http://researchspace.auckland.ac.nz

ResearchSpace@Auckland

Copyright Statement

The digital copy of this thesis is protected by the Copyright Act 1994 (New Zealand).

This thesis may be consulted by you, provided you comply with the provisions of the Act and the following conditions of use:

- Any use you make of these documents or images must be for research or private study purposes only, and you may not make them available to any other person.
- Authors control the copyright of their thesis. You will recognise the author’s right to be identified as the author of this thesis, and due acknowledgement will be made to the author where appropriate.
- You will obtain the author’s permission before publishing any material from their thesis.

To request permissions please use the Feedback form on our webpage. [http://researchspace.auckland.ac.nz/feedback](http://researchspace.auckland.ac.nz/feedback)

General copyright and disclaimer

In addition to the above conditions, authors give their consent for the digital copy of their work to be used subject to the conditions specified on the [Library Thesis Consent Form](http://researchspace.auckland.ac.nz) and [Deposit Licence](http://researchspace.auckland.ac.nz).

Note: Masters Theses

The digital copy of a masters thesis is as submitted for examination and contains no corrections. The print copy, usually available in the University Library, may contain corrections made by hand, which have been requested by the supervisor.
New Insights on the Multistage Insertion Formulation of the Traveling Salesman Problem
Polytopes, Algorithms, and Experiments

Laleh Haerian Ardekani

A thesis submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy (PhD), in Operations Management,
The University of Auckland,
2011.
Abstract

The multistage insertion formulation (MI) is a compact model for the traveling salesman problem (TSP). The MI formulation has given rise to a combinatorial object called pedigree, and a combinatorial polytope called the pedigree polytope. Previous studies have shown some characteristics of the MI formulation and the pedigree polytope. However, very limited computational experiments have been done on this formulation. In this thesis, we perform some empirical studies on the pedigree polytope and the MI formulations for the symmetric TSP (STSP), and the asymmetric TSP (ATSP).

Given a solution to the LP relaxation of the MI formulation, a necessary condition for membership of the solution in the pedigree polytope can be associated with the maximum multicommodity flow in some layered network being equal to one. Using a numerical example, we illustrate a procedure for checking this necessary condition. We answer the question of the necessary condition being sufficient in the negative by providing a numerical example.

We compare the performance of the LP relaxation of the MI formulation with those of various TSP formulations by solving some STSP and ATSP instances from the TSP Library (TSPLIB). We also solve some problems by Papadimitriou and Steiglitz called diamond instances. The LP relaxation of the MI formulation finds the integer solution to these diamond instances in all the cases. We find the MI formulation to perform better than other formulations in terms of solution time or LP relaxation value for both STSP and ATSP.

We develop some branching rules using the structure of the MI formulation to be used in a branch and bound method, and find that compared to other formulations the MI formulation provides smaller branching trees and requires less solution time. Using the structure of pedigrees, we suggest five heuristics for the STSP. We compare their performance with some other existing TSP heuristics through solving some TSPLIB instances. We find two of these heuristics to perform better than other TSP heuristics.
To my parents, Jamileh and Ali, for their love, support and encouragements.

To my sister Laila, for always being my source of inspiration.

To my dearest Babak, for his love that kept me going.
Acknowledgements

I extend my deepest gratitude to my supervisors AP Tiru Arthanari and AP Matthias Ehrgott, for their valuable support and guidance throughout making of this thesis. I feel privileged to have worked with them.

I would like to thank my colleagues in The Department of ISOM who have helped and supported me.

I would also like to thank my friends in the University of Auckland who accompanied me along this journey: Chris, Lincoln, Hendrik, Parizad, Rebecca, and Ioanna.
# Contents

List of Figures vi

List of Tables ix

1 Introduction 1

1.1 A Brief History of the Traveling Salesman Problem 1
1.2 Combinatorial Optimisation Problems 2
1.3 Complexity of Computational Problems 2
1.4 The Traveling Salesman Problem (TSP) 3
1.5 Heuristics for the TSP 5
1.6 Applications of the TSP 7
1.7 The Layout of This Thesis 9

2 Preliminaries and Notations 11

2.1 Graph Theory Preliminaries 11
2.2 Network Flow Problems 12
   2.2.1 The Ford-Fulkerson Algorithm 14
   2.2.2 The Preflow-Push Algorithm 15
   2.2.3 The Forbidden Arcs Transportation Problem 17
   2.2.4 Finding Arcs with Frozen Flow in a Network 18
   2.2.5 The Multicommodity Flow Problem 18
2.3 Polyhedral Theory and Polyhedral Combinatorics 21

3 Various Formulations for the TSP 23

3.1 Flood 23
3.2 Dantzig, Fulkerson and Johnson 24
3.3 Miller, Tucker and Zemlin 25
3.4 Fox, Gavish and Graves 26
3.5 Wong 27
3.6 Picard and Queyranne 28
3.7 Claus 29
List of Figures

4.1 Graph of Rigidity for the Pedigrees in Example 4.2.6 
5.1 The Network for the FAT_k(\lambda) Problem 
5.2 The N_4 Network 
5.3 (a) The N_7 Network, and (b) the Restricted Network N_6([7 : 5,6], [8 : 4,6]). 
5.4 The Solution to the F_5 Problem 
5.5 Enlarged Network N 
5.6 A Flow Chart for Checking the Necessary Condition for X/k + 1 \in conv(P_{k+1}) 
5.7 The Network for the MI-relaxation Solution in Example 5.3.1 
5.8 The N_4 Network 
5.9 Restricted Networks for ([5 : 2,3],[6 : 2,5]), ([5 : 2,3],[6 : 3,5]), and ([5 : 1,4],[6 : 4,5]) 
5.10 The Optimal Flow in F_5 
5.11 The Multicommodity Network N_5 
5.12 Restricted Networks for Links in N_6 
5.13 The Optimal Flows in F_6 
5.14 The N_6 Multicommodity Flow Network 
5.15 Multicommodity Flows in N_6 
5.16 The Restricted Network 
5.17 The F_8 Problem 
5.18 The Restricted Network with the Dummy Arcs Included 
5.19 The F_8 Problem When the Dummy Arcs Are Not Discarded 
5.20 Multicommodity Network for the N_8 with the Dummy Arcs Included 
5.21 The Multicommodity Network for the N_8 with the Dummy Arcs Included 
5.22 Multicommodity Flow in N_8 
5.23 The F_9 Problem in N_9 
5.24 The N_9 Network 
5.25 Multicommodity Flow in the N_9 Network 
5.26 The N_8 Network
LIST OF FIGURES

6.1 Cplex Iterations for Solving STSP Instances in the TSPLIB by Various Formulations .................................................. 78
6.2 CPU Solution Seconds for Solving STSP Instances in the TSPLIB by Various Formulations ............................................. 82
6.3 Solution Seconds for Solving ATSP Instances in the TSPLIB .................. 84
6.4 Cplex Iterations for ATSP Instances ........................................ 85
6.5 The North-south (Left) and East-west (Right) Cycles ....................... 87
6.6 A 2−diamond Instance .......................................................... 88
6.7 The Only Hamiltonian Cycle of a 2-diamond Instance with Cost 16 .......... 88
6.8 CPU Solution Seconds for Diamond Instances by the MI-relaxation .......... 89
6.9 Cplex Solution Seconds for Solving the Diamond Instances ................ 91
6.10 Number of Cplex Iterations for Solving Diamond Instances ............... 93
6.11 A Directed Diamond .......................................................... 93
6.12 The East-west Circuit Passing Through a Diamond ......................... 94
6.13 Cplex Solution Seconds for Solving ATSP Diamond Instances .......... 95
6.14 Number of Cplex Iterations for Solving ATSP Diamond Instances ........ 95

7.1 Branching Rule for a Problem $m$ .............................................. 102
7.2 A Schematic Illustration of the Three MI Branching Rules: (a) MIR$_1$, (b) MIR$_2$, (c) MIR$_3$ .................................................. 105
7.3 Branch and Bound Solution Seconds for Various MI Branching Rules ...... 107
7.4 Branch and Bound Tree Size for Various MI Branching Rules .............. 107
7.5 Branch and Bound Tree Size for TSP Models ................................ 108
7.6 Branch and Bound Solution Seconds for TSP Models ....................... 109
7.7 Branch and Bound Solution Seconds for MI and DFJ Models ............... 109

8.1 The Pedigree Tree of the Pedigree in Example 8.1.1 .......................... 112
8.2 The Gap with the Optimal Solution for the NN and the NP Heuristics .... 115
8.3 The CPU Seconds for the NN and the NP Heuristics ....................... 116
8.4 The Gap with the Optimal Solution for Methods I and II .................. 119
8.5 CPU Seconds for Methods I and II ........................................... 119
8.6 Some 1-disc Neighbours of the Pedigree in Example 8.4.1 .................. 121
8.7 Some 2-disc Neighbours of the Pedigree in Example 8.5.1 .................. 123
8.8 Some Pedigree Branch Neighbours of the Pedigree in Example 8.6.1 ........ 126
8.9 The Gap with the Optimal Solution for Heuristic Combinations (1) ........ 129
8.10 The Gap with the Optimal Solution for Heuristic Combinations (2) ....... 130
8.11 CPU Seconds for Heuristic Combinations (1) ................................ 131
8.12 CPU Seconds for Heuristic Combinations (2) ................................ 132
8.13 The Number of Iterations for Heuristic Combinations (1) .......................................... 133
8.14 The Number of Iterations for Heuristic Combinations (2) .......................................... 134
List of Tables

1.1 Comparison of Construction Heuristics .......................................... 7
3.1 Number of Constraints and Variables of the TSP formulations ............... 31
3.2 Comparison of the Projected Polytopes with the SEP .......................... 32
4.1 Comparison of the STSP and $\text{conv}(P_n)$ Polytopes ......................... 40
5.1 The $x_{ijk}$ Values for the MI-relaxation Solution in Example 5.1.1 ............ 45
5.2 The Relationship Between $x_{ijk}$ and $u_{ijk}$ Values .............................. 55
5.3 The $P$ Matrix Corresponding to Example 5.3.2 .................................. 58
5.4 The $x_{ijk}$ Values for the MI-relaxation Solution in Example 5.4.1 ............ 59
5.5 The $x_{ijk}$ Values for the MI-relaxation Solution in Example 5.5.1 ............ 67
5.6 The Optimal Solution to the $F_9$ Problem ....................................... 68
5.7 All the Possible Pedigrees for $X/9$ .................................................. 70
6.1 Results for the DFJ Formulation Compared with the MI-relaxation .......... 76
6.2 Results for the LP Relaxation of MI Formulation for Some TSPLIB Instances 77
6.3 LP Relaxation Values of Some STSP Formulations for Some TSPLIB Instances 79
6.4 The Gap for Some STSP Instances in the TSPLIB by Various Formulations 80
6.5 Number of Cplex Iterations for Solving Some STSP Instances in the TSPLIB by Various Formulations .............................................. 81
6.6 The Gap of Solving Some Instances in the TSPLIB by the LP Relaxation of ATSP Formulations .................................................. 83
6.7 Cplex Solution Seconds of Solving Some Instances in the TSPLIB by the LP Relaxation of ATSP Formulations ..................................... 83
6.8 Cplex Iterations of Solving Some Instances in the TSPLIB by the LP Relaxation of ATSP Formulations .............................................. 84
6.9 Comparing the MI and MCF Formulation in Terms of Gaps with the Optimal Solution and CPU Solution Seconds ............................. 85
### LIST OF TABLES

6.10 Results for Solving Diamond Instances with $2 \leq k \leq 14$, by the LP Relaxation of the MI Formulation ........................................ 90
6.11 Solution Values of Some Diamond Instances ........................................ 91
6.12 Cplex Solution Seconds for Solving the Diamond Instances .................. 92
6.13 Number of Cplex Iterations for Solving the Diamond Instances ............... 92
6.14 Solution Seconds for Solving Directed Diamond with ATSP Formulations .... 94
6.15 Cplex Iterations for Solving Directed Diamond with ATSP Formulations .... 95

7.1 Solution to the MI Relaxation of Some 13-city Problem ........................ 103
7.2 CPU Seconds for Branch and Bound Method with MI Branching Rules ....... 106
7.3 Size of the Branch and Bound Tree with MI Branching Rules ................. 107
7.4 Size of the Branch and Bound Tree for Various Formulations ................ 108
7.5 Solution Seconds for Branch and Bound on Different Methods ............... 109

8.1 Computational Comparison of the NN and the NP Heuristics ................... 114
8.2 The MI Relaxation Solution in Example 8.3.1 .................................... 116
8.3 The MI Relaxation Solution in Example 8.3.2 .................................... 117
8.4 Computational Results on Pedigree Extraction Using Methods I and II ...... 118
8.5 Vertex Node Connections for Neighbours of the Pedigree in Example 8.6.2 127
8.6 Computational Results for the 2-opt Heuristic .................................... 129
8.7 Computational Results for the 3-opt Heuristic .................................... 130
8.8 Computational Results for the Pedigree Branch Heuristic ...................... 131
8.9 Computational Results for the 1-Disc Heuristic ................................... 132
8.10 Computational Results for the 2-Disc Heuristic ................................ 133
1

Introduction

1.1 A Brief History of the Traveling Salesman Problem

A traveling manual published in 1832, raised the question of finding the shortest tour, for traveling salesmen who wished to visit 45 cities in Germany (Schrijver 2005). The traveling salesman problem was not known in its current form at that time yet. Schrijver (2005) believes that it was not until the 1930s and 1940s that mathematicians started studying the problem as we know it today. The origin of the term traveling salesman problem (TSP) is traced back to 1949 in a report by Robinson (1949) which was published by the RAND Corporation. The TSP is famous for being a difficult problem despite its innocent appearance (Karp 1986). The problem is very easy to define: given a finite set of cities, what is the shortest tour that visits all these cities exactly once, and returns to the original city. But finding the shortest of these tours amongst the exponentially many possibilities is indeed a difficult task.

In 1954 Dantzig et al. (1954) suggested a solution method for a problem of 49 cities that was a breakthrough in the field of optimisation. Hoffman and Wolfe (1985) refer to this as one of the principal events in the history of combinatorial optimization. By the year 1962 the problem was quite well-known, and in fact a $10,000 prize competition for finding the shortest tour of 33 cities in the US was held by Proctor and Gamble in that year (Applegate et al. 2006).

After the year 1972 when Karp (1972) proved the TSP, along with many other combinatorial optimisation problems to be NP-hard, many solution methods, heuristics, and metaheuristics were designed to tackle these problems (Lawler et al. 1985). The TSP has probably attracted the most attention among all combinatorial optimisation problem in this class. It has served as a very good test bed for validating new algorithms and combinatorial methods (Reinelt 1994). Many algorithmic ideas and solution methods have been first developed for or tested on the TSP and then applied on related problems. Examples of such algorithms or methods are branch and bound, Lagrangian relaxation,
1.2 Combinatorial Optimisation Problems

According to Lawler (2001), *combinatorial optimisation problems* in simple words look for the best combination of discrete objects that are finite in number but usually have a practically infinite number of combination possibilities. Combinatorial optimisation problems seek to optimise an objective function over combinations of several discrete and usually integer variables that belong to a finite set, subject to some constraints. Each solution in a combinatorial optimisation problem is a set of several variables that can be associated with a graph, a permutation, an arrangement, or a sequence of discrete objects (Papadimitriou and Steiglitz, 1982).

Generally a typical combinatorial optimisation problem is based on a ground set \( E \) of finite elements \( e \in E \), a variable \( x_e \), and a cost parameter \( c_e \in \mathbb{R} \) associated with each of these elements. A single occurrence of a combinatorial optimisation problem based on a known input in the form of a set \( F \subseteq E \) is called an instance of the problem, and it is indicated by the vector \( x^F \in \{0, 1\}^{|E|} \). The elements of \( x^F \) vectors are defined as

\[
x^F(e) = \begin{cases} 
1, & \text{if } e \in F, \\
0, & \text{otherwise}.
\end{cases}
\]

Assuming that \( \mathcal{F} \) is the set of all possible subsets \( F \), combinatorial optimisation problems search for an element \( F \in \mathcal{F} \) for which \( c(F) = \sum_{e \in F} c_e x^F_e \) is optimised (Lawler et al., 1985). Some of the typical combinatorial optimisation problems are: the traveling salesman problem, the vehicle routing problem, the graph partitioning problem, the job sequencing problem, the matching problem, and the minimum spanning tree problem. Further studies on combinatorial optimisation problems and their solution algorithms can be found in Papadimitriou and Steiglitz (1982), Cook et al. (1998), Nemhauser and Wolsey (1999), Lawler (2001), and Schrijver (2003).

We discuss the complexity of combinatorial optimisation problems in the next section.

1.3 Complexity of Computational Problems

In 1971 William Cook contributed some seminal concepts in the *theory of computational complexity* (Cook, 1971) which are used for classifying computational problems in terms of
their difficulty. The question of whether there exist a feasible solution to a given problem is referred to as the recognition problem, where the answer can be in the form of yes or no. For instance: Does there exist a tour of length less than or equal to $d$, is a recognition problem. Evaluating the correctness of a solution for which the answer to the recognition problem is yes, is called the evaluation problem. The class $P$ refers to a class of recognition problems that can be solved in polynomial time. This class belongs to a larger class of problems called the nondeterministic polynomial time ($NP$) class. For the problems in the $NP$ class, it is not required that all the instances of the problems be solvable in polynomial time, but it is required that if a solution to the problem is given, its corresponding evaluation problem can be solved in polynomial time (Cook et al., 1998).

Cook (1971) has also defined an important subclass of the $NP$ problems which is now known as the $NP$-complete subclass. A recognition problem in the class $NP$ is known to be $NP$-complete if all other problems in class $NP$ polynomially transform to this problem (Papadimitriou and Steiglitz, 1982). This also implies that if an $NP$-complete problem can be solved in polynomial time, then all other $NP$-complete problems can be solved in polynomial time. In other words this would mean that $P = NP$, a result that has neither been proven nor disproved.

Following the theory of computational complexity, Karp (1972) showed that many combinatorial optimisation problems are $NP$-complete. He has shown how many problems in sequencing, routing, matching, and assignment are equivalent; i.e. either they can all be solved by polynomial time algorithms or none of them can. Some examples of $NP$-complete problems are the knapsack problem, the job sequencing problem, the linear 0-1 integer programming, and the traveling salesman problem. The term $NP$-hard is used to address optimisation problems the recognition problems of which are known to be $NP$-complete, e.g. the TSP (the optimisation problem) is known to be $NP$-hard (Papadimitriou and Steiglitz, 1982).

Next we give an overview of the traveling salesman problem as a typical problem in combinatorial optimisation, and polyhedral combinatorics.

1.4 The Traveling Salesman Problem (TSP)

Let $V$ be a finite set of nodes (cities), where the traveling cost between any two nodes $i$ and $j$ in $V$ is given and is indicated by $c_{ij}$. As mentioned in Section 1.1, the TSP aims to find the shortest tour that visits all the nodes in $V$ exactly once, and returns to the original node. If for a problem instance we have $c_{ij} = c_{ji}$, for all nodes $i$ and $j$ in $V$, then the problem is called a symmetric traveling salesman problem (STSP), and it is called an asymmetric traveling salesman problem (ATSP) otherwise. Different variations of the TSP have also been studied, we give some of these variations next.
1.4 The Traveling Salesman Problem (TSP)

- Given a finite set of nodes, the **bottleneck TSP** seeks to find a tour for which the length of the longest edge is minimised (Edmonds and Fulkerson, 1970). Garfinkel and Gilbert (1978) have suggested a model and a solution method for the bottleneck TSP.

- The **time dependent TSP** is a generalization of the TSP where the distance from node $i$ to node $j$ is expressed by $c_{ijt}$, where $t$ indicates the time window in which node $j$ is visited after node $i$. This problem is used to minimise the setup time on a machine for processing $n$ jobs where the setup cost for each job depends on its preceding job, and also on its position in the sequence that defines the job schedule (Picard and Queyranne, 1978).

- Given a set of $n$ nodes, the **multiple TSP** minimises the total distance traveled by multiple traveling salesmen, on the condition that each city, except the starting city, be visited by exactly one salesman. All the salesmen start and finish in the same city (Gavish and Srikant, 1986).

- Given $n$ clusters of nodes, the **generalized TSP** finds the shortest tour that visits exactly one node from each cluster and returns to the original node (Laporte and Nobert, 1983).

- The **stacker crane problem** is a generalization of the TSP where some edges are required to be included in the tour (Johnson and Papadimitriou, 1985).

The TSP is not only a good representative of many other combinatorial optimisation problems, but it also has become a typical problem for validating algorithmic ideas for other difficult combinatorial optimisation problems (Lawler et al., 1985). Many different models and solution algorithms have been suggested for the TSP so far. Bellmore and Nemhauser (1968) have studied the TSP and classified different solution methods for the TSP. A collection of studies on the TSP is gathered by Lawler et al. (1985). Gutin and Punnen (2002) have also studied the TSP and have provided the state of the art in theory and algorithms for the TSP. In a more recent book Applegate et al. (2006) have studied the computational history of the TSP and reported on the latest advances on solution methods for the TSP.

A library of TSP test problems called the **TSPLIB** is collected by Reinelt (1995). The TSPLIB instances are often used for testing TSP solution algorithms. The state of the art solution method for the TSP is a computer code written by Applegate et al. (2004) called Concorde developed for solving the STSP. Concorde is known to have found the optimal solutions to 107 of the 110 TSPLIB instances. The largest problem instance that was solved by Concorde had 15,112 nodes.
In the next section, we give an introduction to some of the heuristics for the TSP.

1.5 Heuristics for the TSP

The NP-hardness of the TSP has motivated many researchers to suggest heuristic solutions for it. TSP heuristics are usually classified either as construction, improvement, or composite heuristics. Given a set of nodes and a cost matrix construction heuristics generate approximate tours, whereas the improvement heuristics take an initial tour and repeatedly modify it to achieve better (shorter) tours (Johnson and McGeoch, 1997). The composite heuristics usually start the solution process with a tour from a construction heuristic and then apply tour improvement algorithms to the solution to improve the tour (Golden and Stewart, 1985). Some of the heuristics for the TSP are given below.

- **The nearest neighbour heuristic** is a construction heuristic that starts from an arbitrarily selected node and its self-loop, and then selects the free nodes one by one and adds them to the end of the tour until the tour is complete. At each step, a free node that has the shortest distance to the last node of the tour is selected. The running time for the nearest neighbour algorithm is $O(n^2)$ (Johnson and McGeoch, 1997).

- **The greedy heuristic** constructs TSP tours by adding the edges one at a time by selecting the shortest available edge each time. As the construction process proceeds, some of the edges that would make the solution infeasible become unavailable. The Greedy algorithm can run in $O(n^2 \log n)$ time (Johnson and McGeoch, 1997).

- **The nearest insertion** and **the cheapest insertion** are both construction heuristics that start from an initial node and its self-loop. In the nearest insertion method, a free node that has minimum distance to the tour nodes is inserted in the tour. In the cheapest insertion method, the node chosen for insertion is a free node the insertion of which would impose the smallest increase to the length of the tour. The tours obtained by these two methods are guaranteed to be shorter than twice the length of the optimal tour (Rosenkrantz et al., 1977). Some other insertion methods for the TSP such as the farthest insertion and the convex hull insertion are studied by Golden et al. (1980).

- **The minimum spanning tree heuristic** constructs TSP tours using minimum spanning trees. The minimum spanning tree in a given graph can be found using a variety of efficient algorithms (Aho and Hopcroft, 1974). The minimum spanning tree is then traversed and a TSP tour is built based on it. This heuristic runs in $O(n^2)$
time. The length of the tour given by this heuristic is guaranteed to be shorter than or equal to twice the length of the optimal tour (Johnson and Papadimitriou, 1985).

- **The Christofides heuristic** has two phases; first the minimum spanning tree in the graph constructed by the vertices of the problem is found. Then an algorithm called a matching algorithm is applied on the minimum spanning tree to build a TSP tour (Christofides, 1976). The Christofides algorithm uses the minimum spanning tree to obtain an Eulerian graph and an Eulerian tour for the graph, and then converts it to a TSP tour. For more information on Eulerian graphs and Eulerian tours we refer the reader to standard texts such as Nemhauser and Wolsey (1999). The algorithm runs in $O(n^3)$ time, and its tour length is not worse than $3/2$ times the optimal length (Johnson and Papadimitriou, 1985).

- A family of tour improvement heuristics, known as the $k$-opt methods, use edge-exchange procedures to improve a given tour. In general, at each step of a $k$-change method, $k$ edges of a given tour are deleted from the tour and replaced with $k$ new edges to produce a shorter tour. The 2-opt (Croes, 1958) and the 3-opt (Lin, 1965) methods are the most commonly used improvement heuristics for the TSP.

- **The Lin-Kernighan method** suggested by Lin and Kernighan (1973) uses a series of $k$-opt movements. The number of the edges to be exchanged are decided dynamically in each iteration (Golden and Stewart, 1985). The expected running time of this heuristic is $O(n^{2.2})$. However, the worst-case running time of the algorithm is unlikely to be polynomial (Korte and Vygen, 2006). The Lin-Kernighan method has remained a key ingredient in today’s TSP tour finding algorithms (Applegate et al., 2006).

A comparison of the running time and solution qualities of TSP heuristics given in Rosenkrantz et al. (1977) and Johnson and McGeoch (1997) is summarized in Table 1.1. We refer to the ratio of the solution provided by the heuristic to the optimal solution as the Solution Ratio in this table. More detailed studies on the insertion and construction heuristics for the TSP can be found in Rosenkrantz et al. (1977), Golden et al. (1980), Golden and Stewart (1985), Kindervater et al. (1989), Gendreau et al. (1992), and Johnson and McGeoch (2002). General metaheuristics such as the genetic algorithm, the simulated annealing, the ant colony, the neural network algorithms, and the tabu search have also been widely used on the TSP. Johnson and McGeoch (1997) have given an extensive survey of these metaheuristics for the TSP.
1.6 Applications of the TSP

Direct applications of the TSP i.e. finding the shortest tour of some given cities for a traveling salesman, or finding the shortest inspection and delivery routes, were probably the first applications of the problem that were considered (Applegate et al., 2006), but the TSP soon showed to have much wider application across different fields. We give some of these applications below.

- The area of logistics is probably the first area where the TSP was applied (Applegate et al., 2006). Early applications of the TSP include planning the school bus route by Flood (1956) and crop surveys by Morton and Land (1955). Other similar problems have also been solved using the TSP, e.g. crop survey studies by Mahalanobis (1940) and Jessen (1943), and mail delivery problems by Edmonds (1965).

- The TSP can be applied directly to the drilling and printed circuit board problems. These problems usually have a large number of nodes where the drilling machine is guided from one drilling position to the other. The TSP is used to minimise the total travel time for the drill (Applegate et al., 2006). Lin and Kernighan (1973) suggested their well-known TSP heuristic by solving a drilling problem with 318 nodes. Grötschel et al. (1991) solved large scale drilling problems for NC machines using TSP heuristics and reported considerable reductions of production time by applying the solutions. They reported that this application of the TSP in drilling problems at Siemens resulted in saving 17% of the circuit drawing time. The TSP is also adopted for guiding laser beams through three dimensional positions in a piece of crystal (Applegate et al., 2006).

- The sequence of aiming telescopes at celestial objects can be optimised using the TSP. Bailey et al. (2000) have reported on a model for minimising the usage of fuel for a pair of satellites in targeting and imaging maneuvers. In this study the image of

<table>
<thead>
<tr>
<th>Heuristic</th>
<th>Running Time</th>
<th>Solution Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nearest Neighbour</td>
<td>$O(n^2)$</td>
<td>$\leq (\lceil \log_2 n \rceil + 1)/2$</td>
</tr>
<tr>
<td>Greedy</td>
<td>$O(n^2 \log n)$</td>
<td>$\leq (\lceil \log_2 n \rceil + 1)/2$</td>
</tr>
<tr>
<td>Nearest Insertion</td>
<td>$O(n^2)$</td>
<td>$\leq 2$</td>
</tr>
<tr>
<td>Cheapest Insertion</td>
<td>$O(n^2)$</td>
<td>$\leq 2$</td>
</tr>
<tr>
<td>Minimum Spanning Tree</td>
<td>$O(n^2)$</td>
<td>$\leq 2$</td>
</tr>
<tr>
<td>Christofides</td>
<td>$O(n^3)$</td>
<td>$\leq 3/2$</td>
</tr>
</tbody>
</table>
the celestial bodies are the cities and the fuel needed for repositioning the satellites from one image to the next is the cost of travel from one city to the other. Carlson (1997) applied a TSP algorithm for minimising the total time for aiming a telescope at various astronomical spots. The problem of telescope scheduling is similar to the problem of using X-ray in crystallography, where X-ray is aimed at a sample crystal. Bland and Shallcross (1989) applied TSP heuristics for finding the best schedule for doing the crystallography on instances of size 2,762 to 14,464.

- **The machine scheduling problem** is perhaps the most studied application area of the TSP (Punnen, 2002). One simple application is finding the best schedule for a set of $n$ jobs that are to be processed on a machine, with different machine setup configurations. Let $c_{ij}$ be the set-up cost of processing job $j$ immediately after job $i$. It is assumed that when all the jobs are done, the machine is returned to its initial setup. The machine scheduling problem finds a permutation of these $n$ jobs so that the total set up cost is minimised. The machine scheduling problem can be reduced to the clustered TSP where similar jobs are clustered into groups. Ozgur and Brown (1995) have developed a heuristic for such a problem.

- In a **cellular manufacturing** system, some parts or products that require similar processing are grouped together in a cell, and a robot is used for material handling in each cell. Aneja and Kamoun (1999) have modeled the problem of sequencing different parts to be produced and sequencing different robot activities in a two machine cell with a single robot using the TSP.

- **Data clustering** classifies given pieces of data with similar properties into the same groups. Lenstra (1974) showed how some data clustering problem can be expressed as the TSP.

- **The frequency assignment problem** assigns frequencies to different transmitters from a set of given available frequencies subject to some interference constraints (Punnen, 2002). Smith and Hurley (1997) have shown that the TSP can be used to calculate a lower bound for this problem.

- In the field of **bio-informatics**, the TSP is used in areas such as genome sequencing, DNA sequencing or multiple sequence alignment. Agarwala et al. (2000) used the Concorde software for genome sequencing. Avner et al. (2001) used Concorde for developing a map of the mouse genome. Korostensky and Gonnet (1999) used the TSP for calculating a near optimal multiple sequence alignment bound. Korostensky and Gonnet (2000) showed how to reduce the problem of constructing an evolutionary tree of optimal score to a TSP.
1.7 The Layout of This Thesis

• **Other applications of the TSP:** Garfinkel (1977) showed how the problem of minimising wall paper waste can be equivalent to the TSP. Raffliff and Rosenthal (1983) have applied the TSP in an order-picking problem in a rectangular warehouse with the objective function of minimising the picking time. Plante (1988) applied the TSP in a problem for overhauling gas turbine engines. Hubert and Baker (1978) showed an application of the TSP and related problems in data analysis models in psychology. Marchand et al. (2000) have shown some potential applications of the TSP in routing in telecommunication networks. Santos et al. (2008) have used TSP constraints in the mobile oil recovery problem with the objective of maximising oil extraction and minimising the travel costs of an oil recovery unit. Lenstra and Kan (1975) showed how the problem of computer wiring with the objective of minimising the length of the wire used can be solved using the TSP.

For further studies on the applications of the TSP, we refer the reader to the works of Lenstra and Kan (1975), Garfinkel (1985), Punnen (2002), Reinelt (1994), and Applegate et al. (2006).

### 1.7 The Layout of This Thesis

In this thesis we concentrate on a compact integer programming model for the TSP called the *multistage insertion (MI) formulation*, and a polytope defined based on this formulation called the *pedigree polytope*. We develop algorithms based on the MI formulation and perform various empirical studies on the MI formulation and some problems. We also study some problems on the pedigree polytope. We used MATLAB (versions 7.5 to 7.9) for coding and running all the algorithms and experiments, and used Cplex9.1 for solving various linear formulations throughout this work. The remainder of this thesis is formatted as follows.

In Chapter 2 we give an introduction to graph theory, network programming, and polyhedral theory that will be referred to throughout the thesis. We give some of the well-known TSP formulations and compare them in terms of compactness in Chapter 3. In Chapter 4 we present the MI formulation for the STSP and the ATSP and give some characteristics of the MI formulation and the pedigree polytope.

In Chapter 5 we study some necessary conditions and some sufficient conditions for the membership of the MI relaxation solutions in the pedigree polytope. We illustrate an algorithm that is used for this purpose and also answer the question of the necessary condition being sufficient. We compare the performance of the LP relaxation of the MI formulation for STSP and ATSP with various other TSP models in terms of LP relaxation value, solution time, and number of Cplex iterations in Chapter 6. We use some of the
problem instances in the TSPLIB and also some problem instances by [Papadimitriou and Steiglitz (1978)] for this purpose.

In Chapter 7, we apply the branch and bound method on the LP relaxation of the MI formulation and develop some branching rules using the special structure of the formulation. We present some computational results on comparing the performance of branch and bound method on the MI formulation with other TSP formulations. In Chapter 8, we present some STSP heuristics based on the structure of the MI formulation and compare them with other STSP heuristics in terms of LP relaxation value and time. And finally, we give the conclusions and suggestions for possible future studies in Chapter 9.
Preliminaries and Notations

In this chapter some graph theory terminologies (Section 2.1), some network flow problems (Section 2.2), and some definitions in polyhedral theory (Section 2.3) are given. The definitions, terminologies, lemmas and algorithms for graph theory and network programming are taken from [Bazaraa et al. (1990), Murty (1992), Graham et al. (1995), and Nemhauser and Wolsey (1999)]. The definitions in polyhedral theory and polyhedral combinatorics are taken from [Papadimitriou and Steiglitz (1982), Grötschel and Padberg (1983), and Schrijver (1995)].

2.1 Graph Theory Preliminaries

Let \( V \) be a finite nonempty set of elements called nodes or vertices. Let \( E \) be a subset of \( V \times V \) with its elements called edges, denoted as \( e = (v, u) \in E \), where \( u \) and \( v \) are called the endpoints of \( e \). An edge \( e \in E \) is called incident to some node \( v \in V \), if node \( v \) is an endpoint of \( e \). If for an edge in \( E \), the edge is directed from one endpoint to the other, it is called a directed edge or an arc, otherwise it is called an undirected edge. Throughout this study we denote both directed and undirected edges as \((v, u)\) and specify whether this representation indicates a directed edge or not. The set of arcs is usually denoted as \( A \).

A graph (directed graph) is defined using a set of nodes and a set of edges (arcs), and it is denoted by \( G = (V, E)(G = (V, A)) \). A graph is called an undirected (directed) graph if all of its edges are undirected (directed). A directed graph is also known as digraph. A graph with both directed and undirected edges is called a mixed graph. If a graph contains all possible edges between all the nodes, it is called a complete graph. A node in a digraph with no entering arcs (exiting arcs) is called a source node (sink node).

Given a graph \( G = (V, E) \), the set of edges that are incident to a node \( v \in V \) are denoted by \( \delta(v) \). If \( G \) is a digraph then \( \delta^+(v) \) is the set of edges incident to \( v \) that are directed out of \( v \), and \( \delta^-(v) \) is the set of edges incident to \( v \) that are directed towards \( v \). The degree of a node is the number of edges incident to that node. Two nodes in a
2.2 Network Flow Problems

A graph are called adjacent to each other if there exists an edge with its endpoints being these two nodes. A node sequence \((v_0, ..., v_k)\) in \(G\) is called a path if no repetitions in the sequence exists, and if \((v_{i-1}, v_i) \in E\), for all \(i = 1, ..., k\). A node sequence \((v_0, ..., v_k)\), for some \(3 \leq k \leq |V|\), is called a cycle if \(v_0 = v_k\), and if the sequence \((v_0, ..., v_k-1)\) is a path. A cycle that contains all the node in \(V\) is called a Hamiltonian cycle.

Let \(E(U) = \{(i, j) \in E|i, j \in U\}\), for some \(U \subseteq V\) in graph \(G\). A graph \(G' = (V', E(V'))\) is called a subgraph of \(G\) if \(V' \subseteq V\). A subgraph \(G' = (V', E(V'))\) of \(G\) is called a component of \(G\), if and only if there is a path between any two nodes in \(V'\) but not between any of the nodes from \(V'\) and \(V \setminus V'\). If a graph has only one component, it is called connected. An edge of a connected graph that does not lie in any cycle of the graph is called a bridge. A connected graph with no cycles is called a tree. Given a digraph, a strongly connected component of the graph is a subset of \(V\) such that for any given set of vertices \(u\) and \(v\) in the component, there is a path from \(u\) to \(v\).

**Definition 2.1.1.** A graph \(G = (V, E)\) is called a bipartite graph if \(V\) can be partitioned into two nonempty subsets \(V_1\) and \(V_2\) such that every edge in \(E\) has an endpoint in \(V_1\) and another endpoint in \(V_2\). A bipartite graph is generally denoted as \(G = (V_1, V_2; E)\).

**Definition 2.1.2.** Given a digraph \(G = (V, A)\), let some function \(u: A(G) \rightarrow R^+\), called the capacities of the edges in \(G\), be defined. A network is denoted by \(N = (G, u, s, t)\), where \(s\) and \(t\) in \(V\) are the sink and the source nodes respectively.

In the next section some network flow problems and some well known algorithms for these problems are given.

