofides, N., Mingozzi, A., Toth, P., Sandi, C. (eds.). *Combinatorial Optimization*, Wiley, Chichester, 73-91.

Savelsbergh, T., and Volgenant, A. (1985). Edge exchanges in the degree-constrained minimum spanning tree problem. *Computers and Operations Research 12*, 4, 341-348.

Shapiro, D.M. (1966). *Algorithms for the Solution of the Optimal Cost and Bottleneck Travelling Salesman Problems*, Sc.D. thesis, Washington University, St. Louis, MO.

Shapiro, J.F. (1979). A survey of Lagrangean techniques for discrete optimization. *Annals of Discrete Mathematics 5*, 113-138.

Smith, T.H.C. (1975). *A LIFO Implicit Enumeration Algorithm for the Asymmetric Travelling Salesman Problem Using a One-Arborescence Relaxation*, Chapter of Ph.D. thesis, Carnegie-Mellon University, Pittsburg, PA.

Smith, T.H.C., Srinivasan, V., and Thompson, G.L. (1977). Computational performance of three subtour elimination algorithms for solving asymmetric travelling salesman problems. *Annals of Discrete Mathematics 1*, 495-506.

Smith, T.H.C., and Thompson, G.L. (1977). A LIFO implicit enumeration search algorithm for the symmetric travelling salesman problem using Held & Karp's 1-tree relaxation. *Annals of Discrete Mathematics 1*, 479-493.

Spielberg, K. (1979). Enumerative methods in integer programming. *Annals of Discrete Mathematics 5*, 139-183.

Stewart, W.R. (1977). A computationally efficient heuristic for the travelling salesman problem. *Proceedings of the 13th Annual Meeting of S.E. TIMS*, 75-85.

Tarjan, R.E. (1972). Depth-first search and linear graph algorithms. *SIAM Journal of Computing 1*, 146-160.

Tarjan, R.E. (1977). Finding optimum branchings. *Networks 7*, 25-35.

Van Laarhoven, P.J.M. and Aarts. E.H.L. (1987). *Simulated annealing: Theory and applications*. Kluwer Academic publishers group, Holland.

Volgenant, T., and Jonker, R. (1982). A branch and bound algorithm for the symmetric travelling salesman problem based on the 1-tree relaxation. *European Journal of Operational Research 9*, 83-89.

Volgenant, T., and Jonker, R. (1983). The symmetric travelling salesman problem and edge-exchanges in minimal 1-trees. *European Journal Operational Research 12*, 394-403.

APPENDIX 1.1

This appendix includes dat for the Euclidean problems problems we generated. The 360k, low density, IBM formatted floppy disk that is provided contains the data for the number of nodes, the problem name, the optimal solution value and the *x*-*y* coordinates of for each of the problems. The first record of the file consists of three fields: *n*, Problem Name, Optimal solution value. The rest of the *n* records have two elements, corresponding to the x and y coordinates of the *i*th point.

Twenty new problems are generated:

n	Number of Problems	Problem_name (File_name)
50	5	KC0500
		KC0501
		KC0502
		KC0503
		KC0504
65	5	KC0650
		KC0651
		KC0652
		KC0653
		KC0654
75	5	KC0750
		KC0751
		KC0752
		KC0753
		KC0754
100	5	KC1000
		KC1001
		KC1002
		KC1030
		KC1005

The cost data for each of the problems is calculated using the Euclidean distance metric, rounded up to the nearest integer.