### 2.2 Network Flow Problems

Generally, network flow problems can be defined given a digraph \(G\), a function \(c: A \rightarrow R^+\) where \(c_{ij}\) is the unit cost of flow in some arc \((i, j) \in A\) with flow capacity \(u_{ij}\), and given a function \(b: V \rightarrow R\). A node \(i\) in the network is regarded as a supply, demand or transit node regarding the corresponding \(b_i\) value being negative, positive or zero, respectively. A feasible flow in the network is defined as some \(x: A \rightarrow R^+\) that satisfies:

\[
\sum_{j \in \delta^+(i)} x_{ij} - \sum_{j \in \delta^-(i)} x_{ji} = b_i, \quad \forall i \in V, \tag{2.1}
\]
\[
x_{ij} \leq u_{ij}, \quad \forall (i, j) \in A, \tag{2.2}
\]
\[
x_{ij} \geq 0, \quad \forall (i, j) \in A. \tag{2.3}
\]

Condition (2.1), also known as the flow conservation constraint, guarantees that for any node in the network, the total flow out of the node minus the total flow into the node...
2.2 Network Flow Problems

is equal to $b$ value for the node. Condition (2.2) makes sure that the flows in the arcs do not exceed the arc capacities. The general *minimum cost network flow problem* is to find a feasible flow that minimises

$$
\sum_{(i,j) \in A} c_{ij} x_{ij}, \quad (2.4)
$$

Given a graph $G = (V, E)$, the *shortest $s-t$ path problem* finds a path from the source node $s$ to node $t$ such that the total cost of the edges in the path is minimised. The shortest path problem can be formulated as follows (Bazaraa et al., 1990).

$$
\min \sum_{(i,j) \in E} c_{ij} x_{ij}
$$

subject to

$$
\sum_{(i,j) \in E} x_{ij} - \sum_{(j,i) \in E} x_{ji} = \begin{cases} 
1 & \text{if } i = s, \\
-1 & \text{if } i = t, \\
0 & \text{otherwise,}
\end{cases} 
(2.5)
$$

$$
x_{ij} \in \{0, 1\}, \quad \forall (i,j) \in E. \quad (2.6)
$$

Dijkstra (1959) suggested an algorithm for finding the shortest path between the source node and some given node in a graph. The complexity of this algorithm is $O(n^2)$.

The general *maximum flow problem* is to maximise the flow from the source node to the sink node, subject to flow conservation constraints for all the nodes, and capacity constraints of the arcs. The maximal flow problem can be formulated as follows (Ahuja et al., 1993).

$$
\max v
$$

subject to

$$
\sum_{j \in \delta^+ (i)} x_{ij} - \sum_{j \in \delta^- (i)} x_{ji} = \begin{cases} 
v & \text{for } i = s, \\
0 & \forall i \in V \setminus \{s,t\},
\end{cases} 
(2.7)
$$

$$
x_{ij} \leq u_{ij}, \quad \forall (i,j) \in A, \quad (2.8)
$$

$$
x_{ij} \geq 0, \quad \forall (i,j) \in A. \quad (2.9)
$$

Several algorithms have been suggested for finding the maximum flow in a network. An algorithm by Ford and Fulkerson (1956a) and an algorithm by Goldberg and Tarjan (1988) are given in the following sections.
Definition 2.2.1. Given some flow \( x \) in a network, the residual capacity of some arc \((i, j)\) is defined as \( r_{ij} = u_{ij} - x_{ij} \).

Definition 2.2.2. The residual network \( G(x) \) corresponding to a feasible flow \( x \) is defined as follows. Each arc \((i, j)\) in the original network is replaced by two arcs \((i, j)\) and \((j, i)\). Arc \((i, j)\) has the unit flow cost value equal to \( c_{ij} \) and the residual capacity equal to \( r_{ij} = u_{ij} - x_{ij} \). The unit flow cost of arc \((j, i)\) is \(-c_{ij}\) and its residual capacity is equal to \( x_{ij} \).

Definition 2.2.3. Given a residual network \( G(x) \), a function \( d : N \mapsto \mathbb{Z}^+ \cup \{0\} \) is called a distance function, if \( d(t) = 0 \) and \( d(i) \leq d(j) + 1 \) for all \((i, j)\) in \( G(x) \). Given some node \( i \), \( d(i) \) is also referred to as the distance label of node \( i \).

Given a network with sink node \( t \), if for each node \( i \), \( d(i) \) is equal to the length of the shortest path from \( i \) to \( t \) in the residual network, the distance labels are called exact distance labels.

2.2.1 The Ford-Fulkerson Algorithm

Ford and Fulkerson (1956a) have suggested in their seminal work an algorithm to find the maximum flow in a network (the Ford-Fulkerson algorithm). It starts with any feasible flow \( x \), e.g. \( x = 0 \). Given the residual network, the algorithm finds a path from the source node to the sink node. Such a path is called an augmenting path. The algorithm then sends some flow along the augmenting paths and updates flow capacities along the arcs of the residual network (Algorithm 1), and it terminates when no more augmenting path can be found. According to Theorem 4.4 in Nemhauser and Wolsey (1999), a feasible flow \( f \) is maximum if and only if no augmenting path with respect to flow \( f \) exists. An algorithm for finding the augmenting paths called the labeling algorithm is shown as Algorithm 2.

Algorithm 1 The Augmenting Algorithm

| track back the labels from \( t \) to \( s \) to get augmenting path \( p \) |
| \( \delta \leftarrow \min \{ r_{ij} : (i, j) \in p \} \) |
| augment \( \delta \) units of flow along path \( p \) and update residual capacities |

Under the assumption that the arc capacities are rational numbers, the algorithms runs in polynomial time. With \( A \) being the set of arcs of the network, each flow augmentation is performed \( O(|A|) \) time. Therefore the complexity of the whole algorithm is \( O(|A|) \) times the number of augmentations, which is bounded by the maximal flow \( f^* \), that is found by the algorithm. So the complexity of the whole algorithm is \( O(|A|f^*) \). One important drawback of the labeling algorithm is that if the capacities have irrational values, the algorithm might not terminate and converge to a flow that is less than maximum (Ahuja et al. 1993) and (Cormen et al. 1990).
2.2 Network Flow Problems

Algorithm 2 The Labeling Algorithm

unlabel all the nodes
list ← \{s\}
while list \(\neq \emptyset\) do
    select some node \(i \in\) list
    for each edge \((i, j)\) in the residual network do
        if \(j\) is not labeled then
            set label of \(j\) equal to \(i\)
            add \(j\) to list
        end if
    end for
    remove \(i\) from list
    if \(t\) is labeled then
        return labels of the nodes
    end if
end while
if list = \(\emptyset\) then
    no augmenting path is available
end if

2.2.2 The Preflow-Push Algorithm

An algorithm for finding the maximum flow in a single commodity network, known as the preflow-push algorithm is suggested by Goldberg and Tarjan (Goldberg and Tarjan, 1988). According to Ahuja et al. (1993), the best preflow-push algorithms can outperform the best augmenting path algorithms. Let preflow be a function \(f : A \mapsto R\) that satisfies condition (2.8) and also satisfies

\[
\sum_{j \in \delta^+(i)} f_{ij} - \sum_{j \in \delta^-(i)} f_{ji} \geq 0, \quad \forall i \in V \setminus \{s, t\}. \tag{2.10}
\]

For a given preflow \(f\), let \(e(i) = \sum_{j \in \delta^+(i)} f_{ij} - \sum_{j \in \delta^-(i)} f_{ji}\) be called the excess flow of node \(i\). In a preflow, \(e(i) \geq 0\) for any \(i \in V \setminus \{s, t\}\). A node \(i\) is called active if \(e(i)\) is strictly positive. An arc \((i, j)\) is called admissible if \(i\) is an active node and \(d(j) = d(i) - 1\).

In the preflow-push algorithm, the existence of an active node indicates the infeasibility of the solution. Therefore in the algorithm, the active nodes are selected and their excess flow is pushed to their neighbours along admissible arcs. The neighbouring nodes that are sent the flow are chosen from the nodes closer to the sink so that eventually all the flow is pushed towards the sink. The algorithm terminates when no active node in the network remains. The distance of the nodes to the sink are expressed in terms of their exact distance. A generic version of the preflow-push algorithm is given as Algorithm 3.
Algorithm 3 A Generic preflow-push Algorithm

for all \((i, j) \in (V \setminus \{s\}) \times (V \setminus \{s\})\) do
\[
x_{ij} \leftarrow 0, \quad x_{ji} \leftarrow 0
\]
end for

for all \(j \in V\) do
\[
\text{if } j \in \delta^+(s) \text{ then }
\]
\[
x_{sj} \leftarrow u_{sj}
\]
\[
\text{else }
\]
\[
x_{sj} \leftarrow 0
\]
end if
end for

compute the exact distance labels \(d(i)\)
\[
d(s) \leftarrow n
\]

while active nodes exist do

select an active node \(i\)

if the network contains an admissible arc \((i, j)\) then

\{The push subroutine\}

push \(\delta = \min\{e(i), r_{ij}\}\) units of flow from \(i\) to \(j\)

else

\{The relabel subroutine\}

replace \(d(i)\) by \(\min\{d(j) + 1 : (i, j) \in A(i), \text{ and } r_{ij} > 0\}\)
end if
end while

\(f\) is the maximum flow

return \(x\)
2.2 Network Flow Problems

In the push subroutine of Algorithm 3, a flow of $\delta$ is pushed through the arcs. If $\delta = r_{ij}$, the push is called saturating and otherwise it is called non-saturating. In the relabel subroutine of the algorithm, the values of $d(i)$ is set equal to the largest possible value allowed by labeling constraints. This algorithm can be computed in at most $O(|V|^2|A|)$ steps. Some other variations to the algorithm are given in Ahuja et al. (1993) and Murt (1992). Cerullia et al. (2008) have studied and compared augmenting path and preflow-push algorithms and have suggested an improved approach by combining the two methods.

2.2.3 The Forbidden Arcs Transportation Problem

Consider a network $N$ corresponding to some bipartite graph $G = (O, D; A)$, where $O = \{O_\alpha : \alpha = 1, ..., n_1\}$ and $D = \{O_\beta : \beta = 1, ..., n_2\}$ are the sets of origin and destination nodes, respectively. Let $s_i \geq 0$ for all $i = 1, ..., n_1$, be the supply of the origin nodes and $d_j \geq 0$ for all $j = 1, ..., n_2$, be the demand of the destination nodes. Given a subset of the arcs in $A$, called the set of forbidden arcs, we define $A' = \{(i, j) \in A | (i, j) \text{ is not a forbidden arc}\}$. Given the arc capacities $u_{ij}$ for all the arcs $(i, j) \in A'$, the forbidden arcs transportation (FAT) problem in $N$ is defined as follows (Murty, 1992).

$$\text{max} \sum_{(i,j) \in A'} f_{ij}$$

subject to

$$\sum_{(i,j) \in A'} f_{ij} = s_i, \quad \forall i \in O,$$  \hspace{1cm} (2.11)

$$\sum_{(i,j) \in A'} f_{ij} = d_j, \quad \forall j \in D,$$  \hspace{1cm} (2.12)

$$f_{ij} \leq u_{ij}, \quad \forall (i, j) \in A',$$  \hspace{1cm} (2.13)

$$f_{ij} \geq 0, \quad \forall (i, j) \in A'.$$  \hspace{1cm} (2.14)

The FAT problem can be solved using variations of maximum flow problem algorithms. Different variations of the preflow-push algorithm can also be used for the FAT problem, which can run in $O(|O|^2|D|)$ time in the worst case. Ahuja et al. (1994) have compared maximal flow algorithms in bipartite networks. Gusfield et al. (1987) have also studied and compared different algorithms for bipartite maximal flow problem.

**Definition 2.2.4.** (Arthanari 2008) Given a FAT problem with a feasible solution $f$, an arc in $(i,j) \in A'$ is called a rigid arc if value of $f_{ij}$ is the same in all feasible solutions to the problem. Rigid arcs are said to have frozen flow.

**Definition 2.2.5.** (Arthanari 2008) Rigid arcs with zero frozen flow are called dummy arcs.
2.2 Network Flow Problems

2.2.4 Finding Arcs with Frozen Flow in a Network

To find arcs with frozen flow in a network, some cycles in the network called the traversable cycles need to be used. These cycles can be defined using the following definitions.

**Definition 2.2.6.** (Arthanari, 2008) Given $f$, a feasible solution to a FAT problem, a mixed graph $G_f = (V, A \cup E)$ is defined by its sets of directed and undirected edges of $G_f$ as follows (Arthanari, 2008).

$$A = \{(O_\alpha, D_\beta) | f_{\alpha\beta} = 0, (\alpha, \beta) \in A \} \cup \{(D_\beta, O_\alpha) | f_{\alpha\beta} = c_{\alpha\beta}, (\alpha, \beta) \in A \},$$

$$E = \{(O_\alpha, D_\beta) | 0 < f_{\alpha\beta} < c_{\alpha\beta}, (\alpha, \beta) \in A \}.$$

**Definition 2.2.7.** (Gusfield, 1988) Given a mixed graph, a cycle in the graph is called a traversable cycle if it is possible to walk through it without violating the direction of the directed edges of the cycle. The undirected edges can be walked through the cycle on both directions in different traversable cycles.

Given a feasible solution $f$ to a FAT problem, the arcs with frozen flow can be found using the traversable cycles in the corresponding $G_f$ graph. The arcs that are not included in any of the traversable cycles of $G_f$ are rigid arcs with frozen flow. Gusfield (1988) has used this concept in data security and has suggested an efficient algorithm for finding the arcs. We give Gusfield's algorithm as Algorithm 4. Given a mixed graph, Algorithm 4 returns the set of edges that are not included in any traversable cycles of the graph.

**Algorithm 4** Gusfield’s Algorithm (Gusfield, 1988)

1. find strongly connected components of $G_f$
2. let $A$ be the set of edges running between these components
3. $B \leftarrow \emptyset$
4. for all the strongly connected components do
   - consider the strongly connected component as one separate undirected graph (ignore the directions on any directed edge)
   - $B \leftarrow$ all bridges (if any) in the component
5. end for
6. return $A \cup B$

Strongly connected components of a graph can be found using an algorithm by Tarjan (1972). This algorithm run in time linear in the number of edges of the graph.

2.2.5 The Multicommodity Flow Problem

The multicommodity flow problem finds the cost minimal flow of different commodities that share common arc capacities in a network. In a multicommodity flow problem, the total
flow of all the commodities in any arc \((i, j)\) is restricted to the capacity of the arc which is denoted as \(u_{ij}\). Let \(k = 1, \ldots, K\) indicate the individual commodities in the network. Each commodity \(k\) has an individual source node \(s^k\) and a sink node \(t^k\). Let \(x^k_{ij}\) indicate the flow of commodity \(k\) in some arc \((i, j)\) and let \(c^k_{ij}\) be the cost of unit flow of commodity \(k\) in arc \((i, j)\). Let \(b^k_i\) indicate the supply of some node \(i\) regarding commodity \(k\). The multicommodity flow problem can be formulated as follows (Ahuja et al., 1993).

\[
\begin{align*}
\min & \sum_{(i,j)\in A} \sum_{1 \leq k \leq K} c^k_{ij}x^k_{ij} \\
\text{subject to} & \\
\sum_{1 \leq k \leq K} x^k_{ij} & \leq u_{ij}, \quad \forall (i, j) \in A, \quad (2.15) \\
\sum_{j \in \delta^+(i)} x^k_{ij} - \sum_{j \in \delta^-(i)} x^k_{ji} & = b^k_i, \quad \forall i \in V, k = 1, \ldots, K, \quad (2.16) \\
x^k_{ij} & \leq u^k_{ij}, \quad \forall (i, j) \in A, \forall k = 1, \ldots, K, \quad (2.17) \\
x^k_{ij} & \geq 0, \quad \forall (i, j) \in A, \forall k = 1, \ldots, K. \quad (2.18)
\end{align*}
\]

Constraint (2.15), also known as the bundle constraint, guarantees that the total flow of all the commodities in each arc does not exceed the capacity of the arc. Constraints (2.16) are flow conservation constraints regarding each individual commodity. There is a flow bound of \(u^k_{ij}\) on the flow of any commodity \(k\) in some edge \((i, j)\) as well as the bundle capacity. The LP has \(k|A|\) variables and \(|A| + k(|V| - 1)\) constraints. What makes the multicommodity flow problem difficult is the bundle constraint. Without this constraint, the problem can be separated into several single commodity flow problems that can be efficiently solved using the algorithms mentioned in the previous sections.

The multicommodity maximum flow problem can be defined as maximising the following objective function subject to constraints (2.15)-(2.17).

\[
\sum_{(i,j)\in A} \sum_{1 \leq k \leq K} x^k_{ij} \quad (2.19)
\]

Ford and Fulkerson (1958) have suggested an algorithm called the arc-chain algorithm for solving this problem. Dantzig and Wolfe (1961) used this algorithm for the decomposition principal in general large scale LP problems. The arc chain formulation leads to an LP with \(|A| + k\) constraints and a large number of variables that exponentially grows with \(|V|\). The advantage of the arc-chain method is that there is no need to store the data regarding all the variables but only that of \(|A| + k\) variables at any time. At each step of the algorithm the data corresponding to one additional variable is generated, which is
2.2 Network Flow Problems

why it is called the column generation method. More analysis and other algorithms for the multicommodity flow problem can be found in (Ahuja et al. 1993). Garg and K"onemann (1998) have suggested an algorithm based on the algorithm by Young (1995) and have shown that their algorithm runs faster than previous algorithms. Korte and Vygen (2006) describe the multicommodity flow problem, its analysis and some algorithms for solving the problem. We give the algorithm by Garg and K"onemann (1998) as Algorithm 5.

Let \((G,H)\) be a pair of digraphs on the same vertices. Let \(u : E(G) \to \mathbb{R}_+ \setminus \{0\}\), and let \(b : E(H) \to \mathbb{R}_+\). For each \(s-t\)-path in \(G\), we consider an edge \((t,s)\) in \(E(H)\). Let \(\mathcal{P}\) be the family of \(s-t\)-paths in \(G\) for all \((t,s) \in E(H)\). Let \(\epsilon\) be a positive number where \(\epsilon \leq \frac{1}{2}\). Algorithm 5 finds values \(y : \mathcal{P} \to \mathbb{R}_+\), with \(\sum_{P \in \mathcal{P}, e \in E(P)} y(P) \leq u(e)\), for all \(e \in E(G)\).

Algorithm 5 Multicommodity Flow Approximation Scheme

\[
\text{for all } P \in \mathcal{P} \text{ do} \\
y(P) \leftarrow 0 \\
\text{end for} \\
\delta \leftarrow (u(1+\epsilon))^{-\lceil \frac{1}{2\epsilon} \rceil}(1+\epsilon) \\
\text{for all } e \in E(G) \text{ do} \\
z(e) \leftarrow \delta \\
\text{end for} \\
\text{let } P \in \mathcal{P} \text{ such that } z(E(P)) \text{ is minimum} \\
\text{while } z(E(P)) < 1 \text{ do} \\
\gamma \leftarrow \min_{e \in E(P)} u(e) \\
y(P) \leftarrow y(P) + \gamma \\
\text{for all } e \in E(P) \text{ do} \\
z(e) \leftarrow z(e)(1 + \frac{\epsilon \gamma}{u(e)}) \\
\text{end for} \\
\text{let } P \in \mathcal{P} \text{ such that } z(E(P)) \text{ is minimum} \\
\text{end while} \\
\text{let } \xi \leftarrow \max_{e \in E(G)} \frac{1}{u(e)} \sum_{P \in \mathcal{P}, e \in E(P)} y(P) \\
\text{for all } P \in \mathcal{P} \text{ do} \\
y(P) \leftarrow \frac{y(P)}{\xi} \\
\text{end for} \\
\text{return } y
\]

Garg and K"onemann (1998) have shown that Algorithm 5 produces a feasible solution that its flow value is not worse than \(\frac{1}{1+\epsilon}OPT(G, H, u)\), and its running time is \(O\left(\frac{1}{\epsilon^2} km(m+n \log n) \log n\right)\), where \(k = |E(H)|\), \(n = |V(G)|\), and \(m = |E(G)|\).

Next we give an introduction to polyhedral theory and polyhedral combinatorics.

20
2.3 Polyhedral Theory and Polyhedral Combinatorics

First we give some basic definitions in polyhedral theory and then give a brief introduction to the field of polyhedral combinatorics.

An affine subspace $S$ of $\mathbb{R}^n$ can be defined as $S = \{x \in \mathbb{R}^n | Ax = b\}$, where $A$ is an $m \times n$ matrix and $b$ is a vector in $\mathbb{R}^m$. Every subspace $S$ has a dimension equal to the maximum number of linearly independent vectors in it, i.e. $n – \text{rank}(A)$. An affine subspace of $\mathbb{R}^n$ of dimension $(n-1)$ is called a hyperplane that can be represented as the set of vectors $x$ such that $a_1x_1 + a_2x_2 + \ldots + a_nx_n = b$, where at least one $a_i$ parameter, for $i = 1, \ldots, n$, is not equal to zero. Each hyperplane defines two halfspaces that can be represented by $a_1x_1 + a_2x_2 + \ldots + a_nx_n \leq b$, and $a_1x_1 + a_2x_2 + \ldots + a_nx_n \geq b$. The convex hull of a finite set $S$ is a set of all the vectors that can be expressed as a convex combination of the elements of $S$, and it is denoted as $\text{conv}(S)$. A polyhedron is the intersection of finitely many halfspaces. A polyhedron $P$ can be represented as $P = \{x \in \mathbb{R}^n | Ax \leq b\}$, where $A$ is an $m \times n$ matrix and $b$ is a vector in $\mathbb{R}^m$. A bounded polyhedron is called a polytope. Equivalently a polytope can be defined as the convex hull of finitely many points in $\mathbb{R}^n$, i.e. $P = \text{conv}(V)$, where $V \subseteq \mathbb{R}^n$, and $|V|$ is finite. Given a polytope $P$ and a halfspace $HS$ defined by some tangent hyperplane $H$, if the intersection $f = P \cap HS$ is a subset of $H$, then $f$ is called a face of $P$. Given a polytope of dimension $d$, a face of dimension $(d-1)$ is called a facet, a face of dimension zero is called a vertex, and a face of dimension one is called an edge of the polytope. Two vertices $x$ and $y$ of a polytope $P$ are called adjacent vertices if the line segment $(x, y)$ is an edge of the polytope.

Definition 2.3.1. (Naddef [1989]) The convex hull of a set $X$ of 0-1 vectors in $\mathbb{R}^n$ is called a 0-1 polytope.

Definition 2.3.2. (Naddef and Pulleyblank [1981]) A 0-1 polytope $P$ is called a combinatorial polytope if in case any two vertices $x$ and $y$ in $P$ are not adjacent, there exist two other vertices $u$ and $v$ in $P$ such that $x + y = v + u$.

Finding adjacent vertices of a given vertex of a polytope is an interesting problem. Matsui and Tamura [1995] believe that the success of the simplex algorithm justifies the existence of some efficient edge-following algorithms for some combinatorial optimisation problems. However, testing non-adjacency for some classes of combinatorial optimisation polyhedra might be a difficult problem. For example, Papadimitriou [1978] has shown that testing adjacency of two given vertices in the STSP polytope is NP-complete.

Some other important problems in polyhedral theory are the membership and the separation problems. Given a vector $v \in \mathbb{R}^n$ and a polytope $P \in \mathbb{R}^n$, the membership problem checks whether $v$ belongs to the polytope $P$ or not. The separation problem either concludes that $v \notin P$, or finds a vector $w \in \mathbb{R}^n$ such that $w^T x < w^T v$, for all $x \in P$. The
2.3 Polyhedral Theory and Polyhedral Combinatorics

complexity of the membership and separation problems in a polytope are closely related to the complexity of the optimisation problem on the polytope. Given a rational objective vector \( c \in \mathbb{R}^n \), the optimisation problem either finds some \( x^* \in P \) that maximises (minimises) \( c^T x \) over all \( x \in P \), or concludes that \( P \) is empty (Cook et al., 1998). Grötschel et al. (1988) have shown that the optimisation problem on a polytope is solvable in polynomial time if the separation (membership) problem is solvable in polynomial time.

**Definition 2.3.3.** (Cook et al., 1998) Let \( P = \{ P_t | t \in O \} \), where \( O \) is some collection of objects and for each \( t \in O \), \( P_t \) is a polyhedron. We call the class \( P \) proper if for each object \( t \in O \) we can compute in polynomial time positive integers \( n_t \) and \( s_t \) such that \( P_t \subseteq \mathbb{R}^{n_t} \) and such that \( P_t \) can be described by a linear system where each inequality has size at most \( s_t \).

**Theorem 2.3.1.** (Cook et al., 1998) For any proper class of polyhedra, the optimisation problem is polynomially solvable if and only if the separation problem is polynomially solvable.

Next we give a brief introduction to a field of research related to combinatorial optimisation called polyhedral combinatorics.

Polyhedral combinatorics is the area of research where the polyhedron of a combinatorial optimisation problem is studied (Grötschel and Padberg, 1985). In polyhedral combinatorics and also in linear programming, we are interested in the polytopes defined by the constraints of optimisation problems. In linear programming, the polytope is usually defined as the bounded intersection of halfspaces, whereas in polyhedral combinatorics the polytope is usually just expressed as the convex hull of a set of vertices. If the polytope of the combinatorial optimisation problem can be expressed as an intersection of some halfspaces, the problem can be solved as a linear optimisation problem (Papadimitriou and Steiglitz, 1982). In fact, one of the main theoretical problems in polyhedral combinatorics is finding a matrix \( A \) and a vector \( b \) to express the polyhedron of the combinatorial optimisation problem with a polynomial number of linear inequalities, to be able to apply LP techniques (Schrijver, 1995) for solving the problem. After the introduction of linear programming in the 1940s and 1950s, many researchers studied combinatorial optimisation problems that could be reduced to linear programming. However for some combinatorial optimisation problems such as the traveling salesman problem, unless it is proven that \( P = \mathbb{NP} \), it is not known whether all the instances of these problems can be solved efficiently using linear programming or not.
Various Formulations for the TSP

Various formulations have been suggested for the TSP. The TSP has been modeled as single-commodity flow, multi-commodity flow, job scheduling and other combinatorial optimisation problems. These formulations differ greatly in number of variables and constraints. In the following sections some of the well-known formulations for the TSP are given and briefly compared. In comparison to the classical formulation suggested by Dantzig, Fulkerson, and Johnson (1954), most of the formulations have a greater number of variables that makes up for their smaller number of constraints. In addition to comparing the compactness of the formulations, it is of significant importance to compare the strength of their LP relaxation in providing close to optimal solutions as well. This is done by comparing the size of projected LP relaxation polytopes of the formulations into one common space. The tightness of a polytope is regarded as an indication of the strength of the formulation (Padberg and Sung, 1991).

Let $G = (V, E)$ be a complete graph of nodes in $V = \{i, \ldots, n\}$ where $E = \{(i, j) | i, j \in V, i \neq j\}$. Throughout this chapter it is assumed that $c_{ij}$, where $(i, j)$ is an edge in $E$, indicates the cost of edge $(i, j)$ unless specified otherwise.

3.1 Flood

Merrill Flood is believed to be the first researcher who publicized the TSP in the mathematical community (Lawler et al., 1985). Flood (1956) formulated the TSP as an integer programming formulation. Assuming that the TSP tours start at node 1, each tour corresponds to a sequence of $n$ edges. Let $x_{ijt}$, for $1 \leq i, j \leq n, i \neq j$ and $1 \leq t \leq n$, be a 0-1 variable. The value of $x_{ijt}$ is set equal to one if edge $(i, j)$ is in the $t^{th}$ position of the tour, and is set to zero otherwise. The formulation suggested by Flood is given by constraints (3.1) – (3.4).
3.2 Dantzig, Fulkerson and Johnson

The most well-known formulation for the TSP is a 0-1 linear formulation suggested by Dantzig, Fulkerson and Johnson (1954) that was originally designed for the ATSP. This formulation which we refer to as the DFJ formulation is given below.

DFJ : \( \min \sum_{j=1}^{n} \sum_{i=1}^{n} c_{ij} x_{ij} \)

subject to:

\[
\begin{align*}
\sum_{i=1}^{n} x_{ij} &= 1, \quad \forall j = 1, \ldots, n, \\
\sum_{j=1}^{n} x_{ij} &= 1, \quad \forall i = 1, \ldots, n, \\
\sum_{i,j \in S} x_{ij} &\leq |S| - 1, \quad \forall S \subseteq V, 2 \leq |S| \leq n - 1, \\
x_{ij} &\in \{0, 1\}, \quad \forall i, j.
\end{align*}
\]
cycles or subtours. The DFJ formulation has $2^{(n-1)} + n - 1$ constraints and $n(n-1)$ variables. The exponential number of the subtour elimination constraints makes the formulation impractical to solve directly. Therefore in order to solve the problem, Dantzig, Fulkerson and Johnson solved the LP relaxation of the formulation with subtour elimination constraints relaxed, and added the violated constraints and solved the problem again until getting a complete tour [Applegate et al. 2006]. This method gave rise to cutting plane methods and branch and bound algorithms (Balas and Toth 1985). The polytope defined by the LP relaxation of the DFJ formulation is known as the subtour elimination polytope (SEP).

3.3 Miller, Tucker and Zemlin

Miller, Tucker, and Zemlin (1960) suggested a mixed integer linear programming formulation defining a more general variation of the TSP. In this formulation, node 1 is considered to be the home node, where the traveling salesman must return to exactly $t$ times, including the final return. During each visit home, the salesman must visit no more than $p$ nodes different from home. Let $u_i$ indicate the position of node $i$ in the tour, where $i \neq 1$. The formulation by Miller et al. (1960) which we refer to as the MTZ formulation is given below.

$$
\text{MTZ} : \min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij}x_{ij}
$$

subject to:

$$
\sum_{i=2}^{n} x_{i1} = t, \quad (3.9)
$$

$$
\sum_{i=2}^{n} x_{1i} = t, \quad (3.10)
$$

$$
\sum_{i=1}^{n} x_{ij} = 1, \quad \forall j = 2, ..., n, \quad (3.11)
$$

$$
\sum_{j=1}^{n} x_{ij} = 1, \quad \forall i = 2, ..., n, \quad (3.12)
$$

$$
u_i - u_j + px_{ij} \leq p - 1, \quad 2 \leq i \neq j \leq n, u_i \geq 0, \quad (3.13)
$$

$$
x_{ij} \in \{0,1\}, \quad \forall i, j. \quad (3.14)
$$

Constraints (3.9) and (3.10) guarantee that the home city is visited exactly $t$ times. Constraint (3.11) and (3.12) make sure that cities other than home are visited exactly once.
and constraint (3.13) discards the tours that visit more than \( p \) cities. If \( t \) is set to 1 and \( p \geq n - 1 \), the formulation models the classical TSP. This formulation has \((n^2 - n + 2)\) constraints and \((n^2 - 1)\) variables.

Gouveia and Pires (1999) have suggested an improved formulation based on the MTZ formulation for the ATSP. In this formulation, they have lifted some facet defining inequalities for the ATSP, that are not dominated by the subtour elimination constraints of the DFJ. They have reported some computational results supporting the dominance of the LP relaxation of this formulation to the DFJ formulation.

### 3.4 Fox, Gavish and Graves

Fox, Gavish and Graves (1980) suggested a very compact formulation which models the TSP as a single commodity flow problem using 0-1 variables \( x_{ijt} \) (Fox et al., 1980). Let \( x_{ijt} = 1 \) if arc \((i, j)\) is assigned to the \( t^{th} \) position of the tour and let \( x_{ijt} = 0 \) otherwise. It can be assumed, without loss of generality, that the cost of an edge is constant regardless of its position in the tour: \( c_{ijt} = c_{ij}, \forall i, j \) and \( t \). The formulation by Fox et al. (1980) which we refer to as the FGG formulation is given below.

\[
\text{FGG : } \min \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{t=1}^{n} c_{ijt} x_{ijt}
\]

subject to:

\[
\sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{t=1}^{n} x_{ijt} = n, \quad (3.15)
\]

\[
\sum_{j=2}^{n} \sum_{t=1}^{n} t x_{ijt} - \sum_{j=1}^{n} \sum_{t=1}^{n} t x_{jit} = 1, \quad \forall i = 2, ..., n, \quad (3.16)
\]

\[
x_{ijt} \in \{0, 1\}, \quad \forall i, j, t. \quad (3.17)
\]

Constraint (3.15) guarantees that each node is visited exactly once and that one edge is assigned to each position of the tour. Constraint (3.16) prevents the formation of subtours by enforcing that if a node is entered at stage \( t \), it should be left at stage \( t + 1 \). Assuming that node 1 is the origin of the tour, an extra set of conditions was imposed by Orman and Williams (2007), forcing node 1 to be left only at stage 1 and entered only at stage \( n \). These conditions are:

\[
x_{i1t} = 0, \quad \forall t \neq n, \quad (3.18)
\]

\[
x_{1jt} = 0, \quad \forall t \neq 1, \quad (3.19)
\]

\[
x_{ijt} = 0, \quad i \neq 1, i \neq j. \quad (3.20)
\]
Some variables such as $x_{iit}$ are always equal to zero for all $i$, $t$, and therefore can be dropped from the formulation. Regardless of such zero variables, there are $2(n-1) + (n+1)(n-2)^2$ zero-one variables and $n$ linear constraints in the formulation [Padberg and Sung, 1991].

### 3.5 Wong

Wong (1980) modeled the TSP as a multicommodity flow problem. He defined two types of commodities indicated by two sets of variables. Let $y^k_{ij}$ and $z^k_{ij}$ be the two types of variables indicating the flow of commodity $k$ in $(i,j)$ where $k = 2, ..., n$. The source node for commodity $y$ (commodity $z$) is node one (node $k$) and the sink is node $k$ (node one). The formulation by [Wong (1980)] is as follows.

**Wong:** \[
\min \sum_i \sum_j c_{ij} x_{ij}
\]

subject to:

\[
\sum_{i=1}^{n} x_{ij} = 1, \quad \forall j = 1, ..., n, \quad (3.21)
\]

\[
\sum_{j=1}^{n} x_{ij} = 1, \quad \forall i = 1, ..., n, \quad (3.22)
\]

\[
\sum_j (y^k_{ij} - y^k_{ji}) = \begin{cases} 
1, & \text{if } i = 1, \\
-1, & \text{if } i = k, \\
0, & \text{if } i \neq 1 \text{ and } k,
\end{cases} \quad k = 2, ..., n, \quad (3.23)
\]

\[
\sum_j (z^k_{ij} - z^k_{ji}) = \begin{cases} 
-1, & \text{if } i = 1, \\
1, & \text{if } i = k, \\
0, & \text{if } i \neq 1 \text{ and } k,
\end{cases} \quad k = 2, ..., n, \quad (3.24)
\]

\[
y^k_{ij} \leq x_{ij}, \quad \forall i, j, k, \quad (3.25)
\]

\[
z^k_{ij} \leq x_{ij}, \quad \forall i, j, k, \quad (3.26)
\]

\[
y^k_{ij} \geq 0, \quad \forall i, j, k, \quad (3.27)
\]

\[
z^k_{ij} \geq 0, \quad \forall i, j, k, \quad (3.28)
\]

\[
x_{ij} \in \{0, 1\}, \quad \forall i, j. \quad (3.29)
\]

Constraints (3.21) and (3.22) are the same as constraints (3.5) and (3.6) in the DFJ formulation. Constraints (3.23) and (3.24) make sure that a single commodity $y$ (commodity $z$) flows from node 1 (node $k$) to node $k$ (node 1). This formulation has $2(n^3 + n^2 + n)$ constraints, $n(n - 1)$ integer variables and $2n(n - 1)^2$ continuous variables.
3.6 Picard and Queyranne

Picard and Queyranne [1978] have given a formulation for the time dependant traveling salesman problem (TDTSP) which is a generalization of the TSP. This formulation is derived from a single-machine scheduling problem where the machine has an initial status of 0 and a final status of $n+1$. In each status, one particular job from 1 to $n$ is processed. Let $x_{ijt}$ be a 0-1 variable indicating the change for machine configuration from status $i$ to $j$ at time $t$, where $i \neq j$. The variables can be interpreted as traveling from node $i$ to $j$ at time $t$. Let $c_{ijt}$ be the cost of setup change of the machine from job $i$ to job $j$ at time $t$. If the setup costs are assumed to be independent of time, $c_{ijt} = c_{ij}$, for all $t$, the problem reduces to the TSP. The TDTSP formulation by Picard and Queyranne (1978) is given below.

\[
PQ : \min \sum_{j=1}^{n} c_{0j0}x_{0j0} + \sum_{t=1}^{n-1} \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ijt}x_{ijt} + \sum_{j=1}^{n} c_{j,n+1,n}x_{j,n+1,n} \\
\text{subject to:} \\
\sum_{j=1}^{n} x_{0j0} = 1, \quad (3.30) \\
x_{0j0} - \sum_{k=1}^{n} x_{jk1} = 0, \quad \forall j = 1, \ldots, n, \quad (3.31) \\
\sum_{i=1}^{n} x_{ijt} - \sum_{k=1}^{n} x_{jk(t+1)} = 0, \quad \forall t = 1, \ldots, n-2, j = 1, \ldots, n, \quad (3.32) \\
\sum_{i=1}^{n} x_{ij(n-1)} - x_{j(n+1)n} = 0, \quad \forall j = 1, \ldots, n, \quad (3.33) \\
x_{0j0} + \sum_{i=1}^{n} \sum_{t=1}^{n} x_{ijt} = 1, \quad j = 1, \ldots, n, \quad (3.34) \\
x_{ijt} \in \{0, 1\}, \quad \forall i, j, t. \quad (3.35)
\]

Constraint (3.30) makes sure that exactly one job is chosen as the initial job at time zero and constraint (3.31) makes sure this job is followed by exactly one other job at time one. Constraint (3.32) guaranties that a job processed at time $t$ would be followed by another job at time $t+1$. Constraint (3.33) assigns exactly one job as the last job. Constraint (3.34) makes sure that all the jobs from 1 to $n$ are assigned to the machine. The TDTSP formulation has $(2n + n(n-1)^2)$ variables and $(n^2 + n + 1)$ constraints.
3.7 Claus

Claus (1984) like Wong (1980) has also modeled the TSP as a multi-commodity flow problem. It was shown by Langevin et al. (1990) that this formulation is a relaxation of the formulation by Wong (1980) in which the $z_{ij}^k$ variables and their related constraints are eliminated. Let some node $s \in V$ be the home node and be duplicated as the sink node $t$. The TSP can then be defined as the problem of finding a Hamiltonian path from node $s$ to $t$ in $G(V,E)$.

Let $V_1 = \{1, \ldots, n\}$, $V = V_1 \cup \{s,t\}$, $V^* = V_1 \cup \{t\}$ and $E = \{(i,j)|\forall i \neq j \in V^1\} \cup \{(s,i),(i,t)|\forall i \in V^1\}$. Let $x_{ij}$ be 1 if $(i,j)$ is in a Hamiltonian path and 0, otherwise. Similar to the formulation by Wong, $y_{ij}^k$ indicates the flow of commodity $k$ in edge $(i,j)$. The formulation involves the flow of $(n+1)$ commodities. Let commodity $k$, be the commodity that is shipped from node $s$ to $k$, where $k = 1, \ldots, n+1$, and let index $(n+1)$ indicate node $t$. The TSP formulation by Claus is given below.

\[\text{Claus} : \min \sum_{(i,j) \in E} c_{ij} x_{ij}\]

subject to:

\[\sum_{j \in V} x_{ij} = \sum_{j \in V} x_{ji} = 1, \quad \forall i \in V^1, \quad (3.36)\]

\[\sum_{i \in V_1} x_{si} = \sum_{i \in V_1} x_{it} = 1, \quad (3.37)\]

\[\sum_{j \in V} y_{ij}^k - \sum_{j \in V} y_{ji}^k = 0, \quad \forall i \in V^1, k \in V^*, i \neq k, \quad (3.38)\]

\[\sum_{i} -y_{ik}^k = -1, \quad \forall k \in V^*, \quad (3.39)\]

\[\sum_{i \in V_1} y_{si}^k = 1, \quad \forall k \in V^*, \quad (3.40)\]

\[\sum_{i} y_{ki}^k = 0, \quad \forall k \in V^*, \quad (3.41)\]

\[y_{ij}^k - x_{ij} \leq 0, \quad \forall i, j, k, \quad (3.42)\]

\[x_{ij}, y_{ij}^k \in \{0,1\}, \quad \forall i, j, k. \quad (3.43)\]

Constraints (3.38)-(3.41) guarantee that there is a path from the source node $s$ to every node $k \in V^*$, carrying a unit flow of commodity $k$. For some variables we have $y_{ik}^k = 0$, for all $i \in V, \forall k \in V^1$ and also $y_{ki}^k = 0$, for all $i, k \in V^*$. Since these variables are always equal to zero, they can be dropped from the formulation. The formulation has $(n^3 - 2n^2 + 6n - 3)$ constraints, $n(n-1)$ integer variables and $(n^3 + 2n)$ continuous variables.
3.8 Carr

Carr (1996) suggested an LP formulation that models the LP relaxation of the DFJ formulation and is called the cycle-shrink model. Given some \( k \in V \), let \( V_k = \{k+1, \ldots, n\} \) and let \( G_k = (V_k, E_k) \) be a subgraph of the complete graph \( G \) that is induced by \( V_k \). For each edge \( e \in E_k \) we define a decision variable \( x^k_e \).

Let \( x^0 \) be an incidence vector of a Hamiltonian cycle \( H^0(x^0) \) in \( G \). Let \( H^1(x^0) \) be a Hamiltonian cycle in \( G^1 \) that is formed by removing vertex 1 from \( H^0(x^0) \) and connecting its neighbours with an edge. Similarly, let \( H^k(x^0) \) be a Hamiltonian cycle in \( G^k \) that is obtained by removing vertex \( k \) from \( H^{k-1}(x^0) \) in \( G^{k-1} \) and connecting its neighbours. The incidence vector of \( H^0 \) is indicated as \( x^0 = (x^0_e | e \in E) \) and for any \( k \in \{1, \ldots, n-3\} \) the incidence vector of \( H^k \) is \( x^k = (x^k_e | e \in E_k) \). The solutions to Carr’s formulation are sequences of nodes that have been removed from initial Hamiltonian cycles in \( G \) and are represented by \( x = (x^0, x^1, \ldots, x^{n-3}) \). The cycle-shrink relaxation formulation by Carr is:

\[
Carr : \min \sum_{e \in E} c_e x^0_e
\]

subject to:

\[
x^0_e \geq 0, \quad \forall e \in E, \quad (3.44)
\]

\[
\sum_{e \in \delta(j) \cap E_k} x^k_e = 2, \quad \forall k \in \{0, \ldots, n-3\}, \forall j \in V_k, \quad (3.45)
\]

\[
x^{k-1}_e - x^k_e \leq 0, \quad k \in \{1, \ldots, n-3\}, e \in E_k. \quad (3.46)
\]

Carr (1996) has shown that a feasible solution for the cycle-shrink model satisfies all the subtour elimination constraints. Let \( \tau_n = \sum_{k=4}^n (k-1)(k-2)/2 \). We will need this parameter \( \tau_n \) later. The Cycle-shrink model has \( (\tau_{n+1} + (n+3)(n-2)/2) \) constraints and \( \tau_{n+1} \) variables.

3.9 Comparison of the TSP Formulations

The major drawback of the DFJ formulation is the exponential number of its constraints. This has motivated researchers to suggest more compact formulations which have polynomial number of constraints. The number of constraints and variables of some of TSP formulations are shown in Table 3.1.

Although most of these formulations have a polynomial number of constraints, they are not better than the DFJ formulation in terms of the quality of their LP relaxation.
### Table 3.1: Number of Constraints and Variables of the TSP Formulations

<table>
<thead>
<tr>
<th>Formulation</th>
<th>Constraints</th>
<th>Integer Variables</th>
<th>Continuous Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flood</td>
<td>$n^2$</td>
<td>$n(n + 1)(n - 1)$</td>
<td></td>
</tr>
<tr>
<td>DFJ</td>
<td>$2^{(n-1)} + n - 1$</td>
<td>$n(n - 1)$</td>
<td></td>
</tr>
<tr>
<td>MTZ</td>
<td>$n^2 - n + 2$</td>
<td>$n(n - 1)$</td>
<td>$(n - 1)$</td>
</tr>
<tr>
<td>FGG</td>
<td>$n$</td>
<td>$n(n - 1)$</td>
<td>$n(n - 1)(n + 1)$</td>
</tr>
<tr>
<td>Wong</td>
<td>$2(n^3 + n^2 + n)$</td>
<td>$n(n - 1)$</td>
<td>$2n(n - 1)^2$</td>
</tr>
<tr>
<td>PQ</td>
<td>$(n^2 + n + 1)$</td>
<td>$(2n + n(n - 1)^2)$</td>
<td></td>
</tr>
<tr>
<td>Claus</td>
<td>$n^3 + n^2 + 3n$</td>
<td>$2n^2 + 2n$</td>
<td></td>
</tr>
<tr>
<td>Carr</td>
<td>$\tau_{n+1} + (n + 3)(n - 2)/2$</td>
<td>$\tau_{n+1}$</td>
<td></td>
</tr>
</tbody>
</table>

solution. To compare the solvability of different formulations when used in LP-based solutions methods, Padberg and Sung (1991) have used a special transformation technique to map polytopes given by other formulations into the DFJ formulation variable space. This affine transformation is used to project a polyhedron in $R^n$ into $R^m$, where $m \leq n$. After finding the projection of different formulations into the DFJ variable space, the sizes of the projected polytopes were compared with that of the DFJ formulation. They showed that the DFJ formulation gives the tightest polytope for the TSP. Langevin et al. (1990) have classified some TSP formulations as single-commodity, two-commodity and multi-commodity formulations, and compared them in terms of their LP relaxation values. Let $v_{LP}(A)$ indicate the optimal values of the LP relaxation for some formulation $A$. Langevin et al. (1990) have shown that

$$v_{LP}(DFJ) = v_{LP}(Wong) = v_{LP}(Claus),$$

$$v_{LP}(MTZ) \leq v_{LP}(DFJ).$$

In a similar and more recent study, Orman and Williams (2007) have compared the strength of eight LP relaxations of the TSP formulations illustrated by computational results on small problems. Let the polytope given by the LP relaxation of some projected formulation $A$ be denoted as $P(A)$. Some of their results are shown in Table 3.2. Gouveia and Volks (1995) have shown that $v_{LP}(PQ) \geq v_{LP}(FGG)$ and $P(PQ) \subset P(FGG)$.

In the next chapter we give the multistage insertion formulation for the STSP by Arthanari (1983) and multistage insertion formulation for the ATSP by Usha (1998). We discuss some characteristics of these formulations and a polytope defined based on the multistage insertion formulation for the STSP.
### Table 3.2: Comparison of the Projected Polytopes with the SEP

<table>
<thead>
<tr>
<th>Formulation</th>
<th>Comparison to SEP</th>
</tr>
</thead>
<tbody>
<tr>
<td>MTZ</td>
<td>$SEP \subseteq P(MTZ)^*$</td>
</tr>
<tr>
<td>FGG</td>
<td>$SEP \subseteq P(FGG)^*$</td>
</tr>
<tr>
<td>Claus</td>
<td>$SEP = P(Claus)^* \dagger$</td>
</tr>
<tr>
<td>Carr</td>
<td>$SEP = P(Carr)^* \ddagger$</td>
</tr>
<tr>
<td>Wong</td>
<td>$SEP = P(Wong)^* \dagger$</td>
</tr>
</tbody>
</table>

* Padberg and Sung (1991)
† Orman and Williams (2007)
‡ Carr (1996)
The Multistage Insertion Formulation and the Pedigree Polytope

In this chapter we describe a model for the TSP known as the multistage insertion formulation (MI). We give the MI formulation for the STSP by Arthanari (1983) in Section 4.1, and the MI formulation for the ATSP by Usha (1998) in Section 4.1.2. In Section 4.2 we give the definitions for a combinatorial object called *pedigree* that is defined based on the structure of the MI formulation. The convex hull of all the pedigrees for a given problem, called the *pedigree polytope*, is defined in this section as well.

4.1 The Multistage Insertion Formulation (MI)

Let $K_n = (V_n, E_n)$ indicate a complete graph of $n \geq 4$ vertices, where $V_n = \{1, \ldots, n\}$ is the set of vertices labeled in an arbitrary order, and $E_n = \{(i, j) \mid i, j \in V_n, i < j\}$ is the set of edges. The cardinality of $E_n$ is denoted by $p_n = n(n-1)/2$. Therefore $\tau_n$ defined earlier in Section 3.8 can also be written as $\tau_n = \sum_{k=4}^{n} p_{k-1}$. We assign a unique edge label $l_{ij} = p_{j-1} + i$ to each edge $e = (i, j) \in E_n$. For a subset $F \subseteq E_n$ the characteristic vector of $F$ is represented by $x_F \in \mathbb{R}^{p_n}$. Assuming that edges in $E_n$ are ordered in increasing order of their edge labels, the characteristic vector is defined as follows.

$$x_F(e) = \begin{cases} 1, & \text{if } e \in F, \\ 0, & \text{otherwise.} \end{cases}$$

For a subset $S \subseteq V_n$, we define $E(S) = \{(i, j) \in E_n \mid i, j \in S\}$. The set of edges with one node in $S$ and the other node in $V_n \setminus S$ is denoted as $\delta(S)$. Let $T_k$ be a TSP tour of size $k$ (also called a *k-tour*) corresponding to a Hamiltonian cycle in graph $K_k = (V_k, E_k)$,
4.1 The Multistage Insertion Formulation (MI)

where \(1 \leq k \leq n\). Let \(v_i \in V_k\) for \(3 \leq i \leq k\) indicate the \(i^{th}\) node in the \(k\)-tour \(T_k\). We denote the tour as \(T_k = [v_1, v_2, ..., v_k, v_1]\).

### 4.1.1 The MI Formulation for the STSP and the MI Polytope

The MI formulation is based on \(n-3\) stages of node insertions into the 3-tour \(T_3 = [1, 2, 3, 1]\). This tour is expanded to an \(n\)-tour as the nodes from 4 to \(n\) are inserted successively into the tour. Through \(n-3\) stages, at each stage \(k\) an edge \((i, j)\) from the tour is chosen for insertion of node \(k\). The decision of choosing an edge for insertion in stage \(k-3\) for \(4 \leq k \leq n\) is represented by variable \(x_{ijk}\), for all \(1 \leq i < j < k\), such that

\[
x_{ijk} = \begin{cases} 
1, & \text{if node } k \text{ is inserted between nodes } i \text{ and } j, \\
0, & \text{otherwise.}
\end{cases}
\]

The first stage of the insertion starts with the decision of inserting node 4 into one of the edges of \(T_3\), i.e. either \((1, 2)\), \((1, 3)\), or \((2, 3)\). Assuming that node 4 is inserted into some edge \((i_4, j_4) \in E_3\), available edges in the second stage would then be \(A_5 = \{(1, 2), (1, 3), (2, 3)\} \cup \{(i_4, 4), (j_4, 4)\} - \{(i_4, j_4)\}\). Generally, the tour that is formed in stage \(k\), depends on available edges in stage \(k-1\) and also on the choice of edge \((i_{k-1}, j_{k-1})\) for insertion of node \(k-1\). The set of edges in each stage can be shown as \(A_k = A_{k-1} \cup \{(i_{k-1}, k), (j_{k-1}, k)\} - \{(i_{k-1}, j_{k-1})\}\). Since through all these \(n-3\) stages, each node \(4 \leq k \leq n\) is inserted into one edge only, we have the condition

\[
\sum_{1 \leq i < j < k} x_{ijk} = 1, \quad \forall 4 \leq k \leq n. \quad (4.1)
\]

Each of the edges of the initial 3-tour can be used for the insertion of at most one node \(4 \leq k \leq n\). This condition can be shown as constrain (4.2)

\[
\sum_{k=4}^{n} x_{ijk} \leq 1, \quad \forall 1 \leq i < j \leq 3. \quad (4.2)
\]

Assuming that in stage \(k-3\) of insertion, an edge \((i, j)\) is needed for insertion of node \(k\), this edge is required to be constructed in one of the stages prior to stage \(k-3\). Furthermore, no node other than \(k\) should be inserted into \((i, j)\). Firstly, if \((i, j) \notin E_3\), in some stage prior to \(k-3\), edge \((i, j)\) needs to be formed by inserting \(j\) into either edge \((r, i)\) or \((i, s)\), where \(1 \leq r < i\) and \(i < s < j\). This requires that the sum \(\sum_{r=1}^{i-1} x_{rij} + \sum_{s=i+1}^{j-1} x_{isj}\) be equal to one. Secondly, this edge could be used for insertion by only one node \(k > i\). This condition can be modeled as constraint (4.3).
4.1 The Multistage Insertion Formulation (MI)

$$- \sum_{r=1}^{i-1} x_{rij} - \sum_{s=i+1}^{j-1} x_{isj} + \sum_{k=j+1}^{n} x_{ijk} \leq 0, \quad \forall 4 \leq j < n, 1 \leq i < j. \quad (4.3)$$

Let $c_{ij}$ denote the cost of an edge $(i,j) \in E$. Insertion of node $k$ into edge $(i,j)$, would replace edge $(i,j)$ with two new edges $(i,k)$ and $(j,k)$. This replacement increases the total cost of the tour by $C_{ijk} = c_{ik} + c_{jk} - c_{ij}$. The MI formulation minimises the total incremental cost of the tour that is made by the node insertions. Since the initial cost of the 3-tour ($c_{12} + c_{13} + c_{23}$) is the same in all the tours of a given instance, it is not included in the objective function of the MI formulation. The complete MI formulation is given below (Arthanari, 1983).

$$\min \sum_{k=4}^{n} \sum_{1 \leq i < j < k} C_{ijk} x_{ijk}$$

subject to:

$$\sum_{1 \leq i < j \leq k-1} x_{ijk} = 1, \quad \forall 4 \leq k \leq n, \quad (4.1)$$

$$\sum_{k=4}^{n} x_{ijk} \leq 1, \quad \forall 1 \leq i < j \leq 3, \quad (4.2)$$

$$- \sum_{r=1}^{i-1} x_{rij} - \sum_{s=i+1}^{j-1} x_{isj} + \sum_{k=j+1}^{n} x_{ijk} \leq 0, \quad \forall 1 \leq i < j, 4 \leq j < n, \quad (4.3)$$

$$x_{ijk} \in \{0,1\}, \quad \forall 1 \leq i < j < k, 4 \leq k \leq n. \quad (4.3)$$

The number of constraints in the MI formulation is $p_n + n - 3$, and the number of variables is $\tau_n = \sum_{k=4}^{n} p_{k-1}$. By relaxing the integrality constraint from the MI formulation and adding the following constraint, the MI-relaxation problem is defined.

$$- \sum_{r=1}^{i-1} x_{rin} - \sum_{s=i+1}^{n-1} x_{isn} \leq 0, \quad i = 1, ..., n - 1. \quad (4.4)$$

Throughout this thesis we use the terms LP relaxations of the MI formulation and the MI-relaxation interchangeably. Although constraint (4.4) is redundant, it is added to the model because of the special meaning of its corresponding slack variables. The polytope given by the LP relaxation of the MI formulation is denoted by $P_{MI}(n)$. Let $u_{ij}$, $1 \leq i < j \leq n$, be the slack variables corresponding to the inequalities in the MI formulation. The slack variables of the inequalities (4.2), (4.3), and (4.4) define the corresponding tour as given in (4.5).
4.1 The Multistage Insertion Formulation (MI)

\[ if \ u_{ij} = \begin{cases} 
1, & \text{then edge } (i, j) \text{ is present in the tour,} \\
0, & \text{then edge } (i, j) \text{ is not present in the tour.} 
\end{cases} \] (4.5)

This is shown in Arthanari and Usha [2000]. Next we give some definitions on the structure of the MI formulation which is used in bringing out characteristics of the polytope given by the LP relaxation of the MI formulation.

**Definition 4.1.1.** (Arthanari and Usha, 2001) Let \( e_k \) be a vector of size \( 1 \times k \) with all its coordinates equal to one. The matrix corresponding to equation (4.1) is denoted as \( E_{[n]} \) and is constructed as follows.

For \( n = 4 \) we have \( E_4 = e_{3 \times 2} \div 2 = (111) \), and for \( n = 5 \) we have \( E_5 = \)

\[
\begin{pmatrix}
1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 & 1
\end{pmatrix}
\]. In general, \( E_{[n]} \) can be constructed recursively as shown below.

\[
E_{[n]} = \begin{pmatrix}
\frac{e_{3 \times 2}}{2} & 0 & 0 & \ldots & 0 & 0 \\
0 & \frac{e_{4 \times 3}}{2} & 0 & \ldots & \vdots & \vdots \\
0 & 0 & \ddots & 0 & \vdots & \vdots \\
\vdots & \vdots & 0 & \ddots & 0 & \vdots \\
0 & 0 & \ldots & 0 & \frac{e_{(n-2) \times (n-3)}}{2} & 0
\end{pmatrix}
= \begin{pmatrix}
E_{[n-1]} & 0 \\
0 & e_{(n-1)(n-2)} \div 2
\end{pmatrix}
\].

Let \( A^{(n)} = \begin{pmatrix}
I_{p_{n-1}} \\
-M_{n-1}
\end{pmatrix} \), where \( M_i \) corresponds to coefficients of constraints (4.2)-(4.4), the matrix corresponding to constraints (4.2)-(4.4) is denoted as \( A_{[n]} \) and it is constructed as follows.

\[
A_{[n]} = \begin{pmatrix}
A^{(4)} \\
0 \\
\vdots \\
0 \\
\vdots \\
0
\end{pmatrix}
= \begin{pmatrix}
A_{[n-1]} \\
A^{(n)}
\end{pmatrix}.
\]

Let \( U \) denote the vector of slack variables in the MI formulation, and let \( C^T = (C_{124}, C_{134}, C_{234}, C_{125}, \ldots, C_{12n}, \ldots, C_{(n-2)(n-1)n}) \). Based on the definition of \( A_{[n]} \) and \( E_{[n]} \), the MI-relaxation can also be defined as Problem 4.1.1.

**Problem 4.1.1.** \( \min C^T X \)

\( s.t. \)

\[
\begin{pmatrix}
E_{[n]} & 0 \\
A_{[n]} & I
\end{pmatrix}
\begin{pmatrix}
X \\
U
\end{pmatrix}
= \begin{pmatrix}
e_{n-3} \\
ed_3 \\
0
\end{pmatrix},
\]

36
4.1 The Multistage Insertion Formulation (MI)

\( X, U \geq 0. \)

**Theorem 4.1.1.** (Arthanari and Usha, 2000) Any integer feasible solution to Problem 4.1.1 corresponds to an \( n \)-tour.

Let \( c^T \) be defined such that \( C^T = -c^T A[n] \). For any solution \((X, U)\) to Problem 4.1.1, we have \( A[n]X + IU = \begin{pmatrix} e_3 \\ 0 \end{pmatrix} \). Multiplying both sides by \( c^T \) we have \( c^T U = c^T \begin{pmatrix} e_3 \\ 0 \end{pmatrix} - c^T A[n]X = c^T \begin{pmatrix} e_3 \\ 0 \end{pmatrix} + C^T X. \) And since \( c^T \begin{pmatrix} e_3 \\ 0 \end{pmatrix} = c_{12} + c_{13} + c_{23} \) is a constant, it can be dropped from the objective function and therefore we can define Problem 4.1.2 which is equivalent to Problem 4.1.1 as follows.

**Problem 4.1.2.** \( \min c^T U \)

\[ \begin{align*}
\text{s.t.} & \quad \begin{pmatrix} E[n] & 0 \\ A[n] & I \end{pmatrix} \begin{pmatrix} X \\ U \end{pmatrix} = \begin{pmatrix} e_{n-3} \\ e_3 \\ 0 \end{pmatrix}, \\
X, U & \geq 0.
\end{align*} \]

**Definition 4.1.2.** (Arthanari and Usha, 2001) The polytope \( u(n) \) is defined as \( u(n) = \{ U | U = \begin{pmatrix} e_3 \\ 0 \end{pmatrix} - A[n]X, X \in \zeta(n) \} \), where \( \zeta(n) = \{ X | E[n]X = e_{n-3}, X \geq 0 \} \).

**Theorem 4.1.2.** (Arthanari and Usha, 2000) The projection of the polytope \( u(n) \)

\[ \begin{align*}
U \geq 0, X \in \zeta(n), A_nX + U = \begin{pmatrix} e_3 \\ 0 \end{pmatrix}
\end{align*} \]

onto the space of the slack variables \( U \), is a subset of the subtour elimination polytope.

**Theorem 4.1.3.** (Arthanari and Usha, 2000) Let \( U^s \begin{pmatrix} e_3 \\ 0 \end{pmatrix} \) be any integer \( X^s \in \zeta(n) \). Then \( U^s \) is an extreme point of \( u(n) \).

We compare the performance of the MI formulation with other TSP formulations through solving some test problems in Chapter 6. In the next section we give the MI formulation for the ATSP by Usha (1998).

### 4.1.2 The MI formulation for the ATSP

Similar to the STSP, the ATSP can be modeled using a multistage insertion procedure in which a complete tour is created by inserting nodes sequentially into an initial tour of two nodes. Let \( A \) be a set of directed edges or arcs, where \( A = \{ (i,j) | i, j \in V_n \setminus V_2, i \neq j \} \). Starting with a directed 2-tour \([1, 2, 1]\), where the cost of arc \((1, 2)\) may be different from
4.1 The Multistage Insertion Formulation (MI)

arc \((2,1)\), nodes from 3 to \(n\) are sequentially inserted into the tour. A complete directed \(n\)-tour is thus constructed within \(n-2\) insertion stages. The decision variables of the MI formulation for the ATSP are defined as follows.

\[
x_{ijk} = \begin{cases} 
1, & \text{if node } k \text{ is inserted into arc } (i,j), \text{ for } 1 \leq i \neq j < k, 3 \leq k \leq n, \\
0, & \text{otherwise.} 
\end{cases} 
\]  

(4.6)

Let \(c_{ij}\) indicate the cost of some arc \((i,j) \in A\) which could be different from the cost of arc \((j,i) \in A\). By inserting a node \(3 \leq k \leq n\) into arc \((i,j) \in A\), this arc is replaced by arcs \((i,k)\) and \((k,j)\). Such an insertion increases the total cost of the tour by \(C_{ijk} = c_{ik} + c_{kj} - c_{ij}\). The MI formulation for the ATSP minimises the total incremental cost of the insertions. The MI formulation for the ATSP (Usha, 1998) is given below.

\[
\min \sum_{k=3}^{n} \sum_{1 \leq i \neq j \leq k-1} C_{ijk}x_{ijk} 
\]

subject to:

\[
\sum_{1 \leq i \neq j \leq k-1} x_{ijk} = 1, \quad 3 \leq k \leq n 
\]  

(4.7)

\[
\sum_{k=3}^{n} x_{ijk} \leq 1, \quad 1 \leq i \neq j \leq 2 
\]  

(4.8)

\[
- \sum_{r=1, r \neq i}^{j-1} x_{irj} + \sum_{k=j+1}^{n} x_{ijk} \leq 0, \quad 1 \leq i < j, 3 \leq j < n - 1 
\]  

(4.9)

\[
- \sum_{s=1, s \neq i}^{j-1} x_{sij} + \sum_{k=j+1}^{n} x_{jik} \leq 0, \quad 1 \leq i < j, 3 \leq j < n - 1 
\]  

(4.10)

\[
- \sum_{r=1, r \neq i}^{n-1} x_{irn} \leq 0, \quad 1 \leq i \leq n - 1 
\]  

(4.11)

\[
- \sum_{s=1, s \neq i}^{n-1} x_{sin} \leq 0, \quad 1 \leq i \leq n - 1 
\]  

(4.12)

\[
x_{ijk} \in \{0,1\}, 1 \leq i < j < k, 3 \leq k \leq n. 
\]  

(4.13)

Constraint (4.7) makes sure that all the nodes from 3 to \(n\) are inserted into the tour. Constraint (4.8) guarantees that the two arcs of the initial 2-tour are used for insertion by at most one node. Constraints (4.9) and (4.10) make sure that a node \(k\) is only inserted into some arc \((i,j)\) which is created in one of the stages earlier and not used before the insertion of \(k\). Constraints (4.11) and (4.12) make sure that node \(n\) is inserted into some arc of the tour.
4.2 The Pedigree Polytope

Theorem 4.1.4. ([Usha, 1998]) There is a 1-1 correspondence between n-tours and the integer feasible solutions to the MI formulation for the ATSP.

We compare the performance of this model with other TSP formulations in Chapter 6 by solving some test problems. In the next section we give the definitions for a combinatorial object called pedigree and a polytope called the pedigree polytope.

4.2 The Pedigree Polytope

In this section we give a definition for pedigrees and the pedigree polytope, and discuss some of their characteristics. We also give some definitions that will be used in the following chapters of the thesis. In Chapter ?? we will discuss some problems and algorithms on the pedigree polytope.

The MI formulation for the STSP has given rise to the definition of a combinatorial object called pedigree. Let $H_n$ be the set of all Hamiltonian cycles of $K_n = (V_n, A_n)$ and let $H^k \in H_n$, for all $3 \leq k \leq n$.

Definition 4.2.1. ([Arthanari, 2005]) Let $e_{k+1}$ be an edge in $H^k$ where $3 \leq k \leq n$. The vector $W = (e_4, ..., e_n)$ is called a pedigree if and only if there exists a $H \in H_n$ where $H$ is obtained from $H^3$ by a sequence of insertions such that:

- $H^4$ results from the insertion of 4 into $e_4$ in $H^3$,
- ...
- $H$ results from the insertion of $n$ into $e_n$ in $H^{n-1}$.

Theorem 4.2.1. ([Arthanari, 2006]) Pedigrees are in 1-1 correspondence with the STSP tours.

[Arthanari, 2006] showed that the non-adjacency testing of two pedigrees in the pedigree polytope can be conducted in polynomial time, while the same problem in the STSP polytope is known to be NP-complete ([Papadimitriou, 1978]). [Arthanari, 2007] compared the pedigree polytope with the STSP polytope. Some of the key differences between the two polytopes presented in that study are shown in Table 4.1.

Definition 4.2.2. Consider a pedigree $W$ with $e_k \in E_{k-1}$ being its $(k-3)^{rd}$ component for $4 \leq k \leq n$. We define $X = (x_4, ..., x_n) \in B^{\tau_n}$ as the characteristic vector of $W$, where the vector $x_k \in B^{\tau_{k-1}}$ is the indicator of edge $e_k$. Let the edge label of $e_k$ be denoted as $l_k$. The vector $x_k$ has 1 in its coordinate number $l_k$, and zeros elsewhere.

Following example gives a pedigree and its corresponding characteristic vector.
Example 4.2.1. Consider the following pedigree \( W = ((1, 2), (2, 4), (4, 5), (1, 3)) \) that corresponds to the 7-tour \( T_7 = [1, 4, 6, 5, 2, 3, 7, 1] \). This pedigree can be constructed through four insertion stages. In the first stage, node 4 is inserted into \( e_4 = (1, 2) \), creating edges \((1, 4)\) and \((2, 4)\). In the next stage \( e_5 = (2, 4) \) is used for the insertion of 5 producing edges \((2, 5)\) and \((4, 5)\). Node 6 and 7 are inserted into \( e_6 = (4, 5) \) and \( e_7 = (1, 3) \), respectively in the next two stages. The corresponding characteristic vector of \( W \) is \( X = (x_4, x_5, x_6, x_7) \), where \( x_4 = (1, 0, 0) \), \( x_5 = (0, 0, 0, 1, 0) \), \( x_6 = (0, 0, 0, 0, 0, 0, 0, 0, 1) \) and \( x_7 = (0, 1, 0, 0, 0, 0, 0, 0, 0) \).

Definition 4.2.3. (Arthanari 2005) Let \( P_n = \{ X \in B^n | X \text{ is the characteristic vector of } W, \text{ the pedigree of } H \in H_n \} \). The convex hull of \( P_n \) is called the pedigree polytope and is denoted as \( \text{conv}(P_n) \).

Definition 4.2.4. (Naddef and Pulleyblank 1981) A 0-1 polytope is called a combinatorial polytope if the midpoint of the line joining any pair of non-adjacent extreme points is the midpoint of the line joining some other pair of non-adjacent extreme points.

Theorem 4.2.2. (Arthanari 2005) The Pedigree polytope is a combinatorial polytope.

Next we give a definition that is used in adjacency testing of pedigrees in the pedigree polytope.

Definition 4.2.5. (Arthanari 2005) Given some edge \( e = (i, j) \in E_n \), a set \( G(e) \) is called the set of generators of \( e \), and it is defined as follows.

\[
G(e) = \begin{cases} 
\delta(i) \cap E_{j-1} & \text{if } j \geq 4, \\
E_3 \setminus \{e\} & \text{otherwise.}
\end{cases}
\]

Example 4.2.2. Given edge \((4, 5) \in E_5\), we are interested in finding the set of generators for this edge. We have \( i = 4 \) and \( j = 5 \). Since \( j = 5 \geq 4 \), we have \( G((4, 5)) = \delta(4) \cap E_4 \). We have \( E_4 = \{(1, 2), (1, 3), (2, 3), (1, 4), (2, 4), (3, 4)\} \), and \( \delta(4) = \{(1, 4), (2, 4), (3, 4), (4, 5)\} \), and therefore \( G(e) = \{(1, 4), (2, 4), (3, 4)\} \).


4.2 The Pedigree Polytope

**Definition 4.2.6.** (Arthanari, 2006) Let \( y(e) \) be the indicator of \( e \in E_k \). Given a pedigree \( W = (e_1, \ldots, e_k) \) and its corresponding characteristic vector \( X \in P_k \), and an edge \( e \in E_k \), we call \( (W, e) = (e_1, \ldots, e_k, e) \) an extension of \( W \) in case \( (X, y(e)) \in P_{k+1} \).

**Remark 4.2.1.** Given \( W = (e_1, \ldots, e_k) \) and \( e \in E_k \), a vector \( (W, e) \) is an extension of \( W \) if and only if there exists some \( 4 \leq a \leq k \), such that \( e_a \) is a generator of \( e \).

In the next section we give some lemmas and definitions on the adjacency of pedigrees in the pedigree polytope.

### 4.2.1 Adjacency in the Pedigree Polytope

Let \( X^{[1]} \neq X^{[2]} \in P_n \) be the characteristic vectors corresponding to two pedigrees \( P^{[1]} \neq P^{[2]} \) and let \( \overline{X} = \frac{1}{2}X^{[1]} + \frac{1}{2}X^{[2]} \). By Theorem 4.2.2 the pedigree polytope is a combinatorial polytope and therefore \( P^{[1]} \) and \( P^{[2]} \) are known to be non-adjacent if it can be proven that for two other pedigrees \( P^{[3]} \neq P^{[4]} \), which are different from \( P^{[1]} \) and \( P^{[2]} \), we have \( \overline{X} = \frac{1}{2}X^{[3]} + \frac{1}{2}X^{[4]} \), where \( X^{[3]} \) and \( X^{[4]} \in P_n \) are the characteristic vectors of \( P^{[3]} \) and \( P^{[4]} \) respectively. In other words, if some of the components of \( P^{[1]} \) and \( P^{[2]} \) can be exchanged to create two other pedigrees, they are proven to be non-adjacent. We illustrate this through the next example.

**Example 4.2.3.** Given two pedigrees \( P^{[1]} = \{(1, 2), (1, 4), (4, 5), (2, 3)\} \), and \( P^{[2]} = \{(1, 2), (2, 4), (4, 5), (5, 6)\} \), we can have \( \overline{X} = \{(1, 0, 0), (0, 0, 0, 0, 0, 0, 0, 0, 0, 1), (0, 0, \frac{1}{2}, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, \frac{1}{2})\} \). \( \overline{X} \) can also be written as the convex combination of the characteristic vectors of \( P^{[3]} = \{(1, 2), (1, 4), (4, 5), (5, 6)\} \), and \( P^{[4]} = \{(1, 2), (2, 4), (4, 5), (2, 3)\} \). This is sufficient to conclude that \( P^{[1]} \) and \( P^{[2]} \) are non-adjacent.

Next we give some definitions that we use to prove two given pedigrees are adjacent or non-adjacent.

**Definition 4.2.7.** (Arthanari, 2006) Given \( X^{[1]}, X^{[2]} \in P_n \), we call \( D = \{q|x_q^{[1]} \neq x_q^{[2]}|, 4 \leq q \leq n\} \) the set of discordant components or the set of discordants.

**Example 4.2.4.** Considering the two pedigrees given in Example 4.2.3, all the components of \( P^{[1]} \) and \( P^{[2]} \) are the same except their second component i.e. \( (1, 4) \) in \( P^{[1]} \), and \( (2, 4) \) in \( P^{[2]} \), for \( k = 5 \) and their forth component, \( (2, 3) \) in \( P^{[1]} \), and \( (5, 6) \) in \( P^{[2]} \) for \( k = 7 \). Therefore the set of discords of the two pedigrees \( P^{[1]} \) and \( P^{[2]} \) is \( D = \{5, 7\} \).

**Definition 4.2.8.** (Arthanari, 2006) Let \( i = 3 - i \), for \( i = 1, 2 \). Given two pedigrees \( P^{[1]} \) and \( P^{[2]} \) of size \( n \) with their corresponding set of discordants \( D \), a graph \( G_R = G(D, E_R) \) is called their graph of rigidity. The set \( E_R \) in this graph is defined according to the following rules. There is an edge connecting some node \( r \in D \) to some other node \( s \in D \) if either

- no generators of \( e_{ir} \) is available in the pedigree \( P^{[i]} \), for some \( i = 1, 2 \), or

--
4.2 The Pedigree Polytope

Figure 4.1: Graph of Rigidity for the Pedigrees in Example 4.2.6

- $e_{[i]_8} = e_{[i]_r}$, for some $i = 1, 2$.

Example 4.2.5. Considering the following two pedigrees:

- $P^{[1]} = ((1, 2), (1, 4), (2, 3), (4, 5), (1, 3))$ and
- $P^{[2]} = ((1, 2), (2, 4), (4, 5), (5, 6), (6, 7))$,

their corresponding set of discords is $D = \{5, 6, 7, 8\}$. Since the forth component of $P^{[1]}$ and the third component of $P^{[2]}$ are the same, there is an edge between 6 and 7 in $G_R$. Also since none of the generators of $e_8$ in $P^{[2]}$, which are $(1, 6), (2, 6), (3, 6), (4, 6)$ and $(5, 6)$, exist in $P^{[1]}$ there is an edge between 7 and 8. Therefore the set of edges in the graph of rigidity is $\{(6, 7), (7, 8)\}$.

Theorem 4.2.3. [Arthanari 2006] Given two pedigrees $P^{[1]} \neq P^{[2]}$ of size $n$, if the cardinality of their set of discords is equal to one they are adjacent in $\text{conv}(P_n)$. If the two pedigrees differ in more than one component, they are adjacent if and only if their corresponding graph of rigidity is connected.

Example 4.2.6. Given the following two pedigrees

- $P^{[1]} = ((1, 2), (2, 4), (4, 5), (5, 6))$ and
- $P^{[2]} = ((2, 3), (2, 4), (1, 2), (1, 6))$,

their set of discords is $D = \{4, 6, 7\}$. There is an edge between 4 and 6 in the graph $G_R$ since $e_{[4]_4} = e_{[6]_6}^2$. There is also an edge between 6 and 7 as no generator of $(1, 6)$ exists in $P^{[1]}$ (and also no generator of $(5, 6)$ exists in $P^{[2]}$). Therefore the graph of rigidity is connected (Figure 4.1) and the pedigrees are adjacent.

In the case that two pedigrees differ in more than one component, their non-adjacency (adjacency) can also be proven by showing whether swapping their components and creating two other pedigrees is possible (not possible). [Arthanari (2006)] has given an algorithm that determines the adjacency or non-adjacency of a given pair of pedigrees which is polynomial in time.

Example 4.2.7. Considering the pedigrees given in Example 4.2.5 the pedigrees are not adjacent because the graph of rigidity is not connected. The non-adjacency of these pedigrees can also be observed by exchanging their second component that creates two new pedigrees given below as pedigrees $P^{[3]}$ and $P^{[4]}$. 

42
\[ P^{[3]} = ((1, 2), (2, 4), (2, 3), (4, 5), (1, 3)) \] and
\[ P^{[4]} = ((1, 2), (1, 4), (4, 5), (5, 6), (6, 7)) \]

### 4.3 Conclusion

In this chapter we presented the MI formulations for the STSP and the ATSP. We also introduced the pedigree polytope and discussed the adjacency problem in the pedigree polytope. We will use the concepts and definitions in this Chapter in the next chapters. In the following chapters, we will study some problems in the pedigree polytope, perform some computational experiments using the MI formulation, and introduce some heuristics designed based on pedigree structure. Very limited computational experiments have been performed on the MI formulation so far. The remainder of this thesis includes some computational experiments on this formulation compared to other TSP models, solutions methods, and algorithms.

In Chapter 5 we give the membership problem of a solution to the LP relaxation of the MI model in the pedigree polytope. In Chapter 6 we compare the LP relaxation of the MI formulation with other TSP formulations in terms of LP relaxation value, solution time, and number of iterations. We also perform some computational experiments on exact algorithms and heuristics designed based on the MI formulation in Chapter 7 and 8.
5

The Pedigree Polytope and the Membership Problem

In this chapter we study the membership problem of a solution to the LP relaxation of the MI formulation in the pedigree polytope. We define this problem in Section 5.1 and give a necessary condition and some sufficient conditions for a solution to the LP relaxation of the MI formulation, to belong to the pedigree polytope. In Section 5.2 we show a procedure to generate feasible solutions to the LP relaxation of the MI formulation. We use these solutions to check the necessary and sufficient conditions. In Section 5.3 we give an LP model, feasibility of which is equivalent to the sufficient condition being satisfied.

Haerian Ardekani and Arthanari (2008) gave a numerical illustration of an algorithm for checking the necessary condition for the membership of a solution in the pedigree polytope. In Section 5.4 we walk through this algorithm for a solution to the LP relaxation of the MI formulation. The question whether the necessary condition for membership in the pedigree polytope is sufficient is raised in Arthanari (2008). We answer this question in the negative in Section 5.5 by providing a numerical example for which the necessary condition is not sufficient. The conclusions are given in Section 5.6.

5.1 The Membership Problem in the Pedigree Polytope

Arthanari (2008) has studied the problem whether or not some feasible solution to the LP relaxation of the MI formulation also belongs to the pedigree polytope. In this section we give some definitions and algorithms to check the membership of a given solution in the pedigree polytope. Recall the definition of the MI polytope denoted as $P_{MI}(n)$ in Section 4.1.1.

Definition 5.1.1. (Arthanari, 2006) For a given $k$, where $4 \leq k \leq n$, any solution $X = (x_4, ..., x_n) \in P_{MI}(n)$ restricted to the first $k - 3$ stages of insertion, is denoted as $X/k = (x_4, ..., x_k)$ and belongs to $P_{MI}(k)$. 

44
### 5.1 The Membership Problem in the Pedigree Polytope

#### Table 5.1: The \( x_{ijk} \) Values for the MI-relaxation Solution in Example 5.1.1

<table>
<thead>
<tr>
<th>( k )</th>
<th>( x_{124} )</th>
<th>( x_{135} )</th>
<th>( x_{145} )</th>
<th>( x_{246} )</th>
<th>( x_{356} )</th>
<th>( x_{456} )</th>
<th>( x_{157} )</th>
<th>( x_{148} )</th>
<th>( x_{269} )</th>
<th>( x_{3510} )</th>
<th>( x_{4710} )</th>
<th>( x_{56710} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>1</td>
<td>3/4</td>
<td>1/4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>2/4</td>
<td></td>
<td>1/4</td>
<td>1/4</td>
<td>1/4</td>
<td>1/4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td>2/4</td>
<td></td>
<td></td>
<td></td>
<td>1/4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1/4</td>
<td></td>
<td></td>
<td>1/4</td>
<td></td>
<td></td>
<td></td>
<td>1/4</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1/4</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Remark 5.1.1.** If some solution to the MI-relaxation problem can be expressed as a convex combination of characteristic vectors of pedigrees, we can conclude that it belongs to the pedigree polytope.

In the following example we show some solution \( X/k \) that is feasible to the LP relaxation of the MI formulation and can be expressed a convex combination of the characteristic vectors of some pedigrees of size \( k \). This is sufficient to conclude that this solution belongs to the pedigree polytope.

**Example 5.1.1.** Consider \( X/5 = ((1,0,0),(0,\frac{3}{4},0,\frac{1}{4},0,0)) \) corresponding to the solution given in Table 5.1. We can express \( X/5 \) as a convex combination of the characteristic vectors \( X_1 = ((1,0,0),(0,1,0,0,0)) \) and \( X_2 = ((1,0,0),(0,0,0,1,0,0)) \), using a weight vector \( \lambda = (\frac{3}{4},\frac{1}{4}) \). Characteristic vectors \( X_1 \) and \( X_2 \) correspond to pedigrees \( W^{1} = ((1,2),(1,3)) \) and \( W^{2} = ((1,2),(1,4)) \), respectively. Therefore we can conclude that \( X/5 \) is a member of the pedigree polytope \( \text{conv}(P_5) \).

**Definition 5.1.2.** (Arthanari, 2006) Given some \( X \in P_{MI}(n) \), the membership problem checks whether \( X/k + 1 \) belongs to \( \text{conv}(P_{k+1}) \), given that \( X/k \in \text{conv}(P_{k}) \).

Next we give some definitions for a problem called the FAT\(_k(\lambda)\) problem for a given \( X/k \). Feasibility of this problem is a sufficient condition for the membership of \( X/k \) in the pedigree polytope \( \text{conv}(P_{k}) \). We give the definitions for a multilayered network and some maximum flow problems defined in this network which we solve for checking a necessary condition for \( X/k \in \text{conv}(P_{k}) \).

#### 5.1.1 The FAT\(_k(\lambda)\) Problem

Arthanari (2006) showed that given some \( X/k \in \text{conv}(P_{k}) \), if a certain FAT problem, called the FAT\(_k(\lambda)\) problem, in some bipartite network is feasible then \( X/k + 1 \) belongs to \( \text{conv}(P_{k+1}) \). Given some \( X \in P_{MI}(n) \), and \( X/k \in \text{conv}(P_{k}) \), let \( \lambda \in R_{+}^{|P_{k}|} \) be a weight vector such that \( X/k \) can be written as a convex combination of \( X^{\tau} \in P_{k} \), i.e.
5.1 The Membership Problem in the Pedigree Polytope

\[ \Lambda_k(X) = \{ \lambda \in R_+^{[P_k]} : \sum_{r \in I(\lambda)} \lambda_r X^r = X/k, \sum_{r \in I(\lambda)} \lambda_r = 1 \} \]

where \( I(\lambda) \) is the index set of positive coordinates of \( \lambda \). Given a \( \lambda \in \Lambda_k(X) \), the FAT\(_k(\lambda)\) problem is defined as follows.

**Definition 5.1.3.** Given some \( X \in P_{MI}(n) \) where \( X/k \in \text{conv}(P_k) \), and given a weight vector \( \lambda \in \Lambda_k(X) \), the FAT\(_k(\lambda)\) problem is to find a feasible solution satisfying the demand in the sink nodes of the bipartite network that is defined as follows.

- **Origins** = \( \{ X^\alpha | \alpha \in I(\lambda) \} \), with the supply of \( \lambda_\alpha \) for all \( \alpha \),
- **Sinks** = \( \{ e_\beta | E_k | x_{k+1}(e_\beta) > 0 \} \), with demands equal to \( x_{k+1}(e_\beta) \), for all \( \beta \),
- The set of arcs \( \{ (X^\alpha, e_\beta) \in \text{Origins} \times \text{Sinks} | X^\alpha \text{ has a generator of } e_\beta \} \).

The feasibility of the FAT\(_k(\lambda)\) problem is equivalent to having a convex combination of pedigree extensions \( (W, e) \), where \( W \) is a pedigree in \( \text{conv}(P_n) \), and where this combination satisfies all the supply, capacity and demand restrictions imposed by \( X/k + 1 \). However, if for some \( \lambda \) the problem is not feasible, we cannot conclude that \( X/k + 1 \) is not a convex combination of pedigrees.

**Example 5.1.2.** Consider the \( x_{ijk} \) values for the X/6 solution given in Table 5.1. The FAT\(_k(\lambda)\) network based on X/6 is illustrated in Figure 5.1. The supply of the origin nodes are shown next to each pedigree \( W^\alpha \) in this figure. A feasible solution to the FAT\(_k(\lambda)\) problem is shown next to the arcs. Since the demands in all the sink nodes are satisfied, the problem is feasible, and we can conclude that \( X/6 \in \text{conv}(P_6) \).

[Arthanari (2008)] has shown that a necessary condition for a MI-relaxation solution to be expressible as a convex combination of pedigrees, can be associated with the existence of a multi-commodity flow with the optimum value equal to one in some multilayered network. This network is constructed through \( n - 4 \) stages. In the next section we give the definition of this network.
5.1 The Membership Problem in the Pedigree Polytope

5.1.2 The Multilayered Flow Network

Given some \( X \in P_{M}(n) \) and given that \( X/k \in \text{conv}(P_{k}) \) for some \( k < n \), a multilayered network called \( N_{k} \) is constructed for checking a necessary condition for \( X/k + 1 \in \text{conv}(P_{k+1}) \). This necessary condition will be defined in Section 5.1.4. The network \( N_{k} \) is made up of \( k - 2 \) layers and is constructed in \( k - 4 \) stages as follows.

The set of nodes in the network \( N_{k} \) corresponds to the \( x_{ijk} \) variables in \( X \) with positive values. The network \( N_{k} \) is made up of \( k - 2 \) layers. The set of nodes in the \((k - 3)^{rd} \) layer of the network is \( V_{[k-3]} = \{(k : i, j) | x_{ijk} > 0\} \). The set of arcs in the network \( N_{k} \) can be defined recursively. We start with \( k = 4 \), the set of nodes in \( N_{4} \) is \( V_{[1]} = \{(4 : i, j) | x_{ij4} > 0\} \cup V_{[2]} = \{(5 : i, j) | x_{ij5} > 0\} \), and the set of arcs is defined as \( A_{[4]} = \{((4 : i, j), (5 : r, s)) | (i, j) \in G((r, s))\} \). The capacities of the arcs in \( A_{[4]} \) are set equal to the node capacity of starting node of each arc. We define a FAT problem in this network, and check if this problem is feasible. If not then we can conclude that \( X/5 \notin \text{conv}(P_{5}) \) and consequently conclude that \( X/n \notin \text{conv}(P_{n}) \) (Arthanari 2005). If the FAT problem is feasible, we find the rigid and dummy arcs in \( A_{[4]} \) and set their capacities of the rigid arcs equal to their flow, and dummy arcs are deleted. The network \( N_{4} \) is then declared well-defined. In Example 5.1.3 we show how to construct the network \( N_{4} \) for a given \( X \).

Example 5.1.3. Consider the solution \( X \) given in Table 5.1. For the network \( N_{4} \), we have \( V_{[1]} = \{(4 : 1, 2)\} \), and \( V_{[2]} = \{(5 : 1, 3), (5 : 1, 4)\} \). Since \( (1, 2) \) is a generator for both \((1, 3)\) and \((1, 4)\), the set of arcs in \( N_{4} \) is \( A_{[4]} = \{((4 : 1, 2), (5 : 1, 3)), ((4 : 1, 2), (5 : 1, 4))\} \). The \( N_{4} \) network is shown in Figure 5.2 Since \( N_{4} \) is feasible and both the arcs are rigid, we have the capacities as \( \frac{3}{4} \) and \( \frac{1}{4} \) respectively.

Given that the network \( N_{k-1} \) is well-defined, we can proceed to define the network \( N_{k} \) recursively. The set of nodes in the network \( N_{k} \) is the union of nodes in the \( N_{k-1} \) and the set \( V_{[k-2]} \). Similarly the set of arcs in the network \( N_{k} \) is the union of the arcs in \( N_{k} \) and the set of arcs between the layers \( k - 3 \) and \( k - 2 \), that are newly added to \( N_{k} \) at this stage. We refer to the potential arcs between the last two layers as links. Similar to the set \( A_{[4]} \), the set of the links in the network \( N_{k} \) is given by \( \{((k : e), (k + 1 : e')) \in V_{[k-3]} \times V_{[k-2]}\} \). For each link in this set, we find a flow capacity and remove the links with zero capacity from

Figure 5.2: The \( N_{4} \) Network
5.1 The Membership Problem in the Pedigree Polytope

Figure 5.3: (a) The $N_7$ Network, and (b) the Restricted Network $N_6([7:5,6], [8:4,6])$

the network. The capacity of each link is set equal to the maximum flow in the restricted network of the link that is given in Definition 5.1.4. The links with positive maximum flow in their restricted networks are added to $N_k$ as arcs of the network.

**Definition 5.1.4.** Given some $k < n$, let link $L = (e_{\alpha}, e_{\beta}) \in V_{k-3} \times V_{k-2}$ be a link in some network $N_k$, where $e_{\alpha} = (r, s)$, and $e_{\beta} = (i, j)$. A network $N_{k-1}(L)$ is called the restricted network of $N_k$ corresponding to the link $L$, if it is induced by the set of nodes in $N_{k-1}$, without any node that may be in the form of:

1. $[l : e_{\beta}]$: for max $(4, j) < l < k$,

2. $[l : e_{\alpha}]$: for max $(4, s) < l < k$,

3. $[j : e]$: $e \notin G(e_{\beta})$, if $e_{\beta} \in E_k \setminus E_3$; otherwise $[4, e_{\beta}]$,

4. $[s : e]$: $e \notin G(e_{\alpha})$, if $e_{\alpha} \in E_{k-1} \setminus E_3$; otherwise $[4, e_{\alpha}]$, or

5. all the nodes in $V_{k-3} \setminus \{[k : e_{\alpha}]\}$.

We define a sink node $t = e_{\alpha}$ with demand value equal to $x(e_{\alpha})$, and source nodes $s \in \{[4 : u, v] \in N_{k-1}(L)\}$ with supply values equal to $x(s)$. We solve the maximum flow problem in the restricted networks corresponding to any link in the $N_k$ network. The maximal flow into a link $L$ is denoted as $C(L)$. We set the capacity of the links in the $N_k$ network equal to their corresponding $C(L)$ values, and remove the links with their $C(L)$ values equal to zero from the network. In the following example we give a restricted network $N_6(L)$ corresponding to a link $L$ in some network $N_7$.

**Example 5.1.4.** Consider the network $N_7$ for the MI-relaxation solution given in Table 5.1 that is shown in part (a) of Figure 5.3. Given link $([7 : 5,6], [8 : 4,6])$ we form the restricted
5.1 The Membership Problem in the Pedigree Polytope

network corresponding to this link. To form the restricted network $N_6([7 : 5, 6], [8 : 4, 6])$, we need to remove node $[6 : 3, 5]$ since $(3, 5)$ is not a generator of $(4, 6)$, and we also remove node $[6 : 2, 4]$ because $(2, 4)$ is not a generator of $(5, 6)$. We also discard all the nodes in the form of $[7 : i, j]$, for $1 \leq i < j < 7$, from the network. The arcs connected to all these removed nodes are also removed from the network. The restricted network is shown in part (b) of Figure 5.3. The maximal flow in the link is equal to 0.25 therefore $C([7 : 5, 6], [8 : 4, 6]) = 0.25$.

Next we give the definition of some $F_k$ problem that is defined using the newly added arcs in the network $N_k$. Infeasibility of this $F_k$ problem is a sufficient condition for $X/k+1 \notin \text{conv}(P_{k+1})$.

5.1.3 The $F_k$ Problem

A FAT problem called the $F_k$ problem is defined in the bipartite network given by the last two layers of some $N_k$ network, using the $C(L)$ values corresponding to the links in the network. The $F_k$ problem is is to find a feasible solution that satisfies demands of the sink nodes in the network defined below.

- Origins: All the nodes in $V_{[k-3]}$, with supply values equal to $x_{ijk}$ for each node,
- Sinks: All the nodes in $V_{[k-2]}$, with demands equal to $x_{ijk}$ values,
- Arcs: $\{L \in V_{[k-3]} \times V_{[k-2]} | C(L) > 0\}$, with arc capacities equal to $C(L)$.

Remark 5.1.2. [Arthanari 2006] If the $F_4$ problem is infeasible, we conclude that $X/5 \notin \text{conv}(P_5)$, and also $X/n \notin \text{conv}(P_n)$.

Theorem 5.1.1. [Arthanari 2008] Given $X \in P_{MI}(n)$ and given $X/k \in \text{conv}(P_k)$ for some $k < n$, if the $F_k$ problem is infeasible then we can conclude that $X/k+1 \notin \text{conv}(P_{k+1})$ and also that $X/n \notin \text{conv}(P_n)$.

If the $F_k$ problem is feasible we proceed with solving a multicommodity flow problem in the $N_k$ network. For doing so, first we need to identify the rigid and dummy arcs in the solution to the $F_k$ problem and set the capacity of these arcs equal to their optimal flow. We use the frozen flow finding algorithm by Gusfield [1988] for this purpose. The dummy arcs are discarded from the $N_k$ network. The necessity of removing dummy arcs from the network is shown in an example Haerian Ardekani and Arthanari [2008]. Example 5.1.5 illustrates the $F_5$ problem for $X$ given in Table 5.1.

Example 5.1.5. The set of origins and destinations in $F_5$ are $V_{[2]}$ and $V_{[3]}$ respectively. Since $(1, 3)$ is not a generator of $(4, 5)$ and $(1, 4)$ is not a generator of $(3, 5)$, the corresponding arcs are not links and are not included in the set of arcs of $N_5$. The set of arcs
5.1 The Membership Problem in the Pedigree Polytope

The solution to the $F_5$ problem is shown in Figure 5.4. All the arcs are rigid since their flow is same in all the feasible solutions to the $F_5$ problem. The arc $([5:1,4],[6:2,4])$ has zero flow and is a dummy arc and therefore removed from the network.

Next we give the definition of the multicommodity flow problem in the network $N_k$. We solve this problem to check a necessary condition for $X/k + 1 \in \text{conv}(P_{k+1})$, given $X/k \in \text{conv}(P_k)$ for some $k < n$.

5.1.4 A Necessary Condition for $X/k \in \text{conv}(P_k)$

We solve a multicommodity flow problem over an enlarged network called $N$. In this network we define a unique commodity corresponding to each link in the final layer of the network $N_k$, and also a sink node corresponding to each commodity. The demand in each sink node is set equal to the $C(L)$ values for each link $L$. We also define one source node with commodity supplies equal to one unit corresponding to each commodity. The network $N$ is shown schematically in Figure 5.5. A multicommodity flow problem is solved in $N$ to check a necessary condition for $X/k + 1 \in \text{conv}(P_{k+1})$. We give the problem below.

Let $V(N_k)$ be the set of nodes in the network $N_k$, and let $A(N)$ be the set of arcs in the network $N$. Let $A_{\text{new}}$ be the set of links in the last layer of $N_k$ and also the arcs from the source node to nodes in the first layer. Let $c_a$ denote the capacity of an arc $a$ in the $N$ network. Let $S$ be the set of commodities where each commodity corresponds to a link. Let $f_{a}^{s}$ be the flow of commodity $s$ in arc $a$, and $v^{s}$ be the total flow of commodity $s$ into its corresponding sink node. We define an upper bound on $f_{a}^{s}$ variables for arc $a$ and commodity $s$ as follows.

\[ u_{a}^{s} = \begin{cases} 
1, & \text{if } a \in A_{\text{new}}, \\
c_a, & \text{if } a \in N_{k-1}(L_{s}) \text{ or } \text{conv}(P_{k-1}), \\
0, & \text{otherwise}.
\end{cases} \]

Let $S_{F}$ denote the set of arcs with rigid flow in $F_k$, and let $b_s$ be the demand for some commodity $s$ which is equal to $C(L)$ values of its corresponding link. The multicommodity
5.1 The Membership Problem in the Pedigree Polytope

![Diagram](image)

Figure 5.5: Enlarged Network $N$

The membership problem is given below (Arthanari, 2008).

$$\max z = \sum_{s \in S} v^s$$

subject to:

$$0 \leq f_a^s \leq u_a^s, \quad \forall s \in S, \forall a \in A(N), \quad (5.1)$$

$$\sum_{u \ni a=(u,v)} f_a^s - \sum_{w \ni a=(v,w)} f_a^s = 0, \quad \forall v \in V(N_k), \forall s \in S, \quad (5.2)$$

$$\sum_{s \in S} f_a^s \leq c_a, \quad \forall a \in A(N), \quad (5.3)$$

$$\sum_{s \in S} \sum_{u \ni a=(u,v)} f_a^s \leq x(v), \quad \forall v \in V(N_k), \quad (5.4)$$

$$v^s - \sum_{s \in S} \sum_{u \ni a=(u,v) \in A_{new}} f_a^s = 0, \quad \forall s \in S, \quad (5.5)$$

$$v^s = b_s, \quad s \in S_F. \quad (5.6)$$

Constraints $[5.1]$ ensure that the flows in the arcs do not exceed the arc capacities for any commodity. Constraint $[5.2]$ is the flow conservation constraint in each node regarding each commodity. Constraint $[5.3]$, also known as the bundle constraint, ensures that the total commodity flows in an arc do not exceed the arc capacity. Constraint $[5.4]$ guarantees that the total flow into each node does not exceed its capacity. Constraints $[5.5]$ and $[5.6]$ guarantee that the flow through the rigid arcs in the last layer is equal to the sink demands corresponding to these arcs.
Theorem 5.1.2. (Arthanari, 2008) Given $X/k \in \text{conv}(P_k)$ for some $k < n$, if $X/k + 1 \in \text{conv}(P_{k+1})$, then the multicommodity flow problem in the expanded network of $N_k$ has an optimal solution of one.

If the total flow in the optimal solution is not equal to one, it is concluded that $X/k + 1 \notin \text{conv}(P_{k+1})$ and so $X/n \notin \text{conv}(P_n)$. A flowchart of an algorithm for checking the necessary condition is shown in Figure 5.6.

This algorithm includes subroutines that all run in polynomial time. These subroutines are the frozen flow finding algorithm and an algorithm for defining the restricted network. Arthanari (2008) has shown that these algorithms can be performed in polynomial time.

We give Algorithm 6 that given $X/k \in \text{conv}(P_k)$ and $N_k$ being well-defined for any $k > 4$, checks the necessary condition for $X/k + 1 \in \text{conv}(P_{k+1})$.

**Algorithm 6 Checking the Necessary Condition for $X/k + 1 \in \text{conv}(P_{k+1})$**

1. identify the links between layers $k - 3$ and $k - 2$
2. find $C(L)$ values for each link $L$ by solving the maximum flow problem in the restricted network $N_{k-1}(L)$
3. solve the $F_k$ problem
   - if the $F_k$ problem is feasible then
     - identify dummy and rigid arcs in $F_k$ using Algorithm 4
   - else
     - return "The solution is not a member of the pedigree polytope."
4. end if
5. construct the expanded network of $N_k$
6. define and solve the multicommodity flow problem in the expanded network of $N_k$
   - if the optimal flow is equal to one then
     - return "The necessary condition for membership is satisfied."
   - else
     - return "The solution is not a member of the pedigree polytope."
7. end if

Next we show that the necessary condition is also sufficient for $X/k + 1 \in \text{conv}(P_{k+1})$ for $k = 4$ and 5.

5.1.5 A Sufficient Condition for $X/5 \in \text{conv}(P_5)$, and $X/6 \in \text{conv}(P_6)$

**Definition 5.1.5.** (Arthanari, 2008) Consider a layered network $N_l$. A path in the network corresponding to a $X^r \in P_{l+1}$, given by $[4 : e^r_4] \rightarrow [5 : e^r_5] \rightarrow \ldots \rightarrow [l + 1 : e^r_{l+1}]$ where $X^r$ is the characteristic vector of $(e^r_4, \ldots, e^r_{l+1})$, is called a pedigree path.

**Remark 5.1.3.** The feasibility of the $F_4$ problem in the $N_4$ network is sufficient to conclude that $X/5$ is a member of the $\text{conv}(P_5)$ for some given $X$ (Arthanari, 2008).
5.1 The Membership Problem in the Pedigree Polytope

Given some $k > 4$, and $X/k \in \text{conv}(P_k)$

Identify links $L$ between the layers $(k-3)$ and $(k-2)$

Find capacities for the links by solving a maximum flow problem in the restricted networks of the links

Solve the $F_k$ problem in the bipartite network of the last two layers

Is it feasible?

Yes

Identify rigid and dummy arcs and update the network and arc capacities

Add the source and the sink nodes and construct $N$

Solve the multicommodity flow problem in $N$

Is the flow equal to one?

Yes

The necessary condition is satisfied

No

The necessary condition is not satisfied

Figure 5.6: A Flow Chart for Checking the Necessary Condition for $X/k + 1 \in \text{conv}(P_{k+1})$
For \( k = 5 \), the necessary condition for \( X/k + 1 \in \text{conv}(P_{k+1}) \) is also sufficient. It is shown in the following theorem.

**Theorem 5.1.3.** ([Haerian Ardekani and Arthanari, 2010]) Given \( X \in P_{MF}(n) \), for \( k = 5 \), if the multicommodity flow problem has a feasible flow with \( z^* = 1 \), then \( X/6 \in \text{conv}(P_6) \).

**Proof:** Consider the enlarged network \( N \) of \( N_5 \). Suppose \( f \) is feasible for the multicommodity flow problem over \( N \), with total flow into the sinks equal to unity. Restrict the attention to the flow in the portion of the network corresponding to the \( F_5 \) problem. This constitutes a feasible solution to the \( F_5 \) problem. Consider any \( s \in S \), with flow \( v^s > 0 \) into \( s \) and \( L_s = (e, e') \). Since any arc in \( N_4(L) \) for any \( L \) with \( C(L) > 0 \) corresponds to a pedigree in \( P_5 \), there exist(s) pedigree path(s) in \( N_4(L_s) \) bringing a flow of \( v^s \) into \([5 : e]\). Thus we have pedigree paths from source \( o \) to \([5 : e]\) for any \( s \in S \) bringing a flow of \( v^s \). And each of these paths can be extended to a \([6 : e']\) corresponding to a specified \( s \), giving a pedigree in \( P_5 \). Finally, as \( z^* = 1 \), the total weight on these pedigrees (paths) equals unity. Since \( f \) saturates all the node capacities at all nodes in \( N_5 \), we have expressed \( X/6 \) as a convex combination of pedigrees in \( P_6 \). \( \square \)

Having reviewed previous work, we present our findings in the next sections.

### 5.2 Finding a Feasible Solution to the MI-relaxation Problem

We use Algorithm 7 to generate feasible solutions to the MI-relaxation problem. Let \( X^k = (x_{1k}, \ldots, x_{(k-2)(k-1)k}) \in [0,1]^n \) be the generated values for \( x_{ijk} \) variables, for \( 4 \leq k \leq n \), and for \((i,j) \) in \( E_k \). We define a vector \( U^k = (u_{1k}, \ldots, u_{(k-2)(k-1)k}) \) in \( \mathbb{R}^n \) that includes the upper limits for the generated \( x_{ijk} \) values. We also define vector \( \alpha^k = (\alpha^k_{(k-1)}, \ldots, \alpha^k_{(k-2)(k-1)}) \in [0,1]^{(k-2)} \), where \( \alpha^k_{(k-1)} = \sum_{i=t}^{j=1} x_{ij(k-1)} \). Starting with \( k = 4 \) we have \( U^4 = (1,1,1) \), and for \( k > 4 \), the \( U^k \) vector can be built recursively as \( U^k = (U^{k-1} - X^{k-1}, \alpha^k) \). Given \( n \), the size of the instance to be generated, Algorithm 7 returns a feasible solution to the LP relaxation of the MI formulation.

**Algorithm 7** Generate a Feasible Solution to the LP Relaxation of the MI formulation

\[
U^4 \leftarrow (1,1,1)
\]

for \( k = 4 \) to \( n \) do

if \( k > 4 \) then

\[
U^k \leftarrow (U^{k-1} - X^{k-1}, \alpha^k)
\]

end if

Populate \( X^k \) vector coordinates with random \( x_{ijk} \) values such that \( \sum_{(i,j) \in E_k} x_{ijk} = 1 \), and \( 0 \leq x_{ijk} \leq u_{ijk} \).

end for

return \( X \)
5.3 Checking the Sufficient Condition for $X/k \in \text{conv}(P_k)$

Table 5.2: The Relationship Between $x_{ijk}$ and $u_{ijk}$ Values

<table>
<thead>
<tr>
<th></th>
<th>$k = 4$</th>
<th></th>
<th>$k = 5$</th>
<th></th>
<th>$k = 6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i$</td>
<td>$j$</td>
<td>$u_{ijk}$</td>
<td>$x_{ijk}$</td>
<td>$u_{ijk}$</td>
<td>$x_{ijk}$</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>$x_{124}$</td>
<td>$1 - x_{124}$</td>
<td>$x_{125}$</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>1</td>
<td>$x_{134}$</td>
<td>$1 - x_{134}$</td>
<td>$x_{135}$</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>1</td>
<td>$x_{234}$</td>
<td>$1 - x_{234}$</td>
<td>$x_{235}$</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>-</td>
<td>-</td>
<td>$x_{124} + x_{134}$</td>
<td>$x_{145}$</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>-</td>
<td>-</td>
<td>$x_{124} + x_{234}$</td>
<td>$x_{245}$</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>-</td>
<td>-</td>
<td>$x_{134} + x_{234}$</td>
<td>$x_{345}$</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>$x_{125} + x_{135} + x_{145}$</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>$x_{125} + x_{235} + x_{245}$</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>$x_{135} + x_{235} + x_{345}$</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>$x_{145} + x_{245} + x_{345}$</td>
</tr>
<tr>
<td>sum</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

According to Algorithm 7 an $x_{ijk}$ variable is given positive value only if its generators are available, that is if the value of its upper bound, $u_{ijk}$, is greater than zero. The upper bound for a $x_{ijk}$ variable is equal to the availability of its generators. This upper bound is equal to the $x$ values of the generators of $x_{ijk}$ minus the amount by which they have been used up by any other nodes up to this stage. This algorithm runs in $O(n^4)$ time. The relationship between $u$ and $x$ vectors for $k = 4, 5,$ and 6 is shown in Table 5.2.

Remark 5.2.1. The $U^n$ vector for a generated MI-relaxation instance corresponds to the slack variable vector of the MI-relaxation model.

5.3 Checking the Sufficient Condition for $X/k \in \text{conv}(P_k)$

As mentioned in Remark 5.1.1, if for a given feasible solution of the LP relaxation of the MI formulation ($X/n$), we are able to express it as a convex combination of pedigrees in $P_n$, then the sufficient condition for $X/n \in \text{conv}(P_n)$ is satisfied. For doing so we need to have all the possible pedigrees in the $N_{n-2}$ network that is constructed based on $X/n$. Algorithm 8 finds all the pedigrees for a given $X/n$ solution by doing a breadth first search in the corresponding $N_{n-2}$ network.

Algorithm 8 systematically enumerates all the possible pedigrees given some $X/n$ and given the $N_{n-2}$ network. This algorithm runs within $O(n^5)$ time in the worst case, that is when all variables $x_{ijk}$ have equal values for all $k$. Usually most of the $(k - 1)(k - 2)/2$ many variables for each $k$ are equal to zero, which results in much lower running time for the algorithm. The following example illustrates how Algorithm 8 can be used.
Algorithm 8 Finding Pedigrees Given a Solution to the LP Relaxation of the MI formulation

\begin{algorithm}
\begin{algorithmic}
\State $P \leftarrow \{\}$
\State $paths \leftarrow V_{[1]}$
\For{$k = 5$ to $n$}
\State $NewPaths \leftarrow \{\}$
\For{index = 1 to $|paths|$}
\State $p \leftarrow paths(index)$
\For{all $[k : i, j] \in V_{[k-3]}$, where an edge in the $N_{n-2}$ between the last node of $p$ and $[k : i, j]$ exists}
\If{a generator of edge $(i, j)$ exists in $p$}
\State add $(p, [k : i, j])$ to the $NewPaths$
\EndIf
\EndFor
\EndFor
\State $paths \leftarrow NewPaths$
\EndFor
\State return $X$
\end{algorithmic}
\end{algorithm}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure5_7.png}
\caption{The Network for the MI-relaxation Solution in Example 5.3.1}
\end{figure}
5.3 Checking the Sufficient Condition for $X/k \in \text{conv}(P_k)$

Example 5.3.1. Using Algorithm $[7]$ we generate an MI relaxation instance of size 7 with the following $x$ values. $x_{134} = 0.25, x_{234} = 0.75, x_{345} = 1, x_{126} = 0.75, x_{246} = 0.25, x_{357} = 0.25, x_{167} = 0.75$. The network $N$ that is built based on $x$ is illustrated in Figure 5.7. In Step 1 and Step 2 we have $P = \{\}$, paths $= \{[4 : 1, 3], [4 : 2, 3]\}$, $k = 5$, NewPaths $= \{\}$, index $= 1$, and $p = \text{paths}(1)$. We have $V_{[2]} = \{[5 : 3, 4]\}$, and since both $(1, 3)$ and $(2, 3)$ are generators of $(3, 4)$, and also there are edges between the nodes in $V_{[1]}$ to the nodes in $V_{[2]}$, we have paths $= \text{NewPaths} = \{[4 : 1, 3], [5 : 3, 4], [4 : 2, 3], [5 : 3, 4]\}$.

For index $= 2$ and $k = 6$, we have $V_{[3]} = \{[6 : 1, 2], [6 : 2, 4]\}$, and NewPaths $= \emptyset$. Considering $[6 : 1, 2]$ and the paths in the set paths, we can see that the generators of $(1, 2)$ exist in the paths $([4 : 1, 3], [5 : 3, 4])$ and $([4 : 2, 3], [5 : 3, 4])$, therefore we can add the following paths to the set NewPaths so we have NewPaths $= \{([4 : 1, 3], 5 : 3, 4), [6 : 1, 2]), ([4 : 2, 3], 5 : 3, 4), [6 : 1, 2])\}$. Considering node $[6 : 2, 4]$, we can see that the generators for $(2, 4)$ are only available in path $([4 : 2, 3], [5 : 3, 4])$, therefore we have NewPaths $= \text{NewPaths} \cup \{([4 : 2, 3], [5 : 3, 4], [6 : 2, 4]\}$, and paths $= \text{NewPaths}$.

For index $= 3$ and $k = 7$ we have $V_{[4]} = \{[7 : 3, 5], [7 : 1, 6]\}$, and NewPaths $= \emptyset$. Considering $[7 : 3, 5]$ and the paths in paths, we can see that the generators of $(3, 5)$ exist in all the paths in the set paths. Therefore we have NewPaths $= \{([4 : 1, 3], [5 : 3, 4], [6 : 1, 2], [7 : 3, 5]), ([4 : 2, 3], [5 : 3, 4], [6 : 1, 2], [7 : 3, 5]), ([4 : 2, 3], [5 : 3, 4], [6 : 2, 4], [7 : 3, 5])\}$. Considering $[7 : 1, 6]$, the generators for $(1, 6)$ are only available in paths with the component $[6 : 1, 2]$ in them and therefore we add the paths $([4 : 1, 3], [5 : 3, 4], [6 : 1, 2], [7 : 1, 6])$, and $([4 : 2, 3], [5 : 3, 4], [6 : 1, 2], [7 : 1, 6])$ to the NewPaths. We have paths $= \text{NewPaths}$, and $k = 8$. Since $k$ is larger than 7, we terminate the algorithm.

The list of the paths in the NewPaths set are the pedigrees in the network. These pedigrees are as follows.

- $W_1 = ((1, 3), (3, 4), (1, 2), (3, 5))$
- $W_2 = ((2, 3), (3, 4), (1, 2), (3, 5))$
- $W_3 = ((2, 3), (3, 4), (2, 4), (3, 5))$
- $W_4 = ((1, 3), (3, 4), (1, 2), (1, 6))$
- $W_5 = ((2, 3), (3, 4), (1, 2), (1, 6))$

Given all the pedigrees corresponding to some $X/n$, we need to find some vector $\lambda$ to express $X/n$ as a convex combination of these pedigrees in $P_n$. According to Arthanari (2006), if we find a $\lambda$ vector for which the FAT$_k(\lambda)$ problem is feasible, then the sufficient condition is satisfied. We define Problem 5.3 which finds such a $\lambda$ vector if it exists. First we give the following definition that is used in Problem 5.3.

Given some MI-relaxation solution $X/n$ where $X/n - 1 \in \text{conv}(P_{n-1})$, let $m$ be the number of all the pedigrees corresponding to $X/n - 1$. Let $P$ be a 0-1 matrix with rows corresponding to the pedigrees in $N_{n-2}$, and columns corresponding to the corresponding $x_{ijk}$ variables. An example of such a $P$ matrix is given in Example 5.3.2.

Example 5.3.2. Consider the list of pedigrees given in Example 5.3.1, the corresponding matrix $P$ is shown in Table 5.3.
5.3 Checking the Sufficient Condition for $X/k \in \text{conv}(P_k)$

<table>
<thead>
<tr>
<th>Pedigree</th>
<th>$x_{134}$</th>
<th>$x_{234}$</th>
<th>$x_{345}$</th>
<th>$x_{126}$</th>
<th>$x_{246}$</th>
<th>$x_{357}$</th>
<th>$x_{167}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>((1, 3), (3, 4), (1, 2), (3, 5))</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>((2, 3), (3, 4), (1, 2), (3, 5))</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>((2, 3), (3, 4), (2, 4), (3, 5))</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>((1, 3), (3, 4), (1, 2), (1, 6))</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>((2, 3), (3, 4), (1, 2), (1, 6))</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Let $m$ be the number of pedigrees in $P$, i.e. the number of rows of matrix $P$. We want to find a column vector $\lambda \in [0, 1]^m$ as a weight vector for expressing $X/n - 1$ as a convex combination of all the pedigrees in $P$, by solving Problem 5.3. Let $T$ be the set of sink nodes that correspond to $x_{ijn} > 0$ variables in $X/n$, and let $s \in S = \{1, \ldots, m\}$ correspond to pedigrees in $P$. Let $P(s)$ correspond to the pedigree in row $s$ of the matrix $P$. Let $f_{st}$ denote the flow from $s \in S$ to $t \in T$. Let $G(e)$ be the set of generators for an edge $e$. An LP model for finding a vector $\lambda \in [0, 1]^m$ for which the FAT$_{n-1}(\lambda)$ problem is feasible, is given as Problem 5.3.

**Problem 5.3**

$$\max \quad \sum_{(s,t) \in S \times T} f_{st}$$

subject to:

$$\sum_{s \in S} \lambda(s) = 1, \quad (5.7)$$

$$P^T \lambda - X/n - 1 = 0, \quad (5.8)$$

$$\sum_{s \in G(e_t) \in P(s)} f_{st} - x_n(e_t) = 0, \quad \forall t \in T, \quad (5.9)$$

$$\sum_{t \in G(e_t) \in P(s)} f_{st} - \lambda(s) = 0, \quad \forall s \in S, \quad (5.10)$$

$$\lambda(s) \geq 0, f_{st} \geq 0, \quad \forall s \in S, \forall t \in T. \quad (5.11)$$

Constraint (5.7) guarantees that for any $\lambda$ vector, the sum of $\lambda(s)$ coordinates is equal to one. Constraint (5.8) makes sure that the pedigree combination given by $\lambda$ satisfies the availability of $x_{ijk}$ variables in $X/n - 1$. Constraints (5.9) and (5.10) ensure that the supply and demand conditions are met. If this problem is feasible, we have a $\lambda$ and a corresponding feasible solution to the FAT$_{n-1}(\lambda)$ problem and vice versa.
5.4 Checking the Necessary Condition for \( X/k + 1 \in \text{conv}(P_{k+1}) \)

We illustrate the process for checking the necessary condition for \( X/k + 1 \in \text{conv}(P_{k+1}) \) using an MI-relaxation solution in the following example.

**Example 5.4.1.** Using Algorithm 7, we generate an MI-relaxation solution of size 9 given in Table 5.4.

We check the necessary condition by taking the steps given in Algorithm 7. Starting with \( k = 4 \), we check the necessary condition for \( X/5 \in \text{conv}(P_5) \). Increasing \( k \) by one at each stage we check the necessary condition for \( X/k + 1 \in \text{conv}(P_{k+1}) \). We also check the sufficient condition for \( X/k \in \text{conv}(P_k) \) before checking the necessary condition for \( X/k + 1 \in \text{conv}(P_{k+1}) \).

### Table 5.4: The \( x_{ijk} \) Values for the MI-relaxation Solution in Example 5.4.1

<table>
<thead>
<tr>
<th>( k )</th>
<th>( x_{124} )</th>
<th>( x_{134} )</th>
<th>( x_{235} )</th>
<th>( x_{356} )</th>
<th>( x_{456} )</th>
<th>( x_{145} )</th>
<th>( x_{146} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>0.5</td>
<td>0.5</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>6</td>
<td>0.5</td>
<td>0.5</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>7</td>
<td>0.5</td>
<td>0.5</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>8</td>
<td>0.5</td>
<td>0.5</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>9</td>
<td>0.5</td>
<td>0.5</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.5</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Although Problem 5.3 can be solved for checking the sufficient condition for the membership of \( X/k + 1 \) in the pedigree polytope, there may be exponentially many pedigrees to be considered. Therefore Problem 5.3 may not be solvable in time polynomial in \( n \). However, for the example given in Section 5.5, it is easily solved.

In the next section we illustrate the process of checking the necessary condition for \( X/k + 1 \in \text{conv}(P_{k+1}) \) given that \( X/k \in \text{conv}(P_k) \).
5.4 Checking the Necessary Condition for $X/k + 1 \in \text{conv}(P_{k+1})$

We apply the frozen flow finding algorithm by Gusfield (1988) to the $F_4$ solution in $N_4$ and find that none of the arcs have frozen flow. We can conclude that $N_4$ is well-defined, and that $X/5$ belongs to $\text{conv}(P_5)$. If $F_4$ was infeasible we would have concluded that $X/5 \notin \text{conv}(P_5)$, and therefore $X/n \notin \text{conv}(P_n)$. Next we walk through Algorithm 6 for checking the necessary condition for $X/k + 1 \in \text{conv}(P_{k+1})$ when $k > 4$.

5.4.2 Checking the Necessary Condition for $X/6 \in \text{conv}(P_6)$

Since we have $X/5 \in \text{conv}(P_5)$, we can construct the $N_5$ network and check whether $X/6$ belongs to $\text{conv}(P_6)$ or not. We check the necessary condition for $X/6 \in \text{conv}(P_6)$ by walking through the following steps.

1. The third layer including the nodes in $V_{[3]} = \{[6 : 2, 5], [6 : 3, 5], [6 : 4, 5]\}$ is added to $N_4$ to construct $N_5$. The set of links between the second and third layers is $\{([5 : 2, 3], [6 : 2, 5]), ([5 : 2, 3], [6 : 3, 5]), ([5 : 1, 4], [6 : 4, 5])\}$.  

2. Figure 5.9 shows the arcs in the restricted networks corresponding to the links $([5 : 2, 3], [6 : 2, 5]), ([5 : 2, 3], [6 : 3, 5]),$ and $([5 : 1, 4], [6 : 4, 5])$. The $C(L_i)$ values are shown in Figure 5.10 as capacities of the links in the network.

3. The $F_5$ problem is feasible. The optimal flows of $F_5$ are shown in Figure 5.10.

4. Using the frozen flow finding algorithm, all of the arcs in the last layer of the $N_5$ network are rigid and we set their capacities equal to their optimal flow in the $F_5$ problem.

5. A temporary layer of sink nodes is added to the $N_5$ network. Each sink in this layer corresponds to a link $L_i$ with demand for some commodity $i$ equal to $C(L_i)$. A source node in a temporary layer zero is also added to the network with one unit of supply available for each commodity.

6. The multicommodity flow problem in the $N_5$ network is solved and the optimal solution is equal to one. Figure 5.11 shows the optimal flows.

7. Since the optimal solution to the multicommodity flow problem is equal to one, we conclude that the necessary condition for membership is satisfied for $X/6 \in \text{conv}(P_6)$. 

Figure 5.8: The $N_4$ Network
Figure 5.9: Restricted Networks for ([5 : 2, 3], [6 : 2, 5]), ([5 : 2, 3], [6 : 3, 5]), and ([5 : 1, 4], [6 : 4, 5])

Figure 5.10: The Optimal Flow in $F_5$

Figure 5.11: The Multicommodity Network $N_5$
5.4 Checking the Necessary Condition for \( X/k + 1 \in conv(P_{k+1}) \)

To check the sufficient condition for \( X/6 \in conv(P_6) \), we need to solve the FAT\(_k\)(\( \lambda \)) problem for the pedigrees in \( N_4 \). Given \( X/6 \), the following six pedigrees are found using Algorithm 7:

\[
\begin{align*}
W_1 &= ((1, 2), (2, 3), (2, 5)) \\
W_2 &= ((1, 3), (2, 3), (2, 5)) \\
W_3 &= ((1, 2), (2, 3), (3, 5)) \\
W_4 &= ((1, 3), (2, 3), (3, 5)) \\
W_5 &= ((1, 2), (1, 4), (4, 5)) \\
W_6 &= ((1, 3), (1, 4), (4, 5))
\end{align*}
\]

Based on Theorem 5.1.3, since the multicommodity flow problem in \( N_5 \) has an optimal flow equal to one, we can conclude that \( X/6 \) is a member of the pedigree polytope \( conv(P_6) \).

5.4.3 Checking the Necessary Condition for \( X/k + 1 \in conv(P_{k+1}) \) for \( k = 6 \), and \( k = 7 \)

Given that \( X/6 \in conv(P_6) \) and the network \( N_5 \) is well defined, we can check the necessary condition for \( X/7 \in conv(P_7) \) by taking the following steps.

1. The fourth layer including the nodes in \( V[4] = \{[7 : 2, 6], [7 : 4, 6], [7 : 5, 6]\} \) is added to the network. The set of the links between the last two layers of the \( N_6 \) network is \( \{(6 : 2, 5), (6 : 2, 5), (6 : 2, 5), (6 : 3, 5), (7 : 5, 6), (6 : 4, 5), (7 : 4, 6), (6 : 4, 5), (7 : 5, 6)\} \).

2. The restricted network for these links is constructed as shown in Figure 5.12.

3. The \( F_6 \) problem is feasible. The optimal flows are shown in Figure 5.13.

4. Links \( (6 : 2, 5), (7 : 2, 6) \), \( (6 : 3, 5), (7 : 5, 6) \), and \( (6 : 4, 5), (7 : 4, 6) \) are rigid and links \( (6 : 2, 5), (7 : 5, 6) \), and \( (6 : 4, 5), (7 : 5, 6) \) are dummy and therefore they are discarded from the network.

5. Three sink nodes are added to the \( N_6 \) network for each of the remaining links. The multicommodity flow network for the \( N_6 \) network is formed as shown in Figure 5.14.

6. The multicommodity flow problem is solved in the \( N_6 \) network and the optimal solution is equal to 1. The flow of commodity 1 and 3 is 0.25 each and the flow of commodity 2 is 0.5. The commodity flow paths are shown in Figure 5.15.

7. We conclude that the necessary condition for \( X/7 \in conv(P_7) \) is therefore satisfied.
5.4 Checking the Necessary Condition for $X/k + 1 \in \text{conv}(P_{k+1})$

**Figure 5.12:** Restricted Networks for Links in $N_6$

**Figure 5.13:** The Optimal Flows in $F_6$

**Figure 5.14:** The $N_6$ Multicommodity Flow Network

**Figure 5.15:** Multicommodity Flows in $N_6$
5.4 Checking the Necessary Condition for \(X/k + 1 \in \text{conv}(P_{k+1})\)

To check the sufficient condition for \(X/7 \in \text{conv}(P_7)\), we first use Algorithm 8 to find the pedigrees in \(N_6\). The pedigrees are given below.

\[
\begin{align*}
W_1 &= ((1,2), (2,3), (2,5), (2,6)) \\
W_2 &= ((1,3), (2,3), (2,5), (2,6)) \\
W_3 &= ((1,2), (2,3), (3,5), (5,6)) \\
W_4 &= ((1,3), (2,3), (3,5), (5,6)) \\
W_5 &= ((1,2), (1,4), (4,5), (4,6)) \\
W_6 &= ((1,3), (1,4), (4,5), (4,6))
\end{align*}
\]

We solve the FAT\(_k(\lambda)\) problem for \(X/7\) to check the sufficient condition. We can see that \(X/7\) can be expressed in terms of a convex combination of these pedigrees e.g. using the weight vector \((0.25, 0.0, 0.25, 0.25, 0.25)\). Therefore the sufficient condition for \(X/7 \in \text{conv}(P_7)\) is satisfied. Continuing similarly with \(k = 7\), we can see that the necessary and the sufficient conditions for \(X/8 \in \text{conv}(P_8)\) are satisfied, since we can express \(X/8\) as a convex combination of the following three pedigrees using the weight vector \((0.25, 0.25, 0.5)\).

\[
\begin{align*}
W_1 &= ((1,3), (2,3), (2,5), (2,6), (3,4)) \\
W_2 &= ((1,3), (2,3), (3,5), (5,6), (3,4)) \\
W_3 &= ((1,2), (1,4), (4,5), (4,6), (2,3))
\end{align*}
\]

5.4.4 Checking the Necessary Condition for \(X/9 \in \text{conv}(P_9)\)

Given that \(X/8 \in \text{conv}(P_8)\) and the network \(N_7\) is well defined, we can check the necessary condition for \(X/9 \in \text{conv}(P_9)\).

1. The sixth layer including the nodes in \(V_6 = \{[9 : 3, 8], [9 : 4, 6]\}\) is added to the network and the set of links between the last two layers is \{([8 : 2, 3], [9 : 3, 8]), ([8 : 3, 4], [9 : 3, 8]), ([8 : 2, 3], [9 : 4, 6]), ([8 : 3, 4], [9 : 4, 6])\}.

2. The restricted network is shown in Figure 5.16

3. The \(F_8\) problem is solved and it is found to be infeasible as shown in Figure 5.17

4. We conclude that \(X/9\) is not a member of the pedigree polytope \(\text{conv}(P_9)\).
5.4 Checking the Necessary Condition for $X/k + 1 \in \text{conv}(P_{k+1})$

![Diagram](image)

**Figure 5.16:** The Restricted Network

5.4.5 Removing Dummy Arcs from the Network

It should be mentioned that identifying the rigid and dummy arcs in the network is necessary for applying Algorithm 6. For example, not discarding the dummy arc $([6:4,5],[7:5,6])$ from the network would have resulted in $C([8:2,3],[9:4,6]) = 0.25$, and $C([8:3,4],[9:4,6]) = 0.25$. The arcs in the restricted network corresponding to these two links when dummy arcs are not discarded are shown in Figure 5.18. This makes the $F_8$ problem feasible as shown in Figure 5.19. The multicommodity network is shown in Figure 5.20 and the multicommodity solution in the $N_8$ network is shown in Figure 5.21. This results in the multicommodity flow to be equal to one in the $N_8$ network as shown in Figure 5.21. The flow of commodities 1 to 4 is 0.5, 0.45, 0 and 0.05 respectively. Note that although arc $([6:4,5],[7:5,6])$ is not used in the multicommodity solution, it has affected the arc capacities in the last layer when considered in the restricted networks.

![Diagram](image)

**Figure 5.17:** The $F_8$ Problem

**Figure 5.18:** The Restricted Network with the Dummy Arcs Included
5.4 Checking the Necessary Condition for $X/k + 1 \in \text{conv}(P_{k+1})$

![Diagram](image)

**Figure 5.19:** The $F_8$ Problem When the Dummy Arcs Are Not Discarded

![Diagram](image)

**Figure 5.20:** Multicommodity Network for the $N_8$ with the Dummy Arcs Included

![Diagram](image)

**Figure 5.21:** The Multicommodity Network for the $N_8$ with the Dummy Arcs Included
Next we give a counterexample for which the necessary condition is not sufficient.

5.5 A Counterexample

We found an MI relaxation solution of size 10 which satisfies the necessary condition but not the sufficient condition for membership in the pedigree polytope. This instance is given in Example 5.5.1.

Example 5.5.1. Consider $X$, an MI-relaxation solution corresponding to a problem of size 10 given in Table 5.5.

We checked the sufficient conditions for $X/k \in \text{conv}(P_k)$, for $k = 4, \ldots, 8$, and found that the conditions are satisfied. Next we show that the sufficient condition for $X/9 \in \text{conv}(P_9)$ is also satisfied. The multicommodity flow problem in $N_8$ is solved and the optimal solution is equal to one. The flows of the commodities are shown in Figure 5.22. The sufficient condition for the membership of $X/9 \in \text{conv}(P_9)$ is satisfied, as all of the commodity flow paths in the $N_8$ network are pedigrees. These paths are given below.

\[
\begin{align*}
W_1 &= ((1, 2), (1, 3), (2, 4), (1, 5), (2, 3), (2, 6)) \\
W_2 &= ((1, 2), (1, 3), (2, 4), (4, 6), (1, 4), (2, 6)) \\
W_3 &= ((1, 2), (1, 3), (3, 5), (5, 6), (1, 4), (1, 8)) \\
W_4 &= ((1, 2), (1, 4), (4, 5), (2, 3), (4, 6), (6, 8))
\end{align*}
\]

To check the necessary condition for the membership of $X/10$ in $\text{conv}(P_{10})$, the maximum flow problems in the restricted networks of $N_9$ given by the links between layers 6 and 7 are solved. The $F_9$ problem is solved and the optimal solution is equal to one. The network for the $F_9$ problem is shown in Figure 5.23, and its optimal solution is shown in Table 5.6. None of the arcs in the network for the $F_9$ problem are found to be rigid. The $N_9$ network is shown in Figure 5.24. The multicommodity flow problem in the $N_9$ network is then solved and the optimal flow is found to be equal to one. The flow of the commodities in the $N_9$ are shown in Figure 5.25.
5.5 A Counterexample

**Figure 5.22:** Multicommodity Flow in $N_8$

**Figure 5.23:** The $F_9$ Problem in $N_9$

**Table 5.6:** The Optimal Solution to the $F_9$ Problem

<table>
<thead>
<tr>
<th>Edges $[k : r, s]$</th>
<th>Edges $[k + 1 : i, j]$</th>
<th>Flow</th>
<th>Capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>$[9: 2, 6]$</td>
<td>$[10: 3, 5]$</td>
<td>0</td>
<td>0.5</td>
</tr>
<tr>
<td>$[9: 2, 6]$</td>
<td>$[10: 4, 7]$</td>
<td>0.25</td>
<td>0.25</td>
</tr>
<tr>
<td>$[9: 2, 6]$</td>
<td>$[10: 6, 7]$</td>
<td>0.25</td>
<td>0.25</td>
</tr>
<tr>
<td>$[9: 1, 8]$</td>
<td>$[10: 3, 5]$</td>
<td>0.25</td>
<td>0.25</td>
</tr>
<tr>
<td>$[9: 1, 8]$</td>
<td>$[10: 4, 7]$</td>
<td>0</td>
<td>0.25</td>
</tr>
<tr>
<td>$[9: 1, 8]$</td>
<td>$[10: 6, 7]$</td>
<td>0</td>
<td>0.25</td>
</tr>
<tr>
<td>$[9: 6, 8]$</td>
<td>$[10: 3, 5]$</td>
<td>0.25</td>
<td>0.25</td>
</tr>
<tr>
<td>$[9: 6, 8]$</td>
<td>$[10: 6, 7]$</td>
<td>0</td>
<td>0.25</td>
</tr>
</tbody>
</table>
5.5 A Counterexample

Figure 5.24: The $N_9$ Network

Figure 5.25: Multicommodity Flow in the $N_9$ Network
Table 5.7: All the Possible Pedigrees for $X/9$

<table>
<thead>
<tr>
<th>No</th>
<th>$k = 4$</th>
<th>$k = 5$</th>
<th>$k = 6$</th>
<th>$k = 7$</th>
<th>$k = 8$</th>
<th>$k = 9$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$(i,j)$</td>
<td>$(i,j)$</td>
<td>$(i,j)$</td>
<td>$(i,j)$</td>
<td>$(i,j)$</td>
<td>$(i,j)$</td>
</tr>
<tr>
<td>1</td>
<td>(1,2),</td>
<td>(1,3),</td>
<td>(2,4),</td>
<td>(2,3),</td>
<td>(1,4),</td>
<td>(2,6)</td>
</tr>
<tr>
<td>2</td>
<td>(1,2),</td>
<td>(1,3),</td>
<td>(2,4),</td>
<td>(1,5),</td>
<td>(1,4),</td>
<td>(2,6)</td>
</tr>
<tr>
<td>3</td>
<td>(1,2),</td>
<td>(1,3),</td>
<td>(2,4),</td>
<td>(4,6),</td>
<td>(1,4),</td>
<td>(2,6)</td>
</tr>
<tr>
<td>4</td>
<td>(1,2),</td>
<td>(1,3),</td>
<td>(2,4),</td>
<td>(2,3),</td>
<td>(1,4),</td>
<td>(1,8)</td>
</tr>
<tr>
<td>5</td>
<td>(1,2),</td>
<td>(1,3),</td>
<td>(3,5),</td>
<td>(2,3),</td>
<td>(1,4),</td>
<td>(1,8)</td>
</tr>
<tr>
<td>6</td>
<td>(1,2),</td>
<td>(1,3),</td>
<td>(2,4),</td>
<td>(1,5),</td>
<td>(1,4),</td>
<td>(1,8)</td>
</tr>
<tr>
<td>7</td>
<td>(1,2),</td>
<td>(1,3),</td>
<td>(3,5),</td>
<td>(1,5),</td>
<td>(1,4),</td>
<td>(1,8)</td>
</tr>
<tr>
<td>8</td>
<td>(1,2),</td>
<td>(1,3),</td>
<td>(2,4),</td>
<td>(4,6),</td>
<td>(1,4),</td>
<td>(1,8)</td>
</tr>
<tr>
<td>9</td>
<td>(1,2),</td>
<td>(1,3),</td>
<td>(3,5),</td>
<td>(5,6),</td>
<td>(1,4),</td>
<td>(1,8)</td>
</tr>
<tr>
<td>10</td>
<td>(1,2),</td>
<td>(1,3),</td>
<td>(2,4),</td>
<td>(1,5),</td>
<td>(2,3),</td>
<td>(2,6)</td>
</tr>
<tr>
<td>11</td>
<td>(1,2),</td>
<td>(1,4),</td>
<td>(2,4),</td>
<td>(1,5),</td>
<td>(2,3),</td>
<td>(2,6)</td>
</tr>
<tr>
<td>12</td>
<td>(1,2),</td>
<td>(1,3),</td>
<td>(2,4),</td>
<td>(4,6),</td>
<td>(2,3),</td>
<td>(2,6)</td>
</tr>
<tr>
<td>13</td>
<td>(1,2),</td>
<td>(1,4),</td>
<td>(2,4),</td>
<td>(4,6),</td>
<td>(2,3),</td>
<td>(2,6)</td>
</tr>
<tr>
<td>14</td>
<td>(1,2),</td>
<td>(1,3),</td>
<td>(2,4),</td>
<td>(2,3),</td>
<td>(4,6),</td>
<td>(2,6)</td>
</tr>
<tr>
<td>15</td>
<td>(1,2),</td>
<td>(1,4),</td>
<td>(2,4),</td>
<td>(2,3),</td>
<td>(4,6),</td>
<td>(2,6)</td>
</tr>
<tr>
<td>16</td>
<td>(1,2),</td>
<td>(1,3),</td>
<td>(2,4),</td>
<td>(1,5),</td>
<td>(4,6),</td>
<td>(2,6)</td>
</tr>
<tr>
<td>17</td>
<td>(1,2),</td>
<td>(1,4),</td>
<td>(2,4),</td>
<td>(1,5),</td>
<td>(4,6),</td>
<td>(2,6)</td>
</tr>
<tr>
<td>18</td>
<td>(1,2),</td>
<td>(1,3),</td>
<td>(2,4),</td>
<td>(2,3),</td>
<td>(4,6),</td>
<td>(6,8)</td>
</tr>
<tr>
<td>19</td>
<td>(1,2),</td>
<td>(1,4),</td>
<td>(2,4),</td>
<td>(2,3),</td>
<td>(4,6),</td>
<td>(6,8)</td>
</tr>
<tr>
<td>20</td>
<td>(1,2),</td>
<td>(1,4),</td>
<td>(4,5),</td>
<td>(2,3),</td>
<td>(4,6),</td>
<td>(6,8)</td>
</tr>
<tr>
<td>21</td>
<td>(1,2),</td>
<td>(1,3),</td>
<td>(2,4),</td>
<td>(1,5),</td>
<td>(4,6),</td>
<td>(6,8)</td>
</tr>
<tr>
<td>22</td>
<td>(1,2),</td>
<td>(1,4),</td>
<td>(2,4),</td>
<td>(1,5),</td>
<td>(4,6),</td>
<td>(6,8)</td>
</tr>
<tr>
<td>23</td>
<td>(1,2),</td>
<td>(1,4),</td>
<td>(4,5),</td>
<td>(1,5),</td>
<td>(4,6),</td>
<td>(6,8)</td>
</tr>
<tr>
<td>24</td>
<td>(1,2),</td>
<td>(1,4),</td>
<td>(4,5),</td>
<td>(5,6),</td>
<td>(4,6),</td>
<td>(6,8)</td>
</tr>
</tbody>
</table>
5.6 Conclusion

To check the sufficient condition for the membership of $X/10$ in $\text{conv}(P_{10})$, some $\lambda$ vector which makes the $\text{FAT}_k(\lambda)$ problem feasible, needs to be found. Algorithm 5 is applied on $X/9$ to find all the corresponding pedigrees. These pedigrees are listed in Table 5.7. Because of the small number of pedigrees in $N_8$, we can use Problem 5.3 for checking the sufficient condition. Problem 5.3 is solved for $X/10$ using the $N_8$ network using Cplex 9.1, and Cplex reported the infeasibility of the problem. Since no $\lambda$ vector exists such that the $\text{FAT}_k(\lambda)$ problem has a feasible solution, and the set of all pedigrees in $N_8$ are considered, it is concluded that $X/10$ is not in $\text{conv}(P_{10})$. Thus we have a counterexample to show the necessary condition given in Section 5.1.4 is not sufficient for membership in the pedigree polytope.

5.6 Conclusion

In this chapter we illustrated Algorithm 6 for checking a necessary condition for membership of a given MI-relaxation solution $X/k + 1$ in $\text{conv}(P_{k+1})$, given that $X/k \in \text{conv}(P_k)$ for some $4 \leq k < n$. We showed the necessity of removing some of the dummy arcs from the layered network that is constructed for this purpose. We showed that this necessary condition is not sufficient by providing an example that satisfies the necessary condition but it is not in the pedigree polytope.
The LP Relaxation of Various TSP Models

In the following sections we compare the LP relaxation of some TSP formulations by solving various STSP and ATSP test instances provided in the TSPLIB (Reinelt, 1995), and some instances designed by Papadimitriou and Steiglitz (1978). The problems by Papadimitriou and Steiglitz are designed to show how common solution algorithms, e.g. 2-opt and 3-opt methods, fail to find the global optimum even for decent sized instances. We refer to these instances as the diamond instances, since they are made by connecting small structures called diamonds in a certain way. Unlike most of the TSPLIB instances that we solve in this chapter, diamond instances are non-Euclidean. We use Cplex9.1 to solve all these problem instances using the formulations given by Carr (1996), Claus (1984), Dantzig et al. (1954), Flood (1956), Fox et al. (1980), Miller et al. (1960), and Wong (1980). We compare these formulations with the MI formulation in terms of CPU seconds, number of Cplex iterations, and the gap with the optimal integer solution. We express the gap between an LP solution and the optimal integer solution by \( \frac{v_{LP}^* - v_{IP}^*}{v_{IP}^*} \), where \( v_{LP}^* \) and \( v_{IP}^* \) indicate the values of the optimal LP and IP solutions, respectively.

In Section 6.1 we explain the method we use for solving the problem instances using the DFJ formulation. In Section 6.2 we give the results of solving the TSPLIB instances by various TSP formulations. In Section 6.3 we give the structure of the diamond instances along with some lemmas from Papadimitriou and Steiglitz (1978). We also give some computational results on solving diamond instances by the TSP formulations mentioned above. We give some conclusions and suggestions for future research in Section 6.4.

6.1 The LP Relaxations of the DFJ Formulation

We compare the LP solution given by the relaxations of various TSP formulations with that of the DFJ formulation. Due to the exponential number of the constraints in the LP
relaxation of the DFJ formulation, the LP cannot be solved directly and therefore other relaxations of the DFJ formulation are used. The first and most straightforward relaxation of the DFJ formulation for the ATSP is the assignment problem (AP) with the TSP cost function. The AP relaxation of the DFJ formulation was first used by Eastman (1958) and Little et al. (1963). Bellmore and Malone (1971) used the 2-matching relaxation of the DFJ formulation for the STSP as a parallel formulation to the AP relaxation for the ATSP. The 2-matching relaxation of the DFJ formulation can be modeled as follows:

\[
\min \sum_{i} \sum_{j} c_{ij} x_{ij}
\]

subject to:

\[
\sum_{i<j} x_{ij} + \sum_{j<i} x_{ji} = 2, \quad \forall i, j \in V, \quad (6.1)
\]

\[
x_{ij} \leq 1, \quad \forall i, j \in V, \quad (6.2)
\]

\[
x_{ij} \geq 0, \quad \forall i, j \in V. \quad (6.3)
\]

In this thesis, following the approach taken by Dantzig, Fulkerson and Johnson to solve the LP relaxation of the DFJ formulation, we use the 2-matching relaxation. Given some solution \( x \) to the 2-matching relaxation corresponding to a problem instance, we find violated subtour elimination constraints using two heuristics (Algorithms 9 and 10) and add them to the relaxation formulation and solve the updated model. We repeat this process until no more violated constraints by \( x \) can be found.

We use Algorithm 9 to find all the possible subtours in \( x \). This algorithm receives some \( x \), a solution to a relaxation of the DFJ formulation, and returns a list of nodes corresponding to possible subtours in \( x \). Algorithm 9 runs within \( O(n^2) \) time. This algorithm checks all the edges in the form of \((i, j)\) corresponding to \( x_{ij} > 0 \) values in the solution to see if they belong to any subtours. The algorithm returns a set of all the existing subtours in \( x \). We add the constraint \( \sum_{i,j \in S} x_{ij} \leq |S| - 1 \) corresponding to each set \( S \) in \( L \) to the relaxation of the DFJ formulation and solve the problem again to get a new solution \( x \).

After removing the subtours from the solution, we apply Algorithm 10 to \( x \) to find possible violated constraints based on the shrinking heuristic by Crowder and Padberg (1980), and Padberg and Hong (1980). This heuristic is given as Algorithm 6.5 in (Applegate et al., 2006). The shrinking heuristic is commonly used for algorithms that look for cuts given the solution to an LP (Applegate et al., 2001). The shrinking heuristic finds some edge \((i, j)\) with the corresponding value of \( x_{ij} = 1 \), and replaces the edge with a new node. This process is called shrinking. The new values of the \( x \) variables are updated and the shrinking process continues until some edge \((i, j)\) with the corresponding \( x_{ij} \) variable larger
6.1 The LP Relaxations of the DFJ Formulation

Algorithm 9 An Algorithm for Detecting Subtours for Some Given $x$

$L \leftarrow \emptyset$
$V \leftarrow \{1, \ldots, n\}$

while $|V| > 0$ do

Select some $i \in V$

cycle $\leftarrow \{i\}$

$V \leftarrow V \setminus \{i\}$

while there exists a node $r \in V$ for which $x_{ir} = 1$, or $x_{ri} = 1$ do

add $r$ to cycle

$i \leftarrow r$

$V \leftarrow V \setminus \{r\}$

end while

if $|\text{cycle}| < n$ then

add cycle to $L$

end if

end while

return $L$

than one is found. Algorithm 10 receives some $x$ that is a solution to a relaxation of the DFJ formulation, and returns a set of nodes that violate subtour elimination constraints using a shrinking process.

Algorithm 10 An Algorithm for the Shrinking Heuristic ([Applegate et al., 2006])

$L \leftarrow \emptyset$

while an edge $e$ with $x_e \geq 1$ exists do

if $x_e > 1$ then

set $S \leftarrow$ the set of original vertices corresponding to $u$ and $v$

add $S$ to $L$

shrink $\{u,v\}$

end if

end while

return $L$

We add constraint $\sum_{i,j \in S} x_{ij} \leq |S| - 1$ corresponding to each set $S$ in $L$ to the current LP, and then solve the LP and update $x$. Algorithms 9 and 10 are applied repeatedly on the solution until no more violated constraints are detected. We give some computational results on some TSPLIB instances using these algorithms to find the solution to the LP relaxation of the DFJ formulation.
6.2 Computational Results on Solving TSPLIB Instances by Various TSP Formulations

We solved some TSPLIB instances using LP relaxations of various TSP formulations. We used Cplex9.1 for solving these problem instances. We compared these formulations in terms of their gap with the optimal integer solutions, CPU seconds, and the number of Cplex iterations. We report on the results of solving STSP instances in Section 6.2.1 and the ATSP instances in Section 6.2.2.

6.2.1 Solving STSP Instances in the TSPLIB by Various Formulations

The results of applying Algorithms 9 and 10 on some of the TSPLIB instances are shown in Table 6.1. The number of violated subtour elimination constraints found by Algorithm 9 and Algorithm 10 is shown in column $SEC_1$ and $SEC_2$, respectively. The value of the 2-matching relaxation and its gap are shown in columns $LP_1$ Value, and $LP_1$ Gap respectively. The solutions given by the MI formulation are given in the table as well as the CPU seconds for the MI and the DFJ formulations. The results for the MI-relaxation formulation are given separately in Table 6.2 in more detail. Table 6.2 includes the optimal solution of the instances and the optimal solution of the MI-relaxation. The solutions to the LP relaxation of the MI formulation are equal to the results of the LP relaxation of the DFJ formulation. Problems larger than 318 for the MI relaxation could not be solved as it exceeded Cplex memory.

The optimal solution of other TSP formulations and their gap with the optimal solution are given in Table 6.3. We did not solve instances larger than eil101 by the Carr and Claus formulations, and instances larger than pr76 by the Wong formulation as the Cplex memory overloaded. Table 6.4 shows the gap with the optimal solution for various formulations. The LP relaxations of the FGG, Flood, and MTZ formulations provide weaker solutions compared to other formulations. Also, Carr’s formulation provides weaker solutions than the MI-relaxation for the problems from rat99 to lin105. The number of required iterations by Cplex for solving the instances by each formulation is shown in Table 6.5 and Figure 6.1. In these figures the minimum, the first quartile, the median, the third quartile, and the maximum values of the number of iterations over all the instances are shown for each formulation. As shown in Figure 6.1, the number of Simplex iterations is significantly lower for the MI, MTZ, and Flood formulations.
### 6.2 Computational Results on Solving TSPLIB Instances by Various TSP Formulations

**Table 6.1: Results for the DFJ Formulation Compared with the MI-relaxation**

<table>
<thead>
<tr>
<th>Problem</th>
<th>Optimal Value</th>
<th>MI Value</th>
<th>LP1 Value</th>
<th>MI gap</th>
<th>LP1 gap</th>
<th>SEC1</th>
<th>SEC2</th>
<th>CPU Seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>gr17</td>
<td>2085.0</td>
<td>2085.0</td>
<td>1684.0</td>
<td>0.00%</td>
<td>19.23%</td>
<td>11</td>
<td>0</td>
<td>1.45</td>
</tr>
<tr>
<td>gr21</td>
<td>2707.0</td>
<td>2707.0</td>
<td>2707.0</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0</td>
<td>0</td>
<td>0.14</td>
</tr>
<tr>
<td>gr24</td>
<td>1272.0</td>
<td>1272.0</td>
<td>1224.5</td>
<td>0.00%</td>
<td>3.73%</td>
<td>2</td>
<td>2</td>
<td>0.73</td>
</tr>
<tr>
<td>fri26</td>
<td>937.0</td>
<td>937.0</td>
<td>880.0</td>
<td>0.00%</td>
<td>6.08%</td>
<td>14</td>
<td>2</td>
<td>2.16</td>
</tr>
<tr>
<td>bays29</td>
<td>1610.0</td>
<td>1608.0</td>
<td>1546.0</td>
<td>0.12%</td>
<td>3.98%</td>
<td>8</td>
<td>2</td>
<td>1.40</td>
</tr>
<tr>
<td>bays9</td>
<td>2020.0</td>
<td>2013.5</td>
<td>1944.0</td>
<td>0.32%</td>
<td>3.76%</td>
<td>3</td>
<td>1</td>
<td>0.63</td>
</tr>
<tr>
<td>danzig42</td>
<td>699.0</td>
<td>697.0</td>
<td>641.0</td>
<td>0.29%</td>
<td>8.30%</td>
<td>5</td>
<td>2</td>
<td>1.24</td>
</tr>
<tr>
<td>swiss42</td>
<td>1273.0</td>
<td>1272.0</td>
<td>1214.5</td>
<td>0.08%</td>
<td>4.60%</td>
<td>3</td>
<td>8</td>
<td>2.03</td>
</tr>
<tr>
<td>att48</td>
<td>10628.0</td>
<td>10604.0</td>
<td>10041.5</td>
<td>0.23%</td>
<td>5.52%</td>
<td>27</td>
<td>7</td>
<td>6.10</td>
</tr>
<tr>
<td>gr48</td>
<td>5046.0</td>
<td>4959.0</td>
<td>4769.0</td>
<td>1.72%</td>
<td>5.49%</td>
<td>5</td>
<td>4</td>
<td>1.80</td>
</tr>
<tr>
<td>hkc48</td>
<td>11461.0</td>
<td>11444.5</td>
<td>11197.0</td>
<td>0.14%</td>
<td>2.30%</td>
<td>14</td>
<td>5</td>
<td>3.30</td>
</tr>
<tr>
<td>eil51</td>
<td>426.0</td>
<td>422.5</td>
<td>416.5</td>
<td>0.82%</td>
<td>2.23%</td>
<td>4</td>
<td>0</td>
<td>0.74</td>
</tr>
<tr>
<td>berlin52</td>
<td>7542.0</td>
<td>7542.0</td>
<td>7163.0</td>
<td>0.00%</td>
<td>5.03%</td>
<td>6</td>
<td>0</td>
<td>1.05</td>
</tr>
<tr>
<td>brazil58</td>
<td>25395.0</td>
<td>25345.5</td>
<td>20896.0</td>
<td>0.19%</td>
<td>17.72%</td>
<td>9</td>
<td>12</td>
<td>4.73</td>
</tr>
<tr>
<td>st70</td>
<td>675.0</td>
<td>671.0</td>
<td>623.5</td>
<td>0.59%</td>
<td>7.63%</td>
<td>19</td>
<td>16</td>
<td>10.08</td>
</tr>
<tr>
<td>eil76</td>
<td>538.0</td>
<td>537.0</td>
<td>534.0</td>
<td>0.19%</td>
<td>0.74%</td>
<td>4</td>
<td>0</td>
<td>0.87</td>
</tr>
<tr>
<td>pr76</td>
<td>108159.0</td>
<td>105120.0</td>
<td>98994.5</td>
<td>2.81%</td>
<td>8.47%</td>
<td>5</td>
<td>5</td>
<td>3.16</td>
</tr>
<tr>
<td>rat99</td>
<td>1211.0</td>
<td>1206.0</td>
<td>1198.0</td>
<td>0.41%</td>
<td>1.07%</td>
<td>9</td>
<td>11</td>
<td>12.13</td>
</tr>
<tr>
<td>kroA100</td>
<td>21282.0</td>
<td>20936.5</td>
<td>19378.5</td>
<td>1.62%</td>
<td>8.94%</td>
<td>18</td>
<td>23</td>
<td>22.79</td>
</tr>
<tr>
<td>kroB100</td>
<td>22141.0</td>
<td>21834.0</td>
<td>20339.5</td>
<td>1.39%</td>
<td>8.14%</td>
<td>14</td>
<td>25</td>
<td>23.09</td>
</tr>
<tr>
<td>kroC100</td>
<td>20749.0</td>
<td>20472.5</td>
<td>19705.0</td>
<td>1.33%</td>
<td>5.03%</td>
<td>27</td>
<td>18</td>
<td>23.20</td>
</tr>
<tr>
<td>kroD100</td>
<td>21294.0</td>
<td>21141.5</td>
<td>19952.5</td>
<td>0.72%</td>
<td>6.30%</td>
<td>23</td>
<td>25</td>
<td>27.87</td>
</tr>
<tr>
<td>kroE100</td>
<td>22668.0</td>
<td>21799.5</td>
<td>20622.0</td>
<td>1.22%</td>
<td>6.55%</td>
<td>33</td>
<td>18</td>
<td>24.50</td>
</tr>
<tr>
<td>rd100</td>
<td>7904.0</td>
<td>7896.5</td>
<td>7334.0</td>
<td>0.09%</td>
<td>7.21%</td>
<td>28</td>
<td>27</td>
<td>31.29</td>
</tr>
<tr>
<td>eil101</td>
<td>629.0</td>
<td>627.5</td>
<td>619.0</td>
<td>0.24%</td>
<td>1.59%</td>
<td>11</td>
<td>13</td>
<td>14.45</td>
</tr>
<tr>
<td>lin105</td>
<td>14379.0</td>
<td>14370.5</td>
<td>13213.5</td>
<td>0.06%</td>
<td>8.11%</td>
<td>40</td>
<td>29</td>
<td>44.46</td>
</tr>
<tr>
<td>pr107</td>
<td>44303.0</td>
<td>44303.0</td>
<td>43831.0</td>
<td>0.00%</td>
<td>2.08%</td>
<td>168</td>
<td>35</td>
<td>173.08</td>
</tr>
<tr>
<td>pr124</td>
<td>59030.0</td>
<td>58067.5</td>
<td>50200.0</td>
<td>1.63%</td>
<td>14.96%</td>
<td>76</td>
<td>23</td>
<td>88.97</td>
</tr>
<tr>
<td>bierl27</td>
<td>118282.0</td>
<td>117431.0</td>
<td>112278.5</td>
<td>0.72%</td>
<td>5.08%</td>
<td>40</td>
<td>28</td>
<td>62.66</td>
</tr>
<tr>
<td>ch130</td>
<td>6110.0</td>
<td>6075.5</td>
<td>5597.0</td>
<td>0.56%</td>
<td>8.40%</td>
<td>50</td>
<td>35</td>
<td>85.77</td>
</tr>
<tr>
<td>pr144</td>
<td>58537.0</td>
<td>58189.3</td>
<td>32863.0</td>
<td>0.59%</td>
<td>43.86%</td>
<td>59</td>
<td>69</td>
<td>234.70</td>
</tr>
<tr>
<td>ch150</td>
<td>6528.0</td>
<td>6490.1</td>
<td>6295.0</td>
<td>0.58%</td>
<td>3.57%</td>
<td>17</td>
<td>32</td>
<td>87.24</td>
</tr>
<tr>
<td>kroA150</td>
<td>26524.0</td>
<td>26299.0</td>
<td>24848.8</td>
<td>0.85%</td>
<td>6.32%</td>
<td>30</td>
<td>47</td>
<td>127.02</td>
</tr>
<tr>
<td>kroB150</td>
<td>26130.0</td>
<td>25732.5</td>
<td>24698.0</td>
<td>1.52%</td>
<td>5.48%</td>
<td>53</td>
<td>30</td>
<td>116.62</td>
</tr>
<tr>
<td>u159</td>
<td>42080.0</td>
<td>41925.0</td>
<td>40685.0</td>
<td>0.37%</td>
<td>3.32%</td>
<td>21</td>
<td>29</td>
<td>85.87</td>
</tr>
<tr>
<td>srl75</td>
<td>21407.0</td>
<td>21374.5</td>
<td>21140.0</td>
<td>0.15%</td>
<td>1.25%</td>
<td>84</td>
<td>40</td>
<td>287.76</td>
</tr>
<tr>
<td>brgl80</td>
<td>1950.0</td>
<td>1950.0</td>
<td>1800.0</td>
<td>0.00%</td>
<td>7.69%</td>
<td>29</td>
<td>17</td>
<td>113.69</td>
</tr>
</tbody>
</table>
# 6.2 Computational Results on Solving TSPLIB Instances by Various TSP Formulations

Table 6.2: Results for the LP Relaxation of MI Formulation for Some TSPLIB Instances

<table>
<thead>
<tr>
<th>Problem</th>
<th>Size</th>
<th>Optimal MI Value</th>
<th>MI relaxation</th>
<th>MI Gap</th>
<th>Cplex Iterations</th>
<th>Solution Seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>gr17</td>
<td>17</td>
<td>2085</td>
<td>2085.0</td>
<td>0.00%</td>
<td>133</td>
<td>0.02</td>
</tr>
<tr>
<td>gr21</td>
<td>21</td>
<td>2707</td>
<td>2707.0</td>
<td>0.00%</td>
<td>298</td>
<td>0.02</td>
</tr>
<tr>
<td>gr24</td>
<td>24</td>
<td>1272</td>
<td>1272.0</td>
<td>0.00%</td>
<td>288</td>
<td>0.03</td>
</tr>
<tr>
<td>fri26</td>
<td>26</td>
<td>937</td>
<td>937.0</td>
<td>0.00%</td>
<td>224</td>
<td>0.02</td>
</tr>
<tr>
<td>bayg29</td>
<td>29</td>
<td>1610</td>
<td>1608.0</td>
<td>0.12%</td>
<td>375</td>
<td>0.03</td>
</tr>
<tr>
<td>bays29</td>
<td>29</td>
<td>2020</td>
<td>2013.5</td>
<td>0.32%</td>
<td>435</td>
<td>0.03</td>
</tr>
<tr>
<td>swiss42</td>
<td>42</td>
<td>1273</td>
<td>1272.0</td>
<td>0.08%</td>
<td>960</td>
<td>0.14</td>
</tr>
<tr>
<td>dantzig42</td>
<td>42</td>
<td>699</td>
<td>697.0</td>
<td>0.29%</td>
<td>935</td>
<td>0.16</td>
</tr>
<tr>
<td>att48</td>
<td>48</td>
<td>10628</td>
<td>10604.0</td>
<td>0.23%</td>
<td>2028</td>
<td>0.58</td>
</tr>
<tr>
<td>gr48</td>
<td>48</td>
<td>5046</td>
<td>4959.0</td>
<td>1.72%</td>
<td>2363</td>
<td>0.69</td>
</tr>
<tr>
<td>hk48</td>
<td>48</td>
<td>11461</td>
<td>11444.5</td>
<td>0.14%</td>
<td>1820</td>
<td>0.52</td>
</tr>
<tr>
<td>ell51</td>
<td>51</td>
<td>426</td>
<td>422.5</td>
<td>0.82%</td>
<td>2217</td>
<td>0.88</td>
</tr>
<tr>
<td>berlin52</td>
<td>52</td>
<td>7542</td>
<td>7542.0</td>
<td>0.00%</td>
<td>2794</td>
<td>0.97</td>
</tr>
<tr>
<td>brazil58</td>
<td>58</td>
<td>25395</td>
<td>25345.5</td>
<td>0.19%</td>
<td>3596</td>
<td>1.42</td>
</tr>
<tr>
<td>st70</td>
<td>70</td>
<td>675</td>
<td>671.0</td>
<td>0.59%</td>
<td>4583</td>
<td>3.39</td>
</tr>
<tr>
<td>ell76</td>
<td>76</td>
<td>538</td>
<td>537.0</td>
<td>0.19%</td>
<td>7983</td>
<td>9.92</td>
</tr>
<tr>
<td>pr76</td>
<td>77</td>
<td>108159</td>
<td>108120.0</td>
<td>0.21%</td>
<td>3380</td>
<td>2.80</td>
</tr>
<tr>
<td>rat99</td>
<td>99</td>
<td>1211</td>
<td>1206.0</td>
<td>0.41%</td>
<td>7994</td>
<td>14.25</td>
</tr>
<tr>
<td>kroA100</td>
<td>100</td>
<td>21282</td>
<td>20936.5</td>
<td>1.62%</td>
<td>11708</td>
<td>29.89</td>
</tr>
<tr>
<td>kroB100</td>
<td>100</td>
<td>22141</td>
<td>21834.0</td>
<td>1.39%</td>
<td>10732</td>
<td>25.25</td>
</tr>
<tr>
<td>kroC100</td>
<td>100</td>
<td>20749</td>
<td>20472.5</td>
<td>1.33%</td>
<td>9386</td>
<td>21.88</td>
</tr>
<tr>
<td>kroD100</td>
<td>100</td>
<td>21294</td>
<td>21141.5</td>
<td>0.72%</td>
<td>11058</td>
<td>27.78</td>
</tr>
<tr>
<td>kroE100</td>
<td>100</td>
<td>22068</td>
<td>21799.5</td>
<td>1.22%</td>
<td>12338</td>
<td>33.61</td>
</tr>
<tr>
<td>rd100</td>
<td>100</td>
<td>7904</td>
<td>7896.5</td>
<td>0.09%</td>
<td>12603</td>
<td>41.14</td>
</tr>
<tr>
<td>ell101</td>
<td>101</td>
<td>629</td>
<td>627.5</td>
<td>0.24%</td>
<td>17586</td>
<td>65.25</td>
</tr>
<tr>
<td>lin105</td>
<td>105</td>
<td>14379</td>
<td>14371.0</td>
<td>0.06%</td>
<td>7358</td>
<td>29.69</td>
</tr>
<tr>
<td>pr107</td>
<td>107</td>
<td>44303</td>
<td>44303.0</td>
<td>0.00%</td>
<td>5988</td>
<td>10.50</td>
</tr>
<tr>
<td>gr120</td>
<td>120</td>
<td>6942</td>
<td>6911.3</td>
<td>0.44%</td>
<td>27553</td>
<td>151.81</td>
</tr>
<tr>
<td>pr124</td>
<td>124</td>
<td>59030</td>
<td>58067.5</td>
<td>1.63%</td>
<td>9792</td>
<td>47.34</td>
</tr>
<tr>
<td>bier127</td>
<td>127</td>
<td>118282</td>
<td>117431.0</td>
<td>0.72%</td>
<td>22753</td>
<td>146.80</td>
</tr>
<tr>
<td>ch130</td>
<td>130</td>
<td>6110</td>
<td>6075.5</td>
<td>0.56%</td>
<td>24542</td>
<td>143.49</td>
</tr>
<tr>
<td>pr136</td>
<td>136</td>
<td>96772</td>
<td>95934.5</td>
<td>0.87%</td>
<td>31818</td>
<td>216.08</td>
</tr>
<tr>
<td>pr144</td>
<td>144</td>
<td>58537</td>
<td>58189.3</td>
<td>0.59%</td>
<td>19418</td>
<td>192.84</td>
</tr>
<tr>
<td>ch150</td>
<td>150</td>
<td>6528</td>
<td>6490.1</td>
<td>0.58%</td>
<td>43906</td>
<td>477.22</td>
</tr>
<tr>
<td>kroA150</td>
<td>150</td>
<td>26324</td>
<td>26299.0</td>
<td>0.85%</td>
<td>49837</td>
<td>733.44</td>
</tr>
<tr>
<td>kroB150</td>
<td>150</td>
<td>26130</td>
<td>25732.5</td>
<td>1.52%</td>
<td>35071</td>
<td>358.89</td>
</tr>
<tr>
<td>pr152</td>
<td>152</td>
<td>73682</td>
<td>73208.5</td>
<td>0.64%</td>
<td>24364</td>
<td>224.98</td>
</tr>
<tr>
<td>u159</td>
<td>159</td>
<td>42080</td>
<td>41925.0</td>
<td>0.37%</td>
<td>80621</td>
<td>1183.81</td>
</tr>
<tr>
<td>si175</td>
<td>175</td>
<td>21407</td>
<td>21374.5</td>
<td>0.15%</td>
<td>47533</td>
<td>799.61</td>
</tr>
<tr>
<td>brg180</td>
<td>180</td>
<td>1950</td>
<td>1950.0</td>
<td>0.00%</td>
<td>135584</td>
<td>2671.59</td>
</tr>
<tr>
<td>rat195</td>
<td>195</td>
<td>2323</td>
<td>2299.3</td>
<td>1.02%</td>
<td>52534</td>
<td>706.19</td>
</tr>
<tr>
<td>d198</td>
<td>198</td>
<td>15780</td>
<td>15722.0</td>
<td>0.37%</td>
<td>69002</td>
<td>958.75</td>
</tr>
<tr>
<td>kroA200</td>
<td>200</td>
<td>29368</td>
<td>29065.0</td>
<td>1.03%</td>
<td>93262</td>
<td>3154.19</td>
</tr>
<tr>
<td>kroB200</td>
<td>200</td>
<td>29437</td>
<td>28865.0</td>
<td>1.94%</td>
<td>99720</td>
<td>3043.31</td>
</tr>
</tbody>
</table>

Continued on next page
6.2 Computational Results on Solving TSPLIB Instances by Various TSP Formulations

As shown in Figure 6.1, solving instances by the MTZ formulation requires fewer number of Simplex iterations compared to the MI and Flood formulations, however, the LP gap given by the MTZ and Flood formulations are significantly larger than the MI formulation. The CPU seconds used by Cplex for the formulations are shown in Figure 6.2. As shown in Figure 6.2, the CPU seconds are much lower for the LP relaxations of MI, MTZ, and Flood formulations.

### 6.2.2 Solving ATSP Instances in the TSPLIB by Various Formulations

We solved some TSPLIB instances for the ATSP ranging from size 17 to 171. The gap with the optimal solution for various ATSP formulations are given in Table 6.6. The Cplex solution seconds are given in Table 6.7 and Cplex iterations are given in Table 6.8. Cplex memory overloaded when solving problems kro124 and ftv170 using the Claus formulation,
### Table 6.3: LP Relaxation Values of Some STSP Formulations for Some TSPLIB Instances

<table>
<thead>
<tr>
<th>Problem</th>
<th>Size</th>
<th>Value</th>
<th>Carr</th>
<th>Flood</th>
<th>FG G</th>
<th>MTZ</th>
<th>Other†</th>
</tr>
</thead>
<tbody>
<tr>
<td>gr17</td>
<td>17</td>
<td>2085</td>
<td>2085.0</td>
<td>1652.0</td>
<td>618.8</td>
<td>1656.0</td>
<td>2085.0</td>
</tr>
<tr>
<td>gr21</td>
<td>21</td>
<td>2707</td>
<td>2707.0</td>
<td>2420.0</td>
<td>794.3</td>
<td>2448.7</td>
<td>2707.0</td>
</tr>
<tr>
<td>gr24</td>
<td>24</td>
<td>1272</td>
<td>1272.0</td>
<td>1052.0</td>
<td>618.0</td>
<td>1066.7</td>
<td>1272.0</td>
</tr>
<tr>
<td>fri26</td>
<td>26</td>
<td>937</td>
<td>937.0</td>
<td>833.0</td>
<td>350.0</td>
<td>835.5</td>
<td>937.0</td>
</tr>
<tr>
<td>bayg29</td>
<td>29</td>
<td>1610</td>
<td>1608.0</td>
<td>1440.0</td>
<td>804.4</td>
<td>1446.0</td>
<td>1608.0</td>
</tr>
<tr>
<td>bays29</td>
<td>29</td>
<td>2020</td>
<td>2013.5</td>
<td>1764.0</td>
<td>908.7</td>
<td>1775.1</td>
<td>2013.5</td>
</tr>
<tr>
<td>dantzig42</td>
<td>42</td>
<td>699</td>
<td>697.0</td>
<td>532.0</td>
<td>197.9</td>
<td>535.7</td>
<td>697.0</td>
</tr>
<tr>
<td>swiss42</td>
<td>42</td>
<td>1273</td>
<td>1272.0</td>
<td>1099.0</td>
<td>244.5</td>
<td>1018.4</td>
<td>1272.0</td>
</tr>
<tr>
<td>att48</td>
<td>48</td>
<td>10628</td>
<td>10604.0</td>
<td>8428.0</td>
<td>2787.8</td>
<td>8490.1</td>
<td>10604.0</td>
</tr>
<tr>
<td>gr48</td>
<td>48</td>
<td>5046</td>
<td>4959.0</td>
<td>4136.0</td>
<td>1326.4</td>
<td>4160.5</td>
<td>4959.0</td>
</tr>
<tr>
<td>hkg48</td>
<td>48</td>
<td>11461</td>
<td>11444.5</td>
<td>9870.0</td>
<td>4736.2</td>
<td>9914.9</td>
<td>11444.5</td>
</tr>
<tr>
<td>eil51</td>
<td>51</td>
<td>426</td>
<td>422.5</td>
<td>376.0</td>
<td>122.4</td>
<td>377.5</td>
<td>422.5</td>
</tr>
<tr>
<td>berlin52</td>
<td>52</td>
<td>7542</td>
<td>7542.0</td>
<td>6287.0</td>
<td>1153.4</td>
<td>6319.7</td>
<td>7542.0</td>
</tr>
<tr>
<td>brazili58</td>
<td>58</td>
<td>25395</td>
<td>25354.5</td>
<td>16565.0</td>
<td>6011.7</td>
<td>16707.5</td>
<td>25345.5</td>
</tr>
<tr>
<td>st70</td>
<td>70</td>
<td>675</td>
<td>671.0</td>
<td>519.0</td>
<td>122.1</td>
<td>521.1</td>
<td>671.0</td>
</tr>
<tr>
<td>eil76</td>
<td>76</td>
<td>538</td>
<td>537.0</td>
<td>484.0</td>
<td>175.0</td>
<td>485.2</td>
<td>537.0</td>
</tr>
<tr>
<td>pr76</td>
<td>76</td>
<td>108159</td>
<td>105120.0</td>
<td>77119.0</td>
<td>30931.2</td>
<td>77663.4</td>
<td>105120.0</td>
</tr>
<tr>
<td>rat99</td>
<td>99</td>
<td>1211</td>
<td>941.5</td>
<td>1089.0</td>
<td>487.3</td>
<td>1090.9</td>
<td>1206.0</td>
</tr>
<tr>
<td>kroA100</td>
<td>100</td>
<td>21282</td>
<td>18820.5</td>
<td>17087.0</td>
<td>2630.6</td>
<td>17128.6</td>
<td>20936.5</td>
</tr>
<tr>
<td>kroB100</td>
<td>100</td>
<td>22141</td>
<td>20114.8</td>
<td>16791.0</td>
<td>3971.3</td>
<td>16856.2</td>
<td>21834.0</td>
</tr>
<tr>
<td>kroC100</td>
<td>100</td>
<td>20749</td>
<td>18295.0</td>
<td>16738.0</td>
<td>3298.0</td>
<td>16782.4</td>
<td>20472.5</td>
</tr>
<tr>
<td>kroD100</td>
<td>100</td>
<td>21294</td>
<td>19246.8</td>
<td>16540.0</td>
<td>2794.3</td>
<td>16590.5</td>
<td>21141.5</td>
</tr>
<tr>
<td>kroE100</td>
<td>100</td>
<td>22068</td>
<td>19152.5</td>
<td>16685.0</td>
<td>3781.3</td>
<td>16753.1</td>
<td>21799.5</td>
</tr>
<tr>
<td>rd100</td>
<td>100</td>
<td>7904</td>
<td>7038.0</td>
<td>6556.0</td>
<td>1049.4</td>
<td>6570.4</td>
<td>7896.5</td>
</tr>
<tr>
<td>eil101</td>
<td>101</td>
<td>629</td>
<td>570.0</td>
<td>571.0</td>
<td>125.6</td>
<td>571.9</td>
<td>627.5</td>
</tr>
<tr>
<td>lin105</td>
<td>105</td>
<td>14379</td>
<td>- 8956.0</td>
<td>4502.2</td>
<td>9032.3</td>
<td>14370.5</td>
<td></td>
</tr>
</tbody>
</table>

† The LP relaxations of the MI, Claus and Wong formulations
# 6.2 Computational Results on Solving TSPLIB Instances by Various TSP Formulations

Table 6.4: The Gap for Some STSP Instances in the TSPLIB by Various Formulations

<table>
<thead>
<tr>
<th>Problem</th>
<th>Size</th>
<th>Carr</th>
<th>FGG</th>
<th>Flood</th>
<th>MTZ</th>
<th>Other †</th>
</tr>
</thead>
<tbody>
<tr>
<td>gr17</td>
<td>17</td>
<td>0.00%</td>
<td>70.32%</td>
<td>20.77%</td>
<td>20.58%</td>
<td>0.00%</td>
</tr>
<tr>
<td>gr21</td>
<td>21</td>
<td>0.00%</td>
<td>70.66%</td>
<td>10.60%</td>
<td>9.54%</td>
<td>0.00%</td>
</tr>
<tr>
<td>gr24</td>
<td>24</td>
<td>0.00%</td>
<td>51.42%</td>
<td>17.30%</td>
<td>16.14%</td>
<td>0.00%</td>
</tr>
<tr>
<td>fri26</td>
<td>26</td>
<td>0.00%</td>
<td>62.65%</td>
<td>11.10%</td>
<td>10.83%</td>
<td>0.00%</td>
</tr>
<tr>
<td>bayg29</td>
<td>29</td>
<td>0.12%</td>
<td>50.04%</td>
<td>10.56%</td>
<td>10.19%</td>
<td>0.12%</td>
</tr>
<tr>
<td>bays29</td>
<td>29</td>
<td>0.32%</td>
<td>55.02%</td>
<td>12.67%</td>
<td>12.13%</td>
<td>0.32%</td>
</tr>
<tr>
<td>dantzig42</td>
<td>42</td>
<td>0.29%</td>
<td>71.68%</td>
<td>23.89%</td>
<td>23.37%</td>
<td>0.29%</td>
</tr>
<tr>
<td>swiss42</td>
<td>42</td>
<td>0.08%</td>
<td>80.79%</td>
<td>20.74%</td>
<td>20.00%</td>
<td>0.08%</td>
</tr>
<tr>
<td>att48</td>
<td>48</td>
<td>0.23%</td>
<td>73.77%</td>
<td>20.70%</td>
<td>20.12%</td>
<td>0.23%</td>
</tr>
<tr>
<td>gr48</td>
<td>48</td>
<td>1.72%</td>
<td>73.71%</td>
<td>18.03%</td>
<td>17.55%</td>
<td>1.72%</td>
</tr>
<tr>
<td>hk48</td>
<td>48</td>
<td>0.14%</td>
<td>58.68%</td>
<td>13.88%</td>
<td>13.49%</td>
<td>0.14%</td>
</tr>
<tr>
<td>eil51</td>
<td>51</td>
<td>0.82%</td>
<td>71.27%</td>
<td>11.74%</td>
<td>11.38%</td>
<td>0.82%</td>
</tr>
<tr>
<td>berlin52</td>
<td>52</td>
<td>0.00%</td>
<td>84.71%</td>
<td>16.64%</td>
<td>16.21%</td>
<td>0.00%</td>
</tr>
<tr>
<td>brazil58</td>
<td>58</td>
<td>0.16%</td>
<td>76.33%</td>
<td>34.77%</td>
<td>34.21%</td>
<td>0.19%</td>
</tr>
<tr>
<td>st70</td>
<td>70</td>
<td>0.59%</td>
<td>81.91%</td>
<td>23.11%</td>
<td>22.80%</td>
<td>0.59%</td>
</tr>
<tr>
<td>eil76</td>
<td>76</td>
<td>0.19%</td>
<td>67.48%</td>
<td>10.04%</td>
<td>9.81%</td>
<td>0.19%</td>
</tr>
<tr>
<td>pr76</td>
<td>76</td>
<td>2.81%</td>
<td>71.40%</td>
<td>28.70%</td>
<td>28.20%</td>
<td>2.81%</td>
</tr>
<tr>
<td>rat99</td>
<td>99</td>
<td>22.25%</td>
<td>59.76%</td>
<td>10.07%</td>
<td>9.92%</td>
<td>0.41%</td>
</tr>
<tr>
<td>kroA100</td>
<td>100</td>
<td>11.57%</td>
<td>87.64%</td>
<td>19.71%</td>
<td>19.32%</td>
<td>1.62%</td>
</tr>
<tr>
<td>kroB100</td>
<td>100</td>
<td>9.15%</td>
<td>82.06%</td>
<td>24.16%</td>
<td>23.87%</td>
<td>1.39%</td>
</tr>
<tr>
<td>kroC100</td>
<td>100</td>
<td>11.83%</td>
<td>84.11%</td>
<td>19.33%</td>
<td>19.12%</td>
<td>1.33%</td>
</tr>
<tr>
<td>kroD100</td>
<td>100</td>
<td>9.61%</td>
<td>86.88%</td>
<td>22.33%</td>
<td>22.09%</td>
<td>0.72%</td>
</tr>
<tr>
<td>kroE100</td>
<td>100</td>
<td>13.21%</td>
<td>82.87%</td>
<td>24.39%</td>
<td>24.08%</td>
<td>1.22%</td>
</tr>
<tr>
<td>rt100</td>
<td>100</td>
<td>10.96%</td>
<td>86.72%</td>
<td>17.05%</td>
<td>16.87%</td>
<td>0.09%</td>
</tr>
<tr>
<td>eil01</td>
<td>101</td>
<td>9.38%</td>
<td>80.03%</td>
<td>9.22%</td>
<td>9.08%</td>
<td>0.24%</td>
</tr>
<tr>
<td>lin105</td>
<td>105</td>
<td>-</td>
<td>68.69%</td>
<td>37.71%</td>
<td>37.18%</td>
<td>0.06%‡</td>
</tr>
</tbody>
</table>

† The LP relaxations of the MI, Claus and Wong formulations

‡ This instance is only solved by the MI formulation.
## 6.2 Computational Results on Solving TSPLIB Instances by Various TSP Formulations

**Table 6.5:** Number of Cplex Iterations for Solving Some STSP Instances in the TSPLIB by Various Formulations

<table>
<thead>
<tr>
<th>Problem Size</th>
<th>Problem</th>
<th>Carr</th>
<th>Claus</th>
<th>FGG</th>
<th>Flood</th>
<th>MI</th>
<th>MTZ</th>
<th>Wong</th>
</tr>
</thead>
<tbody>
<tr>
<td>gr17</td>
<td>17</td>
<td>1002</td>
<td>3151</td>
<td>28</td>
<td>521</td>
<td>137</td>
<td>38</td>
<td>6530</td>
</tr>
<tr>
<td>gr21</td>
<td>21</td>
<td>1816</td>
<td>5779</td>
<td>38</td>
<td>1184</td>
<td>191</td>
<td>65</td>
<td>13877</td>
</tr>
<tr>
<td>gr24</td>
<td>24</td>
<td>3411</td>
<td>11420</td>
<td>56</td>
<td>1686</td>
<td>292</td>
<td>76</td>
<td>25345</td>
</tr>
<tr>
<td>fri26</td>
<td>26</td>
<td>3878</td>
<td>13920</td>
<td>54</td>
<td>2534</td>
<td>205</td>
<td>71</td>
<td>32157</td>
</tr>
<tr>
<td>bavg29</td>
<td>29</td>
<td>7480</td>
<td>21216</td>
<td>65</td>
<td>2098</td>
<td>340</td>
<td>81</td>
<td>38411</td>
</tr>
<tr>
<td>bays29</td>
<td>29</td>
<td>6920</td>
<td>20942</td>
<td>56</td>
<td>2074</td>
<td>371</td>
<td>99</td>
<td>40862</td>
</tr>
<tr>
<td>dantz42</td>
<td>42</td>
<td>28731</td>
<td>72573</td>
<td>105</td>
<td>3488</td>
<td>696</td>
<td>119</td>
<td>142270</td>
</tr>
<tr>
<td>swiss42</td>
<td>42</td>
<td>32197</td>
<td>63261</td>
<td>105</td>
<td>3765</td>
<td>940</td>
<td>134</td>
<td>131001</td>
</tr>
<tr>
<td>att48</td>
<td>48</td>
<td>87420</td>
<td>128984</td>
<td>122</td>
<td>7264</td>
<td>1944</td>
<td>153</td>
<td>238766</td>
</tr>
<tr>
<td>gr48</td>
<td>48</td>
<td>76998</td>
<td>109642</td>
<td>116</td>
<td>8419</td>
<td>2178</td>
<td>137</td>
<td>206858</td>
</tr>
<tr>
<td>hk48</td>
<td>48</td>
<td>79305</td>
<td>109970</td>
<td>86</td>
<td>7294</td>
<td>1924</td>
<td>138</td>
<td>236627</td>
</tr>
<tr>
<td>ell51</td>
<td>51</td>
<td>85525</td>
<td>155510</td>
<td>137</td>
<td>5705</td>
<td>2312</td>
<td>171</td>
<td>360732</td>
</tr>
<tr>
<td>berlin52</td>
<td>52</td>
<td>94569</td>
<td>138487</td>
<td>123</td>
<td>9682</td>
<td>2469</td>
<td>172</td>
<td>261272</td>
</tr>
<tr>
<td>brazil58</td>
<td>58</td>
<td>218641</td>
<td>208237</td>
<td>126</td>
<td>9351</td>
<td>3312</td>
<td>168</td>
<td>402153</td>
</tr>
<tr>
<td>st70</td>
<td>70</td>
<td>345057</td>
<td>372134</td>
<td>195</td>
<td>13688</td>
<td>4754</td>
<td>202</td>
<td>874841</td>
</tr>
<tr>
<td>ell76</td>
<td>76</td>
<td>820626</td>
<td>653385</td>
<td>176</td>
<td>27880</td>
<td>7466</td>
<td>264</td>
<td>1247823</td>
</tr>
<tr>
<td>pr76</td>
<td>76</td>
<td>3586411</td>
<td>551012</td>
<td>126</td>
<td>11217</td>
<td>3380</td>
<td>297</td>
<td>934023</td>
</tr>
<tr>
<td>rat99</td>
<td>99</td>
<td>1658867</td>
<td>1341762</td>
<td>236</td>
<td>44987</td>
<td>7007</td>
<td>376</td>
<td>-</td>
</tr>
<tr>
<td>kroA100</td>
<td>100</td>
<td>769939</td>
<td>1334986</td>
<td>340</td>
<td>31407</td>
<td>11181</td>
<td>332</td>
<td>-</td>
</tr>
<tr>
<td>kroB100</td>
<td>100</td>
<td>646525</td>
<td>1488945</td>
<td>306</td>
<td>27658</td>
<td>10882</td>
<td>373</td>
<td>-</td>
</tr>
<tr>
<td>kroC100</td>
<td>100</td>
<td>456215</td>
<td>1406024</td>
<td>300</td>
<td>35115</td>
<td>9438</td>
<td>340</td>
<td>-</td>
</tr>
<tr>
<td>kroD100</td>
<td>100</td>
<td>580988</td>
<td>1390889</td>
<td>269</td>
<td>29320</td>
<td>11343</td>
<td>364</td>
<td>-</td>
</tr>
<tr>
<td>kroE100</td>
<td>100</td>
<td>504238</td>
<td>1437900</td>
<td>300</td>
<td>23221</td>
<td>11687</td>
<td>358</td>
<td>-</td>
</tr>
<tr>
<td>rd100</td>
<td>100</td>
<td>619436</td>
<td>1363581</td>
<td>288</td>
<td>37835</td>
<td>11739</td>
<td>351</td>
<td>-</td>
</tr>
<tr>
<td>ell101</td>
<td>101</td>
<td>425683</td>
<td>1846705</td>
<td>266</td>
<td>39685</td>
<td>16975</td>
<td>443</td>
<td>-</td>
</tr>
<tr>
<td>lin105</td>
<td>105</td>
<td>-</td>
<td>-</td>
<td>271</td>
<td>34025</td>
<td>7420</td>
<td>391</td>
<td>-</td>
</tr>
</tbody>
</table>
and also when solving problem ftv170 using the FGG formulation. Since the loading time of the formulations onto Cplex took a very long time, we did not solve instances larger than ftv55 using the MTZ formulation, instances larger than kro124 using the Flood formulation, or instances larger than ftv70 using the Carr formulation.

The gaps given by the MI and Claus formulations are equal and significantly smaller than the gaps given by the FGG, Flood and MTZ formulations. Öncan et al. (2009) have also compared the gap and CPU seconds of the Claus formulation with those of the FGG and MTZ formulations. They did not report on the number of Cplex iterations required by each formulation. In Table 6.8 we can see that the Cplex iterations for the MI formulation are significantly lower than the Claus formulation. The Flood, FGG, and MTZ formulations provide the lowest solution times and Cplex iterations; however, their LP relaxation values are inferior to that of the MI formulation. The solution times and iterations are shown in Figures 6.3 and 6.4.

Desrochers and Laporte (1991) and later Gouveia and Pires (1999) have suggested formulations for the ATSP based on the MTZ formulation, with improved LP relaxations compared to the original MTZ formulation. They lifted some of the circuit inequalities of the MTZ formulation into different facet defining inequalities that are not dominated by the subtour elimination constraints of the DFJ formulation. Using this method, Gouveia and Pires (1999) have suggested various improved MTZ formulations. They have reported computational results on solving some ATSP instances in the TSPLIB by these formulations. All these formulations yield results that are dominated by the MI formulation in
### Table 6.6: The Gap of Solving Some Instances in the TSPLIB by the LP Relaxation of ATSP Formulations

<table>
<thead>
<tr>
<th>Problem</th>
<th>Size</th>
<th>Carr</th>
<th>Claus</th>
<th>FGG</th>
<th>Flood</th>
<th>MI</th>
<th>MTZ</th>
</tr>
</thead>
<tbody>
<tr>
<td>br17</td>
<td>17</td>
<td>0.00%</td>
<td>0.00%</td>
<td>85.37%</td>
<td>100.00%</td>
<td>0.00%</td>
<td>94.23%</td>
</tr>
<tr>
<td>ftv33</td>
<td>34</td>
<td>11.43%</td>
<td>0.00%</td>
<td>75.49%</td>
<td>7.85%</td>
<td>0.00%</td>
<td>7.64%</td>
</tr>
<tr>
<td>ftv35</td>
<td>36</td>
<td>15.68%</td>
<td>1.06%</td>
<td>77.06%</td>
<td>6.25%</td>
<td>1.06%</td>
<td>6.12%</td>
</tr>
<tr>
<td>ftv38</td>
<td>39</td>
<td>17.88%</td>
<td>1.02%</td>
<td>76.95%</td>
<td>6.01%</td>
<td>1.02%</td>
<td>5.86%</td>
</tr>
<tr>
<td>p43</td>
<td>43</td>
<td>83.63%</td>
<td>0.16%</td>
<td>88.83%</td>
<td>97.37%</td>
<td>0.16%</td>
<td>97.34%</td>
</tr>
<tr>
<td>ftv44</td>
<td>45</td>
<td>21.17%</td>
<td>1.74%</td>
<td>72.23%</td>
<td>5.70%</td>
<td>1.74%</td>
<td>5.55%</td>
</tr>
<tr>
<td>ftv47</td>
<td>48</td>
<td>22.07%</td>
<td>1.54%</td>
<td>74.35%</td>
<td>6.98%</td>
<td>1.54%</td>
<td>6.76%</td>
</tr>
<tr>
<td>ry48p</td>
<td>48</td>
<td>7.47%</td>
<td>0.92%</td>
<td>74.17%</td>
<td>13.21%</td>
<td>0.92%</td>
<td>12.88%</td>
</tr>
<tr>
<td>ft53</td>
<td>53</td>
<td>47.54%</td>
<td>0.00%</td>
<td>72.99%</td>
<td>14.11%</td>
<td>0.00%</td>
<td>14.04%</td>
</tr>
<tr>
<td>ftv55</td>
<td>56</td>
<td>20.40%</td>
<td>1.49%</td>
<td>74.18%</td>
<td>10.76%</td>
<td>1.49%</td>
<td>-</td>
</tr>
<tr>
<td>ftv64</td>
<td>65</td>
<td>21.86%</td>
<td>1.71%</td>
<td>74.79%</td>
<td>6.42%</td>
<td>1.71%</td>
<td>-</td>
</tr>
<tr>
<td>ft70</td>
<td>70</td>
<td>17.19%</td>
<td>0.05%</td>
<td>37.47%</td>
<td>1.80%</td>
<td>0.05%</td>
<td>-</td>
</tr>
<tr>
<td>ftv70</td>
<td>71</td>
<td>-</td>
<td>2.10%</td>
<td>74.28%</td>
<td>9.44%</td>
<td>2.10%</td>
<td>-</td>
</tr>
<tr>
<td>kro124</td>
<td>100</td>
<td>-</td>
<td>72.87%</td>
<td>-</td>
<td>0.64%</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>ftv170</td>
<td>171</td>
<td>-</td>
<td>-</td>
<td>1.45%</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

### Table 6.7: Cplex Solution Seconds of Solving Some Instances in the TSPLIB by the LP Relaxation of ATSP Formulations

<table>
<thead>
<tr>
<th>Problem</th>
<th>Size</th>
<th>Carr</th>
<th>Claus</th>
<th>FGG</th>
<th>Flood</th>
<th>MI</th>
<th>MTZ</th>
</tr>
</thead>
<tbody>
<tr>
<td>br17</td>
<td>17</td>
<td>0.02</td>
<td>0.05</td>
<td>0.03</td>
<td>0.00</td>
<td>0.03</td>
<td>0.02</td>
</tr>
<tr>
<td>ftv33</td>
<td>34</td>
<td>0.58</td>
<td>0.63</td>
<td>0.03</td>
<td>0.16</td>
<td>0.05</td>
<td>0.03</td>
</tr>
<tr>
<td>ftv35</td>
<td>36</td>
<td>1.09</td>
<td>1.27</td>
<td>0.06</td>
<td>0.11</td>
<td>0.08</td>
<td>0.00</td>
</tr>
<tr>
<td>ftv38</td>
<td>39</td>
<td>1.52</td>
<td>0.84</td>
<td>0.08</td>
<td>0.27</td>
<td>0.05</td>
<td>0.03</td>
</tr>
<tr>
<td>p43</td>
<td>43</td>
<td>3.05</td>
<td>4.02</td>
<td>0.11</td>
<td>0.20</td>
<td>0.09</td>
<td>0.02</td>
</tr>
<tr>
<td>ftv44</td>
<td>45</td>
<td>2.02</td>
<td>1.73</td>
<td>0.17</td>
<td>0.19</td>
<td>0.06</td>
<td>0.00</td>
</tr>
<tr>
<td>ftv47</td>
<td>48</td>
<td>6.48</td>
<td>2.94</td>
<td>0.11</td>
<td>0.48</td>
<td>0.01</td>
<td>0.06</td>
</tr>
<tr>
<td>ry48p</td>
<td>48</td>
<td>15.64</td>
<td>3.23</td>
<td>0.17</td>
<td>0.33</td>
<td>0.05</td>
<td>0.05</td>
</tr>
<tr>
<td>ft53</td>
<td>53</td>
<td>12.47</td>
<td>5.47</td>
<td>0.11</td>
<td>0.77</td>
<td>0.11</td>
<td>0.05</td>
</tr>
<tr>
<td>ftv55</td>
<td>56</td>
<td>22.23</td>
<td>6.08</td>
<td>0.23</td>
<td>0.34</td>
<td>0.19</td>
<td>-</td>
</tr>
<tr>
<td>ftv64</td>
<td>65</td>
<td>51.33</td>
<td>13.45</td>
<td>0.41</td>
<td>1.17</td>
<td>0.17</td>
<td>-</td>
</tr>
<tr>
<td>ft70</td>
<td>70</td>
<td>90.34</td>
<td>19.75</td>
<td>0.17</td>
<td>2.23</td>
<td>0.03</td>
<td>-</td>
</tr>
<tr>
<td>ftv70</td>
<td>71</td>
<td>-</td>
<td>20.05</td>
<td>0.44</td>
<td>1.09</td>
<td>0.36</td>
<td>-</td>
</tr>
<tr>
<td>kro124</td>
<td>100</td>
<td>-</td>
<td>1.33</td>
<td>-</td>
<td>1.52</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ftv170</td>
<td>171</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>7.52</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
6.2 Computational Results on Solving TSPLIB Instances by Various TSP Formulations

<table>
<thead>
<tr>
<th>Problem Size</th>
<th>Carr</th>
<th>Claus</th>
<th>FGG</th>
<th>Flood</th>
<th>MI</th>
<th>MTZ</th>
</tr>
</thead>
<tbody>
<tr>
<td>br17 17</td>
<td>1637</td>
<td>3584</td>
<td>26</td>
<td>152</td>
<td>259</td>
<td>19</td>
</tr>
<tr>
<td>ftv33 34</td>
<td>116415</td>
<td>28097</td>
<td>66</td>
<td>3675</td>
<td>1668</td>
<td>123</td>
</tr>
<tr>
<td>ftv35 36</td>
<td>128834</td>
<td>35602</td>
<td>74</td>
<td>5770</td>
<td>1970</td>
<td>108</td>
</tr>
<tr>
<td>ftv38 39</td>
<td>148431</td>
<td>44602</td>
<td>77</td>
<td>6997</td>
<td>2494</td>
<td>133</td>
</tr>
<tr>
<td>p43 43</td>
<td>244530</td>
<td>148712</td>
<td>77</td>
<td>2202</td>
<td>2956</td>
<td>66</td>
</tr>
<tr>
<td>ftv44 45</td>
<td>206550</td>
<td>60540</td>
<td>87</td>
<td>7355</td>
<td>3669</td>
<td>155</td>
</tr>
<tr>
<td>ftv47 48</td>
<td>433138</td>
<td>102454</td>
<td>82</td>
<td>13085</td>
<td>3303</td>
<td>160</td>
</tr>
<tr>
<td>ry48p 48</td>
<td>862255</td>
<td>95119</td>
<td>145</td>
<td>7554</td>
<td>4839</td>
<td>154</td>
</tr>
<tr>
<td>ft.53 53</td>
<td>940570</td>
<td>126529</td>
<td>121</td>
<td>25096</td>
<td>4246</td>
<td>189</td>
</tr>
<tr>
<td>ftv55 56</td>
<td>1203697</td>
<td>144203</td>
<td>102</td>
<td>12634</td>
<td>5355</td>
<td>-</td>
</tr>
<tr>
<td>ftv64 65</td>
<td>2077895</td>
<td>223126</td>
<td>118</td>
<td>20940</td>
<td>7836</td>
<td>-</td>
</tr>
<tr>
<td>ftv70 70</td>
<td>3527638</td>
<td>292633</td>
<td>226</td>
<td>62298</td>
<td>14896</td>
<td>-</td>
</tr>
<tr>
<td>ftv70 71</td>
<td>71</td>
<td>-</td>
<td>298865</td>
<td>130</td>
<td>23402</td>
<td>10887</td>
</tr>
<tr>
<td>kro124 100</td>
<td>-</td>
<td>-</td>
<td>308</td>
<td>-</td>
<td>34728</td>
<td>-</td>
</tr>
<tr>
<td>ftv170 171</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>150507</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 6.8: Cplex Iterations of Solving Some Instances in the TSPLIB by the LP Relaxation of ATSP Formulations

Figure 6.3: Solution Seconds for Solving ATSP Instances in the TSPLIB
6.2 Computational Results on Solving TSPLIB Instances by Various TSP Formulations

![Graph showing Cplex iterations for TSP instances.](image)

**Figure 6.4:** Cplex Iterations for ATSP Instances

**Table 6.9:** Comparing the MI and MCF Formulation in Terms of Gaps with the Optimal Solution and CPU Solution Seconds

<table>
<thead>
<tr>
<th>Problem</th>
<th>The MI</th>
<th>The MCF</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Gap</td>
<td>Time</td>
</tr>
<tr>
<td>ftv33</td>
<td>0.00%</td>
<td>0.05</td>
</tr>
<tr>
<td>ftv35</td>
<td>1.06%</td>
<td>0.08</td>
</tr>
<tr>
<td>ftv38</td>
<td>1.02%</td>
<td>0.05</td>
</tr>
<tr>
<td>ftv44</td>
<td>1.74%</td>
<td>0.06</td>
</tr>
<tr>
<td>ftv47</td>
<td>1.54%</td>
<td>0.01</td>
</tr>
<tr>
<td>ftv55</td>
<td>1.49%</td>
<td>0.19</td>
</tr>
<tr>
<td>ftv64</td>
<td>1.71%</td>
<td>0.17</td>
</tr>
<tr>
<td>ftv70</td>
<td>2.10%</td>
<td>0.36</td>
</tr>
</tbody>
</table>

terms of LP relaxation value and solution time, except a formulation called the multicommodity flow (MCF) formulation. In Table 6.9, we compare the computational results by Gouveia and Pires (1999) for solving some TSPLIB instances by the MCF formulation, with those of the MI formulation.

The gaps with the optimal solution of the MCF formulation for some of the instances solved by (Gouveia and Pires 1999) are smaller than those of the MI formulation, whereas the CPU times required to solve these instances by the MCF formulation are significantly greater than those of the MI formulation.
6.3 The TSP Instances by Papadimitriou and Steiglitz

The following definitions and lemmas are from [Papadimitriou and Steiglitz, 1978] and [Papadimitriou and Steiglitz, 1977].

Papadimitriou and Steiglitz designed a series of STSP instances to show how some local search algorithms, in particular the $k$-change heuristics, get stuck in local optima and fail to find the global optimum. All of these instances have a single global optimum and also exponentially many second best local optima that are all significantly inferior to the global optimum. Papadimitriou and Steiglitz applied some of the best known local search algorithms to the instances and showed how these algorithms fail to find the optimal solution. We refer to these instances as diamond instances. The instances are made up of some structures called diamonds. An STSP diamond has 8 vertices, and an ATSP diamond has 6 vertices. An instance can be formed by connecting $k \geq 2$ diamonds to each other in a specific way. In Sections 6.3.1 and 6.3.3 we give the structures for STSP and ATSP diamonds, respectively.

6.3.1 The Structure of the Diamond Instances for the STSP

To make a graph of size $8k$, $k$ diamonds are connected to each other using edges of cost $2M$, $M$, one or zero. Let an instance of $k$ diamonds be called a $k$-diamond. The vertices of some diamond $i$, for $1 \leq i \leq k$, in a $k$-diamond instance are labeled as $N_i, E_i, S_i, W_i, X_i, Y_i, U_i$ and $V_i$. A diamond can be traversed through two different paths (see Figure 6.5). These two paths are known as the north-south path $(N, U, W, Y, X, E, V, S)$, and the east-west path $(E, V, S, Y, X, N, U, W)$. All the edges on the east-west path have cost one and edges $(X, E)$ and $(W, Y)$ have cost zero.

**Lemma 6.3.1.** If a diamond is a subgraph of some graph $G$, with a Hamiltonian circuit $C$, then $G$ traverses the diamond in exactly one of the two paths of north-south or east-west.

Cost of the edges in a diamond instance are given as follows.

1. Each edge on the east-west cycle $(E_1, V_1, S_1, ..., W_1, E_2, ..., W_{i-1}, E_i, ..., W_l, E_l)$ is assigned cost one.

2. The vertices $\{N_2, ..., N_k, S_1, ..., S_k\}$ are connected with edges of cost zero, except edges $(N_i, S_i)$, for $i = 2, ..., k$, that are connected with edges of cost $M$.

3. $N_1$ is connected to every node in $\{N_2, ..., N_k, S_1, ..., S_k\}$ with an edge of cost $M$.

4. Any other edge in the graph is assigned a cost of $2M$. 
Figure 6.5: The North-south (Left) and East-west (Right) Cycles

A 2–diamond graph is shown in Figure 6.6. The edges with costs zero, one and $M$ are shown in this Figure. The cost of any other edge that is not shown in the Figure is $2M$. The only Hamiltonian cycle of cost $8k = 16$ for an instance with $k = 2$ is shown in Figure 6.7.

Lemma 6.3.2. The Papadimitriou and Steiglitz (1978) instances have one optimal solution of cost $8k$ given by the east-west cycle, and there are $2^{k-1}(k-1)!$ second-best solutions with a cost of $(M + 5k)$ that differ from the optimal solution in exactly $3k$ edges.

The global optimum in a diamond instance is the only Hamiltonian cycle in the graph with all its edges having cost one, shown in Figure 6.7. The cost of the optimum solution for an instance with $k$ diamonds is $8k$. In an instance of $k$ diamonds, as many as $2^{k-1}(k-1)!$ second best solutions have cost $(M + 5k)$, where $M$ is an arbitrarily large number. To have the second best solution be worse than the optimal solution, the value of parameter $M$ should be greater than $3k$. When solving these instances with a heuristic algorithm, the many edges of cost zero in this graph make it attractive for the algorithm to select them rather than edges with cost one. This forces the algorithms to select edges with costs of $M$ and $2M$ on the north-south cycle as well. The second-best cycles have one edge with cost $M$, $(3k - 1)$ edges with cost zero, and $5k$ edges with cost one. They differ from the optimal solution in only $3k$ edges, but having $(3k - 1)$ edges with cost zero makes the global optimum neighbourhood less attractive for the algorithm to be explored. So unless
6.3 The TSP Instances by Papadimitriou and Steiglitz

Figure 6.6: A 2–diamond Instance

Figure 6.7: The Only Hamiltonian Cycle of a 2-diamond Instance with Cost 16
choosing a $3k$-change (or higher) neighbourhood rule, the local search algorithms cannot find the global optimum. Papadimitriou and Steiglitz reported results of applying three different local search heuristics on diamond instances of size 24 to 40 ($k = 3, 4$, and 5), and reported that the global optimum is not found in any of these cases.

Although the optimal solutions to diamond instances cannot be found by such local search algorithms, Padberg and Sung (1988) have shown that these instances can be optimized in polynomial time using 2-matching relaxation of the DFJ formulation appended with $k$ subtour elimination constraints.

In Section 6.3.2, we give some computational results on solving STSP diamond instances by various formulations.

### 6.3.2 Solving Diamond Instances by Various STSP Formulations

Some STSP diamond instances, with $2 \leq k \leq 25$, are solved by the MI-relaxation. The value of $M$ is set equal to 1000. The optimal solution is found in all the cases by the LP relaxation of the MI formulation. The average CPU solution seconds by Cplex for 100 trials of solving these instance is reported in Table 6.10 and the CPU solution times are shown in Figure 6.8.

We solved diamond instances from size 16 to 104 using the LP relaxations of various TSP formulations. We did not solve instances larger than 11-diamonds by the Wong formulation, or instances larger than 12-diamonds by the Claus formulation, or instances larger than 13-diamonds by the Carr formulation as it would exceed the Cplex memory. The results are shown in Table 6.11. Figure 6.9 shows the CPU solution seconds taken by Cplex. The MI formulation seems to outperform other formulations in terms of solution seconds and number of iterations. Although the solution times are slightly less for the
### 6.3 The TSP Instances by Papadimitriou and Steiglitz

Table 6.10: Results for Solving Diamond Instances with $2 \leq k \leq 14$, by the LP Relaxation of the MI Formulation

<table>
<thead>
<tr>
<th>$k$</th>
<th>Size</th>
<th>Optimal Value</th>
<th>CPU Seconds</th>
<th>Cplex Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>16</td>
<td>16</td>
<td>0.03</td>
<td>105</td>
</tr>
<tr>
<td>3</td>
<td>24</td>
<td>24</td>
<td>0.02</td>
<td>337</td>
</tr>
<tr>
<td>4</td>
<td>32</td>
<td>32</td>
<td>0.05</td>
<td>700</td>
</tr>
<tr>
<td>5</td>
<td>40</td>
<td>40</td>
<td>0.03</td>
<td>1088</td>
</tr>
<tr>
<td>6</td>
<td>48</td>
<td>48</td>
<td>0.05</td>
<td>2820</td>
</tr>
<tr>
<td>7</td>
<td>56</td>
<td>56</td>
<td>0.02</td>
<td>3403</td>
</tr>
<tr>
<td>8</td>
<td>64</td>
<td>64</td>
<td>0.14</td>
<td>4553</td>
</tr>
<tr>
<td>9</td>
<td>72</td>
<td>72</td>
<td>0.09</td>
<td>6699</td>
</tr>
<tr>
<td>10</td>
<td>80</td>
<td>80</td>
<td>0.19</td>
<td>9519</td>
</tr>
<tr>
<td>11</td>
<td>88</td>
<td>88</td>
<td>0.23</td>
<td>11164</td>
</tr>
<tr>
<td>12</td>
<td>96</td>
<td>96</td>
<td>0.27</td>
<td>14714</td>
</tr>
<tr>
<td>13</td>
<td>104</td>
<td>104</td>
<td>0.41</td>
<td>19479</td>
</tr>
<tr>
<td>14</td>
<td>112</td>
<td>112</td>
<td>0.48</td>
<td>25390</td>
</tr>
<tr>
<td>15</td>
<td>120</td>
<td>120</td>
<td>0.67</td>
<td>27552</td>
</tr>
<tr>
<td>16</td>
<td>128</td>
<td>128</td>
<td>0.63</td>
<td>35455</td>
</tr>
<tr>
<td>17</td>
<td>136</td>
<td>136</td>
<td>1.05</td>
<td>40796</td>
</tr>
<tr>
<td>18</td>
<td>144</td>
<td>144</td>
<td>1.02</td>
<td>45452</td>
</tr>
<tr>
<td>19</td>
<td>152</td>
<td>152</td>
<td>1.27</td>
<td>60468</td>
</tr>
<tr>
<td>20</td>
<td>160</td>
<td>160</td>
<td>1.55</td>
<td>64043</td>
</tr>
<tr>
<td>21</td>
<td>168</td>
<td>168</td>
<td>2.00</td>
<td>71765</td>
</tr>
<tr>
<td>22</td>
<td>176</td>
<td>176</td>
<td>2.30</td>
<td>86806</td>
</tr>
<tr>
<td>23</td>
<td>184</td>
<td>184</td>
<td>3.69</td>
<td>116384</td>
</tr>
<tr>
<td>24</td>
<td>192</td>
<td>192</td>
<td>4.09</td>
<td>114678</td>
</tr>
<tr>
<td>25</td>
<td>200</td>
<td>200</td>
<td>4.14</td>
<td>117560</td>
</tr>
</tbody>
</table>


Table 6.11: Solution Values of Some Diamond Instances

<table>
<thead>
<tr>
<th>$k$</th>
<th>Size</th>
<th>FGG</th>
<th>Flood</th>
<th>MTZ</th>
<th>Other†</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>16</td>
<td>2.19</td>
<td>8.00</td>
<td>8.40</td>
<td>16.00</td>
</tr>
<tr>
<td>3</td>
<td>24</td>
<td>2.63</td>
<td>12.00</td>
<td>12.35</td>
<td>24.00</td>
</tr>
<tr>
<td>4</td>
<td>32</td>
<td>2.91</td>
<td>16.00</td>
<td>16.32</td>
<td>32.00</td>
</tr>
<tr>
<td>5</td>
<td>40</td>
<td>3.07</td>
<td>20.00</td>
<td>20.31</td>
<td>40.00</td>
</tr>
<tr>
<td>6</td>
<td>48</td>
<td>3.19</td>
<td>24.00</td>
<td>24.30</td>
<td>48.00</td>
</tr>
<tr>
<td>7</td>
<td>56</td>
<td>3.27</td>
<td>28.00</td>
<td>28.29</td>
<td>56.00</td>
</tr>
<tr>
<td>8</td>
<td>64</td>
<td>3.33</td>
<td>32.00</td>
<td>32.29</td>
<td>64.00</td>
</tr>
<tr>
<td>9</td>
<td>72</td>
<td>3.37</td>
<td>36.00</td>
<td>36.28</td>
<td>72.00</td>
</tr>
<tr>
<td>10</td>
<td>80</td>
<td>3.41</td>
<td>40.00</td>
<td>40.28</td>
<td>80.00</td>
</tr>
<tr>
<td>11</td>
<td>88</td>
<td>3.44</td>
<td>44.00</td>
<td>44.28</td>
<td>88.00</td>
</tr>
<tr>
<td>12</td>
<td>96</td>
<td>3.47</td>
<td>48.00</td>
<td>48.27</td>
<td>96.00</td>
</tr>
<tr>
<td>13</td>
<td>104</td>
<td>3.49</td>
<td>52.00</td>
<td>52.27</td>
<td>104.00</td>
</tr>
</tbody>
</table>

† Carr, Claus, MI, and Wong

Figure 6.9: Cplex Solution Seconds for Solving the Diamond Instances

MTZ formulation, the solution by the MI dominates that of the MTZ formulation. The number of Cplex iterations required to solve the problems are given in Table 6.13 and illustrated in Figure 6.10. The number of iterations for the MI formulation is significantly less than the six other formulations except for the MTZ and FGG formulations which give solutions with greater gap to the optimal solution.

Next we give the structure of the ATSP diamond instances.

6.3.3 The Structure of the Diamond Instances for the ATSP

Papadimitriou and Steiglitz (1978) have designed directed diamond instances for the ATSP as well as the STSP. These asymmetric diamond instances are made up of directed diamonds of 6 nodes that are connected to each other with arcs of cost zero, $M$, or $2M$, with
### Table 6.12: Cplex Solution Seconds for Solving the Diamond Instances

<table>
<thead>
<tr>
<th>k</th>
<th>Size</th>
<th>Carr</th>
<th>Claus</th>
<th>FGG</th>
<th>Flood</th>
<th>MI</th>
<th>MTZ</th>
<th>Wong</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>16</td>
<td>0.02</td>
<td>0.05</td>
<td>0.02</td>
<td>0.03</td>
<td>0.03</td>
<td>0.03</td>
<td>0.11</td>
</tr>
<tr>
<td>3</td>
<td>24</td>
<td>0.03</td>
<td>0.13</td>
<td>0.02</td>
<td>0.09</td>
<td>0.02</td>
<td>0.02</td>
<td>0.34</td>
</tr>
<tr>
<td>4</td>
<td>32</td>
<td>0.08</td>
<td>0.64</td>
<td>0.05</td>
<td>0.09</td>
<td>0.05</td>
<td>0.03</td>
<td>1.14</td>
</tr>
<tr>
<td>5</td>
<td>40</td>
<td>0.16</td>
<td>2.17</td>
<td>0.06</td>
<td>0.23</td>
<td>0.03</td>
<td>0.03</td>
<td>4.23</td>
</tr>
<tr>
<td>6</td>
<td>48</td>
<td>0.67</td>
<td>4.42</td>
<td>0.16</td>
<td>0.23</td>
<td>0.05</td>
<td>0.06</td>
<td>12.61</td>
</tr>
<tr>
<td>7</td>
<td>56</td>
<td>1.63</td>
<td>9.09</td>
<td>0.34</td>
<td>0.34</td>
<td>0.02</td>
<td>0.03</td>
<td>25.02</td>
</tr>
<tr>
<td>8</td>
<td>64</td>
<td>3.31</td>
<td>18.80</td>
<td>0.33</td>
<td>0.47</td>
<td>0.14</td>
<td>0.06</td>
<td>54.73</td>
</tr>
<tr>
<td>9</td>
<td>72</td>
<td>6.75</td>
<td>35.06</td>
<td>0.33</td>
<td>1.08</td>
<td>0.09</td>
<td>0.02</td>
<td>100.30</td>
</tr>
<tr>
<td>10</td>
<td>80</td>
<td>14.00</td>
<td>59.44</td>
<td>0.72</td>
<td>1.56</td>
<td>0.19</td>
<td>0.02</td>
<td>179.80</td>
</tr>
<tr>
<td>11</td>
<td>88</td>
<td>28.63</td>
<td>96.83</td>
<td>0.83</td>
<td>2.22</td>
<td>0.23</td>
<td>0.03</td>
<td>406.91</td>
</tr>
<tr>
<td>12</td>
<td>96</td>
<td>98.23</td>
<td>153.48</td>
<td>1.05</td>
<td>3.25</td>
<td>0.27</td>
<td>0.03</td>
<td>-</td>
</tr>
<tr>
<td>13</td>
<td>104</td>
<td>181.50</td>
<td>-</td>
<td>1.11</td>
<td>6.69</td>
<td>0.41</td>
<td>0.11</td>
<td>-</td>
</tr>
</tbody>
</table>

### Table 6.13: Number of Cplex Iterations for Solving the Diamond Instances

<table>
<thead>
<tr>
<th>k</th>
<th>Size</th>
<th>Carr</th>
<th>Claus</th>
<th>FGG</th>
<th>Flood</th>
<th>MI</th>
<th>MTZ</th>
<th>Wong</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>16</td>
<td>650</td>
<td>2752</td>
<td>20</td>
<td>281</td>
<td>105</td>
<td>28</td>
<td>5604</td>
</tr>
<tr>
<td>3</td>
<td>24</td>
<td>2719</td>
<td>12814</td>
<td>46</td>
<td>1328</td>
<td>337</td>
<td>40</td>
<td>25041</td>
</tr>
<tr>
<td>4</td>
<td>32</td>
<td>8531</td>
<td>33003</td>
<td>160</td>
<td>2907</td>
<td>700</td>
<td>67</td>
<td>59101</td>
</tr>
<tr>
<td>5</td>
<td>40</td>
<td>19476</td>
<td>74117</td>
<td>63</td>
<td>4851</td>
<td>1088</td>
<td>87</td>
<td>129887</td>
</tr>
<tr>
<td>6</td>
<td>48</td>
<td>36153</td>
<td>139494</td>
<td>75</td>
<td>8291</td>
<td>2820</td>
<td>112</td>
<td>306989</td>
</tr>
<tr>
<td>7</td>
<td>56</td>
<td>90297</td>
<td>231870</td>
<td>124</td>
<td>12693</td>
<td>3403</td>
<td>149</td>
<td>471068</td>
</tr>
<tr>
<td>8</td>
<td>64</td>
<td>242365</td>
<td>36883</td>
<td>101</td>
<td>17745</td>
<td>4553</td>
<td>151</td>
<td>805869</td>
</tr>
<tr>
<td>9</td>
<td>72</td>
<td>429831</td>
<td>577001</td>
<td>171</td>
<td>23166</td>
<td>6699</td>
<td>173</td>
<td>1147458</td>
</tr>
<tr>
<td>10</td>
<td>80</td>
<td>687136</td>
<td>746530</td>
<td>137</td>
<td>30347</td>
<td>9519</td>
<td>211</td>
<td>1746559</td>
</tr>
<tr>
<td>11</td>
<td>88</td>
<td>1187092</td>
<td>1049270</td>
<td>405</td>
<td>42101</td>
<td>11164</td>
<td>272</td>
<td>2263181</td>
</tr>
<tr>
<td>12</td>
<td>96</td>
<td>2731459</td>
<td>1415613</td>
<td>451</td>
<td>50720</td>
<td>14714</td>
<td>252</td>
<td>-</td>
</tr>
<tr>
<td>13</td>
<td>104</td>
<td>4516859</td>
<td>-</td>
<td>501</td>
<td>66407</td>
<td>19479</td>
<td>284</td>
<td>-</td>
</tr>
</tbody>
</table>
6.3 The TSP Instances by Papadimitriou and Steiglitz

![Figure 6.10: Number of Cplex Iterations for Solving Diamond Instances](image)

$$\begin{align*}
\text{Figure 6.11: A Directed Diamond}
\end{align*}$$

$M$ being an arbitrary large integer. A directed diamond is shown in Figure 6.11. An ATSP diamond instance is made up of connecting $k \geq 2$ directed diamond. The costs assigned to the arcs are given as follows.

- the arcs in the east-west circuit (Figure 6.12) with cost zero,
- the arcs in the form of $(E_i, W_{(i+1)\mod k})$, for all $i = 1, ..., k$, and in the form of $(N_j, S_i)$, for all $i = 1, ..., k$, and $j = 2, ..., k$, with cost zero,
- the arcs in the form of $(N_1, S_i)$ for $i = 1, ..., k$, with cost $M$, and
- any other arc not mentioned above, with cost $2M$.

The optimal solution to an ATSP $k$-diamond instance is given by the east-west circuit (Figure 6.12) and arc $(E_k, W_1)$, and its value is equal to zero.

**Lemma 6.3.3.** (Papadimitriou and Steiglitz [1978]) The directed diamond instances have an optimal tour with cost zero and $(k - 1)!$ second best tours with cost $M$. 
6.3 The TSP Instances by Papadimitriou and Steiglitz

Figure 6.12: The East-west Circuit Passing Through a Diamond

Table 6.14: Solution Seconds for Solving Directed Diamond with ATSP Formulations

<table>
<thead>
<tr>
<th>( k )</th>
<th>Size</th>
<th>Carr</th>
<th>Claus</th>
<th>FGG</th>
<th>Flood</th>
<th>MI</th>
<th>MTZ</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>12</td>
<td>0.03</td>
<td>0.05</td>
<td>0.00</td>
<td>0.02</td>
<td>0.01</td>
<td>0.03</td>
</tr>
<tr>
<td>3</td>
<td>18</td>
<td>0.02</td>
<td>0.02</td>
<td>0.02</td>
<td>0.05</td>
<td>0.02</td>
<td>0.03</td>
</tr>
<tr>
<td>4</td>
<td>24</td>
<td>0.03</td>
<td>0.16</td>
<td>0.05</td>
<td>0.02</td>
<td>0.02</td>
<td>0.00</td>
</tr>
<tr>
<td>5</td>
<td>30</td>
<td>0.09</td>
<td>0.44</td>
<td>0.03</td>
<td>0.09</td>
<td>0.03</td>
<td>0.03</td>
</tr>
<tr>
<td>6</td>
<td>36</td>
<td>0.16</td>
<td>1.00</td>
<td>0.16</td>
<td>0.16</td>
<td>0.03</td>
<td>0.02</td>
</tr>
<tr>
<td>7</td>
<td>42</td>
<td>0.55</td>
<td>3.16</td>
<td>0.08</td>
<td>0.09</td>
<td>0.05</td>
<td>0.02</td>
</tr>
<tr>
<td>8</td>
<td>48</td>
<td>1.17</td>
<td>5.09</td>
<td>0.31</td>
<td>0.23</td>
<td>0.06</td>
<td>0.05</td>
</tr>
<tr>
<td>9</td>
<td>54</td>
<td>1.30</td>
<td>9.89</td>
<td>0.33</td>
<td>0.39</td>
<td>0.09</td>
<td>0.03</td>
</tr>
<tr>
<td>10</td>
<td>60</td>
<td>2.66</td>
<td>16.19</td>
<td>0.39</td>
<td>0.73</td>
<td>0.10</td>
<td>0.05</td>
</tr>
<tr>
<td>11</td>
<td>66</td>
<td>4.56</td>
<td>28.86</td>
<td>0.73</td>
<td>0.84</td>
<td>0.14</td>
<td>0.05</td>
</tr>
<tr>
<td>12</td>
<td>72</td>
<td>7.33</td>
<td>47.39</td>
<td>0.58</td>
<td>0.84</td>
<td>0.19</td>
<td>0.05</td>
</tr>
</tbody>
</table>

In Section 6.3.4, we give some computational results on solving ATSP diamond instances by various formulations.

6.3.4 Solving Diamond Instances by Various ATSP Formulations

We solved some directed diamond instances, for \( k = 2, ..., 12 \), using the LP relaxation of the Carr, Claus, Flood, FGG, MI, and MTZ formulations. All the formulations found the optimal solution for all these instances. The solution times for the formulations are shown in Table 6.14, and the Cplex iterations for the formulations are shown in Table 6.15. The MTZ and FGG formulations require the lowest Cplex iterations, and the MI and MTZ formulations require the least amount of solution time. These results are illustrated in Figures 6.13 and 6.14.
6.3 The TSP Instances by Papadimitriou and Steiglitz

Table 6.15: Cplex Iterations for Solving Directed Diamond with ATSP Formulations

<table>
<thead>
<tr>
<th>k</th>
<th>Size</th>
<th>Carr</th>
<th>Claus</th>
<th>FGG</th>
<th>Flood</th>
<th>MI</th>
<th>MTZ</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>12</td>
<td>146</td>
<td>1230</td>
<td>10</td>
<td>9</td>
<td>7</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>18</td>
<td>1351</td>
<td>5636</td>
<td>16</td>
<td>105</td>
<td>38</td>
<td>16</td>
</tr>
<tr>
<td>4</td>
<td>24</td>
<td>3000</td>
<td>16510</td>
<td>24</td>
<td>206</td>
<td>88</td>
<td>22</td>
</tr>
<tr>
<td>5</td>
<td>30</td>
<td>5261</td>
<td>34980</td>
<td>30</td>
<td>384</td>
<td>243</td>
<td>28</td>
</tr>
<tr>
<td>6</td>
<td>36</td>
<td>12422</td>
<td>66997</td>
<td>37</td>
<td>587</td>
<td>464</td>
<td>35</td>
</tr>
<tr>
<td>7</td>
<td>42</td>
<td>24594</td>
<td>120050</td>
<td>52</td>
<td>400</td>
<td>339</td>
<td>42</td>
</tr>
<tr>
<td>8</td>
<td>48</td>
<td>43944</td>
<td>191265</td>
<td>58</td>
<td>794</td>
<td>751</td>
<td>46</td>
</tr>
<tr>
<td>9</td>
<td>54</td>
<td>58468</td>
<td>296440</td>
<td>69</td>
<td>974</td>
<td>1048</td>
<td>52</td>
</tr>
<tr>
<td>10</td>
<td>60</td>
<td>93089</td>
<td>422779</td>
<td>77</td>
<td>1565</td>
<td>1270</td>
<td>58</td>
</tr>
<tr>
<td>11</td>
<td>66</td>
<td>133179</td>
<td>633209</td>
<td>78</td>
<td>2256</td>
<td>1599</td>
<td>64</td>
</tr>
<tr>
<td>12</td>
<td>72</td>
<td>178982</td>
<td>949096</td>
<td>95</td>
<td>3276</td>
<td>2162</td>
<td>70</td>
</tr>
</tbody>
</table>

Figure 6.13: Cplex Solution Seconds for Solving ATSP Diamond Instances

Figure 6.14: Number of Cplex Iterations for Solving ATSP Diamond Instances
6.4 Conclusion

In this chapter we compared the performance of the LP relaxation of various TSP formulations with the LP relaxation of the MI formulation. We solved some STSP and ATSP instances from the TSPLIB, and some diamond instances by Papadimitriou and Steiglitz (1978) for this purpose. In solving the TSPLIB instances the MI relaxation seems to outperform other formulations either in terms of solution time and number of iterations, or the quality of the solutions. The solution times and number of Cplex iterations for the MI formulation are significantly lower than those of other formulations with the same LP relaxation value. Despite their lower solution times, the gap with the optimal solution given by the FGG, Flood, and MTZ formulations are significantly higher than that of the MI formulation. For the ATSP instances in the TSPLIB, we found that the MI formulation performs better than the Carr, Claus, and Flood formulations in terms of solution times and number of iterations. With a gap equal to that of the Claus formulation, the MI formulation provides significantly smaller gaps with the optimal integer solution compared to the Carr, Flood, FGG, and MTZ formulations. For the ATSP instances reported by Gouveia and Pires (1999), the gap with the optimal solution for the MCF formulation dominates that of the MI formulation; however, solution times required by this formulation are significantly greater than those of the MI formulation.

The LP relaxation of the MI formulation found the integer solution to all the STSP diamond instances. the MI formulation also outperformed other formulations in terms of LP relaxation value or solution time. Although the solution times for the MTZ formulation is lower than the MI formulation, the solution provided by the MTZ formulation is significantly inferior to that of the MI formulation. For the ATSP diamond instances, the MI formulation outperformed the Carr, Claus, and Flood formulations in terms of solution time and number of iterations.

The good performance of the MI formulation is promising for consideration in future solution methods on problems with the TSP as their subproblem. In future studies instead of solving the MI formulation with the full set of constraints, it is planned to first solve a relaxation of the formulation including all the equality constraints and inequality constraints corresponding to $1 \leq i < j \leq 3$, and then add the violated constraints and solve the problem. It is also planned to prove analytically whether the LP relaxation of the MI formulation is able to find the integer solution of the diamond instances for any value of $k$, or not.
The Branch and Bound Method for the TSP

In this chapter we study the branch and bound method on the MI formulation compared to other models. In Section 7.1 we give a summary of previous studies on the branch and bound method for the TSP. We give a summary of various relaxations used in the branch and bound method for the TSP, and a summary of various branching rules applied on the TSP in Sections 7.2 and 7.3 respectively.

We define different branching rules for the MI formulation in Section 7.4. We compare the performance of these rules for the MI formulation followed by some computational results on applying branch and bound for various TSP models on some TSPLIB instances. The computational results are given in Section 7.5 followed by some conclusions and suggestions for future studies in Section 7.6.

7.1 A Brief History of the Branch and Bound Method for the TSP

The branch and bound method is based on the work by Dantzig et al. (1959) on the TSP. The term branch and bound originates from the algorithm suggested by Little et al. (1963). In a generic branch and bound method feasible solutions of an optimisation problem are enumerated to find the optimal integer solution. The problem is constantly split into two subproblems by adding two mutually exclusive and exhaustive constraints (branching), and lower bounds are used to construct a proof of optimality without exhaustive search (bounding). (Papadimitriou and Steiglitz 1982). Brown and Lomnicki (1966) have used the branch and bound method for a machine scheduling problem and provided an algorithm for finding the exact solution to the problem. Lawler and Wood (1966) have studied branch and bound methods, dedicating a section to applying the branch and bound method for the TSP. Bellmore and Nemhauser (1968) have compared different branch and bound variations.
7.2 Relaxations for the TSP

for the TSP. Balas and Toth (1985) have studied various branch and bound methods and relaxations for both ATSP and STSP. Applegate et al. (2006) have studied the branch and bound method as a part of a chapter on *History of TSP Computation* in their book. In this section we give the summary of these studies.

A combination of branch and bound method and use of cutting planes, known as the *branch and cut* method, was used for solving the TSP. The name branch and cut is due to the algorithm by Padberg and Rinaldi (1987) for a 532-city STSP instance. A successful application of the branch and cut method is the Concorde TSP Solver (Applegate et al., 2004). Concorde is by far the most efficient software implementation for solving STSP instances (Applegate et al., 2006).

### 7.2 Relaxations for the TSP

In order for the branch and bound method to perform well, it is important to start with an LP relaxation of the IP problem with a small gap. Different relaxations of the TSP have been considered by researchers to apply in the branch and bound methods. We give some of these results below.

**The assignment problem (AP) relaxation** of the ATSP is used by Eastman (1958), Little et al. (1963), and Bellmore and Malone (1971). The solution to the AP is either a directed tour or a set of directed sub-tours. Eastman used the network flow algorithms by Ford and Fulkerson (1956b) to solve the AP. The AP can be formulated as follows.

\[
\begin{align*}
\text{min} & \quad \sum_{i} \sum_{j} c_{ij} x_{ij} \\
\text{subject to:} & \\
\sum_{(i,j) \in A} x_{ij} &= 1, \quad \forall i \in V, \quad (7.1) \\
\sum_{(i,j) \in A} x_{ij} &= 1, \quad \forall j \in V, \quad (7.2) \\
x_{ij} &\leq 1, \quad \forall i, j \in V, \quad (7.3) \\
x_{ij} &\geq 0, \quad \forall i, j \in V. \quad (7.4)
\end{align*}
\]

The solution to the AP is either a directed tour or a set of directed sub-tours. Eastman used the network flow algorithms by Ford and Fulkerson (1956b) to solve the AP. The AP can be solved using the Hungarian method in \(O(n^3)\) time (Ahuja et al., 1993).

**The 2-matching relaxation** is used by Bellmore and Malone (1971) for the STSP. The answer to the 2-matching problem is either a tour or a collection
of subtours. The objective function of the 2-matching problem is the same as the AP, and its constraints can be formulated as follows.

\[ \sum_{i<j} x_{ij} + \sum_{j<i} x_{ij} = 2, \quad \forall i, j \in V, \quad (7.5) \]

\[ 0 \leq x_{ij} \leq 1, \quad \forall i, j \in V. \quad (7.6) \]

**The 1-tree relaxation** of the STSP is first used by Held and Karp (1971) and Christofides (1970). Let \( V' \) be \( V - \{1\} \). This relaxation is given as the following problem.

\[
\min \sum_i \sum_j c_{ij} x_{ij}
\]

subject to:

\[
\sum_{(i,j) \in S \times (V' - S), j > i} x_{ij} + \sum_{(i,j) \in (V' - S) \times S, j > i} x_{ij} \geq 1, \quad \forall S \subset V', \quad (7.7)
\]

\[
\sum_{i \in V} \sum_{j > i} x_{ij} = n, \quad (7.8)
\]

\[
\sum_{i \in V} x_{ij} = 2, \quad (7.9)
\]

\[
x_{ij} \leq 1, \quad \forall i, j \in V, \quad (7.10)
\]

\[
x_{ij} \geq 0, \quad \forall i, j \in V. \quad (7.11)
\]

The **n-path problem** finds the shortest path in a graph that includes \( n \) nodes (\( n \)-path) starting and ending at some node \( v \in V \). The LP relaxation of the \( n \)-path problem is first used by Houck Jr. et al. (1980) for the TSP. The \( n \)-path problems can be solved using dynamic programming in \( O(n^3) \) steps (Balas and Toth, 1985).

**Solving an LP with cutting planes** was first suggested by Gomory (1958) for solving generic integer programming optimisation problems. Crowder and Padberg (1980) used this solution method for the STSP. The main feature of their method is finding appropriate inequalities to use as cutting planes in each step (Balas and Toth, 1985).

Held and Karp (1970) improved the lower bound of the TSP by using the LP relaxation of a minimum spanning tree problem. If one of the nodes in the TSP graph is duplicated and regarded as city \( n + 1 \), the minimum spanning tree in the graph can be used as a lower bound for the TSP. Held and Karp used the minimum spanning tree problem to approximate the LP given as Problem 7.2.
Problem 7.2

\[
\min \sum_i \sum_j c_{ij} x_{ij}
\]

subject to:

\[
\sum_{j \in V} x_{ij} + \sum_{j \in V} x_{ji} = 2, \quad \forall i \in V, \quad (7.12)
\]

\[
\sum_{i \in S, j \notin S} x_{ij} + \sum_{i \in S, j \notin S} x_{ji} \geq 2, \quad S \subset V, S \neq \emptyset \quad (7.13)
\]

\[
x_{ij} \leq 1, \quad \forall i, j \in V, \quad (7.14)
\]

\[
x_{ij} \geq 0, \quad \forall i, j \in V. \quad (7.15)
\]

Held and Karp (1971) combined this lower bound with a branching rule that was designed to increase the lower bounds of the subproblems and succeeded in solving some instances as large as 64-nodes. Other researchers have explored various branch and bound method variations inspired by the solution method of Held and Karp. Amongst them is the duality based procedure suggested by Bazaraa and Goode (1977) for the STSP.

7.3 Branching Methods

We borrow the notations used by Balas and Toth (1985), for defining the subproblems in a branching tree. Starting with the root problem labeled as problem 1, we use string labels for the subproblems in a way that they show the hierarchy of the problem in the branch and bound tree. Given some problem \( m \) in the branching tree, let \( E_m \) indicate the set of edges \((i, j)\) that are excluded from problem \( m \), and let \( I_m \) indicate the set of edges included in the problem. Using variables \( x_{ij} \), the sets \( E_m \) and \( I_m \) for some subproblem \( m \) can be defined by the following condition.

\[
\begin{cases}
(i, j) \in I_m, & \text{if } x_{ij} = 1 \text{ in problem } m, \\
(i, j) \in E_m, & \text{if } x_{ij} = 0 \text{ in problem } m.
\end{cases} \quad (7.16)
\]

The solution to problem \( m \) is given subject to the constraints of the predecessor problem of \( m \) and subject to condition (7.16). Given problem \( m \), let the \( r \)th successor of problem \( m \) be labeled as problem \( m_r \). We use the notations \( I_{mr} \) and \( E_{mr} \) to indicate the set of variables that are included in and excluded from \( m_r \) problem respectively. Next we give some branching rules defined for the DFJ formulation.

Little et al. (1963) designed a branching rule that creates two subproblems where some arc \((i, j)\) is included in one branch and excluded from the other branch. This rule does not use the structure in the TSP and can
7.3 Branching Methods

be applied to any IP problem [Balas and Toth, 1985]. Little et al. do not solve the LP at each node but they calculate a lower bound for the LP using the dual solution. Their algorithm has more branching steps but requires less computation at each step compared to other methods (Lawler and Wood, 1966). At each branching step, Little et al. (1963) select a pair \((i, j)\) such that the subproblem corresponding to \(x_{ij} = 0\) would give a bound as large as possible (Lawler and Wood, 1966). The branching rule can be expressed as follows.

\[
\begin{align*}
E_{m1} &= E_m \cup \{(i, j)\}, & I_{m1} &= I_m, \\
E_{m2} &= E_m, & I_{m2} &= I_m \cup \{(i, j)\}.
\end{align*}
\]  

(7.17)

Eastman (1958) and later Shapiro (1966) developed a branching rule which given a set of edges \(\{(i_1, i_2), \ldots, (i_s, i_{s+1})\}\), creates a subproblem corresponding to each of these edges by setting its corresponding \(x_{ij}\) variable equal to zero. One drawback of this branching rule is that it does not give a partition of the feasible region of the problem and therefore different successors of a subproblem may give the same solution (Balas and Toth, 1985). The branching rule for a subproblem \(r\) branching from some problem \(m\), where \(1 \leq r \leq s\), is shown below.

\[
\begin{align*}
E_{mr} &= E_m \cup \{(i_r, i_{r+1})\}, \\
I_{mr} &= I_m.
\end{align*}
\]  

(7.18)

Bellmore and Nemhauser (1968) have noted that using Shapiro’s algorithm is more desirable compared to Little’s.

Murty (1968) improved the previous branching rule as shown below.

\[
\begin{align*}
E_{mr} &= E_m \cup \{(i_r, i_{r+1})\}, \\
I_{mr} &= I_m \cup \{(i_1, i_2), \ldots, (i_{r-1}, i_r)\}.
\end{align*}
\]  

(7.19)

Other variations to this rule are given by Bellmore and Malone (1971).

Volgenant and Jonker (1982) have given a branching rule for the 1-tree relaxation. Based on their branching rule, a vertex \(i\) the degree of which in the current 1-tree is greater than 2, along with two free edges \((i, j_1)\) and \((i, j_2)\) is chosen for branching. The subproblems defined by this branching rule are given as follows.

\[
\begin{align*}
E_{m1} &= E_m \cup \{(i, j): j \notin \{j_1, j_2\}\}, & I_{m1} &= I_m \cup \{(i, j_1), (i, j_2)\}, \\
E_{m2} &= E_m \cup \{(i, j_2)\}, & I_{m2} &= I_m \cup \{(i, j_1)\}, \\
E_{m3} &= E_m \cup \{(i, j_1)\}, & I_{m3} &= I_m.
\end{align*}
\]  

(7.20)

If \(i\) is incident to an edge in \(J_k\), then subproblem \(m1\) is not generated.
Other branching rules, TSP relaxations and bounds can be found in Balas and Toth (1985) and Applegate et al. (2006).

7.4 Branch and Bound Method for the MI Formulation

Similar to the notations used in the previous section for \( x_{ij} \) variables, we use the following notations based on the MI formulation structure to define subproblems in a branch and bound tree. Let the sets \( E_m \) and \( I_m \) correspond to the sets of \( x_{ijk} \) variables that are excluded from or included in the solution of some subproblem \( m \) respectively. We define \( E_m \) and \( I_m \) by the following condition for the subproblem \( m \).

\[
\begin{align*}
(i, j, k) & \in I_m, \quad \text{if } x_{ijk} = 1, \\
(i, j, k) & \in E_m, \quad \text{if } x_{ijk} = 0.
\end{align*}
\] (7.21)

We can use sets \( E_m \) and \( I_m \) to define subproblems \( m_1 \) and \( m_2 \) that partition the solution space of problem \( m \) into two regions.

**Example 7.4.1.** Let’s assume in the solution to some problem \( m \) a variable \( 0 < x_{i_1j_1k_1} < 1 \) is selected to branch on. As a branching rule we can have \( x_{i_1j_1k_1} = 1 \), for the first branch and \( x_{i_1j_1k_1} = 0 \), for the second branch of \( m \). Therefore we can define \( I_{m1} = I_1 \cup \{(i_1, j_1, k_1)\} \) and \( E_{m2} = E_m \cup \{(i_1, j_1, k_1)\} \), which shows the set of variables that are included in or excluded from \( m_1 \) and \( m_2 \) respectively. Figure 7.1 illustrates this example.

**Example 7.4.2.** Table 7.1 shows some solution to a 13-city MI-relaxation problem. Having \( x_{3,8,12} = 0.5 \), \( x_{3,10,12} = 0.5 \), suggests that edge \((3, 12)\) is created by having variables with \( k = 12 \) and \( i = 3 \) greater than zero. We use the following branching rule that partitions the solution space into two regions \( \sum_{i \neq 3, j < 12} x_{i,j,12} \) and \( \sum_{i=3, j < 12} x_{i,j,12} \). Similarly for \( k = 8 \) we have \( x_{3,7,8} = 0.5 \), \( x_{6,7,8} = 0.5 \), so we can have \( \sum_{j \neq 7} x_{ij,8} = 0 \), and \( \sum_{j=7} x_{ij,8} = 0 \). And finally based on having \( x_{3,6,10} = 0.5 \) and \( x_{6,8,10} = 0.5 \), we can use \( \sum_{i \neq 6, j \neq 6} x_{i,j,10} = 0 \), and \( \sum_{i=6 \lor j=6} x_{i,j,10} = 0 \), to partition the solution space into two regions.

The following branching rules generalize the branching formulations given in Example 7.4.2.

---

Figure 7.1: Branching Rule for a Problem
Table 7.1: Solution to the MI Relaxation of Some 13-city Problem

<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
<th>k</th>
<th>x_{ijk}</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>7</td>
<td>0.5</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>7</td>
<td>0.5</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>8</td>
<td>0.5</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>8</td>
<td>0.5</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>10</td>
<td>0.5</td>
</tr>
<tr>
<td>6</td>
<td>8</td>
<td>10</td>
<td>0.5</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>12</td>
<td>0.5</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>12</td>
<td>0.5</td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td>13</td>
<td>1</td>
</tr>
</tbody>
</table>

MI Branching Rule 1 (MIR₁)

Given the solution to some problem \( m \), if for some \( \hat{k} \) and some \( \hat{i} \) we have \( 0 < x_{ijk} < 1 \), then the successors of \( m \) are partitioned into two groups based on the following rules.

\[
\begin{cases} 
\mathcal{E}_1 = \mathcal{E}_m \cup \{(i, j, k)|i = \hat{i}, k = \hat{k}, i < j < k\}, I_{m1} = I_m, \\
\mathcal{E}_2 = \mathcal{E}_m, I_{m2} = I_m \cup \{(i, j, k)|i = i, k = \hat{k}, i < j < k\}, 
\end{cases}
\]  
(7.22)

Similarly if for some \( \hat{k} \) and some \( \hat{j} \) for some problem \( m \) we have \( 0 < x_{ij\hat{k}} < 1 \), then the successors of \( m \) are partitioned into two groups using the following rules.

\[
\begin{cases} 
\mathcal{E}_1 = \mathcal{E}_m \cup \{(i, j, k)|j = \hat{j}, k = \hat{k}, 1 < i < j\}, I_{m1} = I_m, \\
\mathcal{E}_2 = \mathcal{E}_m, I_{m2} = I_m \cup \{(i, j, k)|i = \hat{i}, k = \hat{k}, 1 < i < j\}, 
\end{cases}
\]  
(7.23)

More generally, if for some \( \hat{k} \) and some \( \hat{r} \) we have \( 0 < x_{ir\hat{k}} < 1 \) and/or \( 0 < x_{i\hat{r}k} < 1 \), then we can use the following branching rules.

\[
\begin{cases} 
\mathcal{E}_1 = \mathcal{E}_m \cup \{(i, j, k)|(i = \hat{r}, k = \hat{k}, i < j < k) \lor (j = \hat{r}, k = \hat{k}, 1 < i < j)\}, I_{m1} = I_m, \\
\mathcal{E}_2 = \mathcal{E}_m, \\
\mathcal{I}_1 = \mathcal{I}_m \cup \{(i, j, k)|(i = \hat{r}, k = \hat{k}, i < j < k) \lor (j = \hat{r}, k = \hat{k}, 1 < i < j)\}. 
\end{cases}
\]  
(7.24)

MI Branching Rule 2 (MIR₂)
7.4 Branch and Bound Method for the MI Formulation

Given some problem \( m \), if for some \( \hat{k} \) and some \( \hat{i} \), we have \( 0 < x_{i\hat{j}\hat{k}}, x_{i\hat{j}\hat{k}} < 1 \), the successors of \( m \) can be defined as follows.

\[
\begin{align*}
E_m &= E_m, \mathcal{J}_m = \mathcal{J}_m \cup \{(\hat{i}\hat{j}\hat{k})\}.
E_m &= E_m, \mathcal{J}_m = \mathcal{J}_m \cup \{(\hat{i}\hat{j}\hat{k})\}.
E_m &= E_m, \mathcal{J}_m = \mathcal{J}_m \cup \{(\hat{i}\hat{j}\hat{k})\}.
\end{align*}
\] (7.25)

Similarly, for some \( \hat{k} \) and some \( \hat{i} \), we have \( 0 < x_{i\hat{j}\hat{k}} < 1 \), and \( 0 < x_{i\hat{j}\hat{k}} < 1 \), the successors of \( m \) can be defined as follows.

\[
\begin{align*}
E_m &= E_m, \mathcal{J}_m = \mathcal{J}_m \cup \{(\hat{i}\hat{j}\hat{k})\}
E_m &= E_m, \mathcal{J}_m = \mathcal{J}_m \cup \{(\hat{i}\hat{j}\hat{k})\}
E_m &= E_m, \mathcal{J}_m = \mathcal{J}_m \cup \{(\hat{i}\hat{j}\hat{k})\}.
\end{align*}
\] (7.26)

More generally, if we have \( 0 < x_{i\hat{j}\hat{k}}, x_{i\hat{j}\hat{k}}, ..., x_{i\hat{j}\hat{k}} < 1 \), for some \( t < \tau(\_1) \), then we can use the following branching rule.

\[
\begin{align*}
E_m &= E_m, \mathcal{J}_m = \mathcal{J}_m \cup \{(\hat{i}\hat{j}\hat{k})\}
E_m &= E_m, \mathcal{J}_m = \mathcal{J}_m \cup \{(\hat{i}\hat{j}\hat{k})\}
E_m &= E_m, \mathcal{J}_m = \mathcal{J}_m \cup \{(\hat{i}\hat{j}\hat{k})\}.
\end{align*}
\] (7.27)

**MI Branching Rule 3 (MIR3)**

This branching rule is the same as the generic branching rule for the IP methods. Given an MI relaxation solution for some \( 0 < x_{i\hat{j}\hat{k}} < 1 \), we define the following branching rule.

\[
\begin{align*}
E_m &= E_m, \mathcal{J}_m = \mathcal{J}_m \cup \{(\hat{i}\hat{j}\hat{k})\},
E_m &= E_m \cup \{(\hat{i}\hat{j}\hat{k})\}, \mathcal{J}_m = \mathcal{J}_m.
\end{align*}
\] (7.28)

These three branching rules are illustrated schematically in Figure 7.2.

When choosing \( x_{i\hat{j}\hat{k}} \) variables to branch on, we have the option of choosing variables with \( k \) values as close to \( n \) or as close to 4 as possible. For example in the solution given in Example 7.4.2, we can choose between branching on \( x_{i\hat{j}\hat{k}} \) variables with \( k = 7 \) or \( k = 12 \). We represent the combination of a branching rule MIR with branching on variables of either greatest value of \( k \) or smallest values of \( k \), using MIR\(_{i,1}\), and MIR\(_{i,2}\), respectively.

In the next section we give some computational results on these different branching rules for some TSPLIB instances.
7.4 Branch and Bound Method for the MI Formulation

Figure 7.2: A Schematic Illustration of the Three MI Branching Rules: (a) MIR\textsubscript{1}, (b) MIR\textsubscript{2}, (c) MIR\textsubscript{3}
7.5 Computational Results on Branch and Bound for Various TSP Formulations

We applied the different MI branching rules, MIR_{i,j} for i = 1, 2, 3 and j = 1, 2, on some TSPLIB instances. We use Cplex 9.1 for solving the subproblems in each tree node. The solution times are compared in Table 7.2 and the sizes of the branch and bound trees are compared in Table 7.3. These results are also illustrated in Figure 7.3 and Figure 7.4. The median, minimum and maximum values of the solution times and the sizes of the trees are shown in these figures. The first and third quartiles are illustrated using boxes.

From Figure 7.3 we can observe that MIR_{1,1} and MIR_{1,2} provide smaller branch and bound trees compared to other methods. The solution times for MIR_{1,1} are less than those for other rules, except for problem st70, a Euclidean 70-city problem. For the MIR_{1} rules, branching on the variables with smallest value of k seems to perform better than using larger values for k.

We applied branch and bound with the Carr, Claus, DFJ and Wong models and compared with branch and bound on the MI formulation. We do not solve instances using the FGG, Flood or MTZ models, since the LP gaps of these models that are to be closed by branch and bound are very large, which makes the branch and bound method very inefficient. The results are given in Table 7.4 and 7.5 and illustrated in Figures 7.5 and 7.6. The number of violated subtour elimination constraints for the DFJ model that are found and added to the subproblems are shown in Table 7.5 in column SEC. Apart from the DFJ
Table 7.3: Size of the Branch and Bound Tree with MI Branching Rules

<table>
<thead>
<tr>
<th>Problem</th>
<th>MIR$_{1,1}$</th>
<th>MIR$_{2,1}$</th>
<th>MIR$_{3,1}$</th>
<th>MIR$_{1,2}$</th>
<th>MIR$_{2,2}$</th>
<th>MIR$_{3,2}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>bayg29</td>
<td>5</td>
<td>7</td>
<td>5</td>
<td>3</td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td>bayx29</td>
<td>7</td>
<td>7</td>
<td>11</td>
<td>5</td>
<td>16</td>
<td>15</td>
</tr>
<tr>
<td>dantzig42</td>
<td>11</td>
<td>7</td>
<td>13</td>
<td>5</td>
<td>7</td>
<td>11</td>
</tr>
<tr>
<td>swiss42</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>att48</td>
<td>5</td>
<td>10</td>
<td>9</td>
<td>17</td>
<td>25</td>
<td>17</td>
</tr>
<tr>
<td>hkk48</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>9</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>brazil58</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>st70</td>
<td>99</td>
<td>147</td>
<td>151</td>
<td>19</td>
<td>13</td>
<td>39</td>
</tr>
<tr>
<td>eil76</td>
<td>19</td>
<td>53</td>
<td>13</td>
<td>33</td>
<td>41</td>
<td>73</td>
</tr>
<tr>
<td>rd1100</td>
<td>7</td>
<td>11</td>
<td>9</td>
<td>9</td>
<td>13</td>
<td>9</td>
</tr>
<tr>
<td>eil101</td>
<td>19</td>
<td>70</td>
<td>81</td>
<td>35</td>
<td>62</td>
<td>63</td>
</tr>
<tr>
<td>lin105</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>13</td>
<td>9</td>
</tr>
<tr>
<td>gr120</td>
<td>71</td>
<td>90</td>
<td>93</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Figure 7.3: Branch and Bound Solution Seconds for Various MI Branching Rules

Figure 7.4: Branch and Bound Tree Size for Various MI Branching Rules
### 7.5 Computational Results on Branch and Bound for Various TSP Formulations

<table>
<thead>
<tr>
<th>Problem</th>
<th>Carr</th>
<th>Claus</th>
<th>DFJ</th>
<th>MIR$_{1,1}$</th>
<th>MIR$_{2,1}$</th>
<th>MIR$_{3,1}$</th>
<th>Wong</th>
</tr>
</thead>
<tbody>
<tr>
<td>bayg29</td>
<td>7</td>
<td>19</td>
<td>5</td>
<td>5</td>
<td>7</td>
<td>5</td>
<td>19</td>
</tr>
<tr>
<td>bays29</td>
<td>11</td>
<td>33</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>11</td>
<td>33</td>
</tr>
<tr>
<td>dantzig42</td>
<td>11</td>
<td>55</td>
<td>5</td>
<td>11</td>
<td>7</td>
<td>13</td>
<td>49</td>
</tr>
<tr>
<td>swiss42</td>
<td>5</td>
<td>17</td>
<td>5</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>27</td>
</tr>
<tr>
<td>att48</td>
<td>9</td>
<td>25</td>
<td>7</td>
<td>5</td>
<td>10</td>
<td>9</td>
<td>27</td>
</tr>
<tr>
<td>hk48</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>13</td>
</tr>
<tr>
<td>brazil58</td>
<td>3</td>
<td>25</td>
<td>7</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>21</td>
</tr>
<tr>
<td>st70</td>
<td>-</td>
<td>-</td>
<td>117</td>
<td>99</td>
<td>147</td>
<td>151</td>
<td>-</td>
</tr>
<tr>
<td>eil76</td>
<td>-</td>
<td>-</td>
<td>117</td>
<td>19</td>
<td>53</td>
<td>13</td>
<td>-</td>
</tr>
<tr>
<td>rd100</td>
<td>-</td>
<td>-</td>
<td>7</td>
<td>7</td>
<td>11</td>
<td>9</td>
<td>-</td>
</tr>
<tr>
<td>eil101</td>
<td>-</td>
<td>-</td>
<td>47</td>
<td>19</td>
<td>70</td>
<td>81</td>
<td>-</td>
</tr>
<tr>
<td>lin105</td>
<td>-</td>
<td>-</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>-</td>
</tr>
<tr>
<td>gr120</td>
<td>-</td>
<td>-</td>
<td>2055</td>
<td>71</td>
<td>90</td>
<td>93</td>
<td>-</td>
</tr>
</tbody>
</table>

**Figure 7.5:** Branch and Bound Tree Size for TSP Models

model, all the three MI branching rules provide the smallest size for branching trees and require the least amount of computational time. For the DFJ model, the size of the branch and bound tree is greater than MI, except for problem eil76, but the computational time for the DFJ model is less than for the MI formulation. This is probably due to the small size of the LPs solved at each branch and bound node. The size of the branch and bound tree is significantly larger for the DFJ model for problem gr120 which is a 120-city problem with geographical distances compared to the MI formulation. The MI branching methods provide smaller branch and bound trees for gr120, although their solution time is greater than that of the DFJ model.
7.5 Computational Results on Branch and Bound for Various TSP Formulations

Table 7.5: Solution Seconds for Branch and Bound on Different Methods

<table>
<thead>
<tr>
<th>Problem</th>
<th>Carr</th>
<th>Claus</th>
<th>DFJ</th>
<th>SEC</th>
<th>MIR_{1,1}</th>
<th>MIR_{2,1}</th>
<th>MIR_{3,1}</th>
<th>Wong</th>
</tr>
</thead>
<tbody>
<tr>
<td>bayg29</td>
<td>29.87</td>
<td>136.07</td>
<td>3.97</td>
<td>65</td>
<td>1.15</td>
<td>1.41</td>
<td>1.04</td>
<td>558.0</td>
</tr>
<tr>
<td>bays29</td>
<td>45.88</td>
<td>218.88</td>
<td>2.26</td>
<td>34</td>
<td>1.47</td>
<td>1.43</td>
<td>2.23</td>
<td>934.7</td>
</tr>
<tr>
<td>dantzig42</td>
<td>547.43</td>
<td>4706.28</td>
<td>3.37</td>
<td>32</td>
<td>5.44</td>
<td>3.27</td>
<td>6.07</td>
<td>13777.5</td>
</tr>
<tr>
<td>swiss42</td>
<td>260.33</td>
<td>1418.20</td>
<td>2.85</td>
<td>44</td>
<td>1.42</td>
<td>1.83</td>
<td>1.45</td>
<td>4878.5</td>
</tr>
<tr>
<td>att48</td>
<td>3097.80</td>
<td>6765.88</td>
<td>9.26</td>
<td>285</td>
<td>4.70</td>
<td>8.69</td>
<td>8.46</td>
<td>15277.8</td>
</tr>
<tr>
<td>att48</td>
<td>1964.80</td>
<td>2415.82</td>
<td>4.84</td>
<td>139</td>
<td>2.93</td>
<td>3.64</td>
<td>2.92</td>
<td>6764.8</td>
</tr>
<tr>
<td>brazil58</td>
<td>5075.50</td>
<td>18168.93</td>
<td>6.53</td>
<td>131</td>
<td>6.23</td>
<td>8.34</td>
<td>6.98</td>
<td>23619.5</td>
</tr>
<tr>
<td>st70</td>
<td>-</td>
<td>-</td>
<td>125.21</td>
<td>90757</td>
<td>408.91</td>
<td>588.33</td>
<td>629.31</td>
<td>-</td>
</tr>
<tr>
<td>eil76</td>
<td>-</td>
<td>-</td>
<td>12.70</td>
<td>106</td>
<td>187.67</td>
<td>504.81</td>
<td>139.58</td>
<td>-</td>
</tr>
<tr>
<td>rd100</td>
<td>-</td>
<td>-</td>
<td>43.15</td>
<td>456</td>
<td>305.99</td>
<td>441.49</td>
<td>361.57</td>
<td>-</td>
</tr>
<tr>
<td>eil101</td>
<td>-</td>
<td>-</td>
<td>174.90</td>
<td>14006</td>
<td>1209.30</td>
<td>3342.42</td>
<td>3956.10</td>
<td>-</td>
</tr>
<tr>
<td>lin105</td>
<td>-</td>
<td>-</td>
<td>46.17</td>
<td>138</td>
<td>113.78</td>
<td>148.27</td>
<td>119.61</td>
<td>-</td>
</tr>
<tr>
<td>gr120</td>
<td>-</td>
<td>-</td>
<td>5411.50</td>
<td>4930140200</td>
<td>&gt; 10^4</td>
<td>&gt; 10^4</td>
<td>&gt; 10^4</td>
<td>-</td>
</tr>
</tbody>
</table>

Figure 7.6: Branch and Bound Solution Seconds for TSP Models

Figure 7.7: Branch and Bound Solution Seconds for MI and DFJ Models
7.6 Conclusion

In this chapter we suggested three branching rules for the MI formulation of the TSP, which we refer to as MIR$_1$, MIR$_2$, and MIR$_3$. We applied these three branching rules in branch and bound algorithms and found rule MIR$_1$ to perform better than the other two rules. We also compared the performance of branch and bound method on various TSP formulations by solving some TSPLIB instances. We found that the branch and bound method on the MI formulation performs better than other TSP formulations in terms of solution time and the size of the branch and bound tree. Considering the small solution time for instances smaller than 58, it seems feasible to use the branch and bound method with the branching rule MIR$_{1,1}$ for solving such TSP instances.
Some Heuristics Based on the MI Formulation

In this chapter we give some heuristics for the STSP that are based on the pedigree structure. In Section 8.1 we introduce a tree-like representation of pedigrees, which we use to illustrate the heuristics given throughout the following sections. In Sections 8.2–8.6 we give five different STSP heuristics, followed by their computational performance results on some TSPLIB instances in Section 8.7. The conclusions are given in Section 8.8.

8.1 The Pedigree Tree

We use a tree-like illustration of pedigrees which we refer to as the pedigree tree. We define two types of nodes in the pedigree tree; the vertex nodes and the edge nodes. The vertex nodes correspond to the nodes $4 \leq k \leq n$ that we use in the insertion process of the MI formulation. The edge nodes correspond to the edges that are generated by these insertions, and also the three initial edges $(1, 2)$, $(1, 3)$, and $(2, 3)$. Each vertex node $k \geq 4$ of the pedigree tree is directly descended from an edge node, which we refer to as its edge node parent. Also, two edge nodes are derived from each vertex node that correspond to the edges created by the insertion of node $k$ into its parent edge node, and they are sorted in increasing order of their corresponding edge labels. We use the terms parent node and descendant node in the same sense that is used in the graph theory for nodes of a tree. Example 8.1.1 shows a sample pedigree tree corresponding to a pedigree in $P_7$.

Example 8.1.1. Consider pedigree $W = ((1, 2), (1, 4), (4, 5), (2, 3))$, with its pedigree tree illustrated in Figure 8.1. We form the pedigree tree by having three edge nodes that represent edges $(1, 2)$, $(1, 3)$, and $(2, 3)$, descending from vertex node 3. Since in this pedigree vertex node 4 is inserted into edge node $(1, 2)$, node 4 is attached to leaf $(1, 2)$, and the two edge nodes corresponding to edges $(1, 4)$ and $(2, 4)$ descend from this node. Vertex node 5 is inserted into the edge node $(1, 4)$ that is descending from vertex node 4. The rest of the
The Pedigree Tree

Figure 8.1: The Pedigree Tree of the Pedigree in Example 8.1.1

pedigree tree is formed in a similar fashion. In this pedigree tree, vertex nodes 5 and 6 are descendants of node 4.

Given any vertex node \( k \geq 3 \) in the tree, let \( \text{desc}(k, i) \) for \( i = 1, 2 \) (or \( i = 1, 2, 3 \) if \( k = 3 \)), indicate the edge node number \( i \) that is descending from vertex node \( k \). Also given any vertex node \( k \) in the pedigree tree, let \( \text{parent}(k) \) indicate the edge node into which vertex \( k \) is inserted.

**Example 8.1.2.** Consider the pedigree tree in Example 8.1.1 again. The edges resulting from the insertion of node 6 into edge \((4, 5)\), are indicated as \(\text{desc}(6, 1) = (4, 6)\), and \(\text{desc}(6, 2) = (5, 6)\). Also, \(\text{parent}(6)\) refers to edge \((4, 5)\) in which node 6 is inserted.

Given a vertex node \( 4 \leq k \leq n \) in some pedigree tree, we refer to the set of all the vertex nodes \( k' > k \) that are inserted into the edge node descendants of node \( k \), as the branch of vertex node \( k \). For computational purposes, for each vertex node of the pedigree tree we store the total cost of all the node insertions corresponding to all the vertex nodes in its branch. We refer to this value as the branch cost of the vertex node. The total cost of a pedigree can be then calculated by adding the sum \( c_{12} + c_{13} + c_{23} \) to the branch costs of all the vertex nodes that are inserted into the edge nodes \((1, 2), (1, 3), \) and \((2, 3)\).

Next we give an STSP heuristic that uses a greedy approach to create pedigrees.
8.2 The Nearest Pedigree Heuristic

As mentioned in Section 1.5, the nearest neighbour (NN) heuristic for the STSP is used to create STSP tours using a greedy approach. These tours can be further improved using some improvement heuristics such as 2-opt or 3-opt. In the nearest neighbour heuristic, we select a random node from 1 to \( n \) as the first node of the tour. We select the rest of the free nodes one by one and add them to the tour, each time selecting the free node that is the nearest to the last node that was selected. Algorithm 11 is an algorithm for the nearest neighbour heuristic, which receives an \( n \times n \) cost matrix \( C \) and returns an \( n \)-tour. Algorithm 11 runs in \( O(n^2) \) time.

Using the pedigree structure and the multistage insertion process for the STSP, we can define a similar approach for creating a pedigree. We refer to this approach as the nearest pedigree (NP) heuristic. We randomly change the labels of the nodes each time running this algorithm to have a different insertion sequence of the nodes. Algorithm 12 is an algorithm for the NP heuristic that returns a pedigree \( W \) given an \( n \times n \) cost matrix \( C \). This algorithm runs in \( O(n^2) \) time.

We compare the performance of the NN and the NP heuristics in terms of CPU seconds and solution quality on some TSPLIB instances. The average solution value, the gap with the optimal solution, and CPU seconds over 100 runs for each instance are given in Table 8.1, and illustrated in Figures 8.2 – 8.3. The NP heuristic yields better solutions than the NN heuristic.

Next we give another heuristic that extracts pedigrees from a given solution to the LP relaxation of the MI formulation.
8.2 The Nearest Pedigree Heuristic

**Algorithm 12 A Nearest Pedigree Heuristic**

\[ O \leftarrow \text{a random permutation of integers from 1 to } n \]

\[ \text{Edges} \leftarrow [(1, 2), (1, 3), (2, 3)] \]

**for** \( k = 4 \) to \( n \)** do

Find some \( e \in \text{Edges} \) for which

\[ C[O(e(1)), O(k)] + C[O(e(2)), O(k)] - C[O(e(1)), O(e(2))] \]

is smallest

\( W(k - 3) \leftarrow e \)

remove \( e \) from \( \text{Edges} \), and add \([ (e(1), k), (e(2), k) ] \) to \( \text{Edges} \)

**end for**

Convert the edges of the pedigree \( W \) into their original labels using \( O \)

**return** \( W \)

**Table 8.1: Computational Comparison of the NN and the NP Heuristics**

<table>
<thead>
<tr>
<th>Problem</th>
<th>Optimal Value</th>
<th>NN Value</th>
<th>NN Gap</th>
<th>Time (ms)</th>
<th>NP Value</th>
<th>NP Gap</th>
<th>Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>gr17</td>
<td>2085</td>
<td>2312.3</td>
<td>9.83%</td>
<td>0.37</td>
<td>2157.2</td>
<td>3.35%</td>
<td>0.67</td>
</tr>
<tr>
<td>gr21</td>
<td>2707</td>
<td>3272.7</td>
<td>17.29%</td>
<td>0.40</td>
<td>2900.6</td>
<td>6.67%</td>
<td>0.81</td>
</tr>
<tr>
<td>gr24</td>
<td>1272</td>
<td>1704.2</td>
<td>25.36%</td>
<td>0.44</td>
<td>1369.0</td>
<td>7.08%</td>
<td>0.93</td>
</tr>
<tr>
<td>fri26</td>
<td>937</td>
<td>1069.1</td>
<td>12.36%</td>
<td>0.48</td>
<td>966.7</td>
<td>5.99%</td>
<td>1.03</td>
</tr>
<tr>
<td>bayg29</td>
<td>1610</td>
<td>2060.0</td>
<td>21.84%</td>
<td>0.51</td>
<td>1692.1</td>
<td>4.85%</td>
<td>1.19</td>
</tr>
<tr>
<td>bays29</td>
<td>2020</td>
<td>2355.2</td>
<td>14.23%</td>
<td>0.55</td>
<td>2122.4</td>
<td>4.83%</td>
<td>1.18</td>
</tr>
<tr>
<td>dantzig42</td>
<td>699</td>
<td>929.2</td>
<td>24.77%</td>
<td>0.79</td>
<td>754.6</td>
<td>7.37%</td>
<td>1.75</td>
</tr>
<tr>
<td>swiss42</td>
<td>1273</td>
<td>1595.7</td>
<td>20.23%</td>
<td>0.71</td>
<td>1372.4</td>
<td>7.24%</td>
<td>1.72</td>
</tr>
<tr>
<td>att48</td>
<td>10628</td>
<td>12983.4</td>
<td>18.14%</td>
<td>0.83</td>
<td>11293.3</td>
<td>5.89%</td>
<td>1.97</td>
</tr>
<tr>
<td>gr48</td>
<td>5046</td>
<td>6252.4</td>
<td>19.29%</td>
<td>0.81</td>
<td>5381.2</td>
<td>6.23%</td>
<td>1.90</td>
</tr>
<tr>
<td>hk48</td>
<td>11461</td>
<td>13661.1</td>
<td>16.10%</td>
<td>0.81</td>
<td>12330.5</td>
<td>7.08%</td>
<td>1.95</td>
</tr>
<tr>
<td>eil51</td>
<td>426</td>
<td>524.7</td>
<td>18.82%</td>
<td>0.88</td>
<td>459.2</td>
<td>7.23%</td>
<td>2.08</td>
</tr>
<tr>
<td>berlin52</td>
<td>7542</td>
<td>9413.7</td>
<td>19.88%</td>
<td>0.90</td>
<td>8425.5</td>
<td>10.49%</td>
<td>2.16</td>
</tr>
<tr>
<td>brazil58</td>
<td>25395</td>
<td>30372.7</td>
<td>16.39%</td>
<td>1.00</td>
<td>27221.2</td>
<td>6.71%</td>
<td>2.46</td>
</tr>
<tr>
<td>st70</td>
<td>675</td>
<td>834.7</td>
<td>19.14%</td>
<td>1.23</td>
<td>719.5</td>
<td>6.18%</td>
<td>2.94</td>
</tr>
<tr>
<td>eil76</td>
<td>538</td>
<td>664.5</td>
<td>19.04%</td>
<td>1.34</td>
<td>586.5</td>
<td>8.27%</td>
<td>3.32</td>
</tr>
<tr>
<td>pr76</td>
<td>108159</td>
<td>146619.3</td>
<td>26.23%</td>
<td>1.39</td>
<td>116339.5</td>
<td>7.03%</td>
<td>3.38</td>
</tr>
<tr>
<td>rat99</td>
<td>1211</td>
<td>1525.7</td>
<td>20.63%</td>
<td>1.86</td>
<td>1352.0</td>
<td>10.43%</td>
<td>4.78</td>
</tr>
<tr>
<td>kroA100</td>
<td>21282</td>
<td>26900.0</td>
<td>20.88%</td>
<td>1.88</td>
<td>23024.1</td>
<td>7.57%</td>
<td>4.66</td>
</tr>
<tr>
<td>kroB100</td>
<td>22141</td>
<td>28060.0</td>
<td>21.09%</td>
<td>1.81</td>
<td>23797.1</td>
<td>6.96%</td>
<td>4.68</td>
</tr>
<tr>
<td>kroC100</td>
<td>20749</td>
<td>26437.1</td>
<td>21.52%</td>
<td>2.99</td>
<td>22437.9</td>
<td>7.53%</td>
<td>5.05</td>
</tr>
<tr>
<td>kroD100</td>
<td>21294</td>
<td>27493.3</td>
<td>22.55%</td>
<td>1.79</td>
<td>23202.6</td>
<td>8.23%</td>
<td>4.59</td>
</tr>
<tr>
<td>kroE100</td>
<td>22068</td>
<td>27368.9</td>
<td>19.37%</td>
<td>1.83</td>
<td>23918.4</td>
<td>7.74%</td>
<td>4.57</td>
</tr>
<tr>
<td>rd100</td>
<td>7904</td>
<td>9982.8</td>
<td>20.82%</td>
<td>1.87</td>
<td>8705.8</td>
<td>9.21%</td>
<td>4.68</td>
</tr>
<tr>
<td>eil101</td>
<td>629</td>
<td>833.2</td>
<td>24.51%</td>
<td>1.85</td>
<td>690.0</td>
<td>8.84%</td>
<td>4.71</td>
</tr>
<tr>
<td>lin105</td>
<td>14379</td>
<td>18781.1</td>
<td>23.44%</td>
<td>2.40</td>
<td>15815.2</td>
<td>9.08%</td>
<td>5.41</td>
</tr>
<tr>
<td>pri107</td>
<td>44303</td>
<td>50454.7</td>
<td>12.19%</td>
<td>2.21</td>
<td>45797.2</td>
<td>3.26%</td>
<td>5.66</td>
</tr>
<tr>
<td>gr120</td>
<td>6942</td>
<td>9006.8</td>
<td>22.92%</td>
<td>2.37</td>
<td>7614.5</td>
<td>9.15%</td>
<td>6.20</td>
</tr>
<tr>
<td>pri124</td>
<td>59030</td>
<td>71474.1</td>
<td>17.41%</td>
<td>2.79</td>
<td>63228.6</td>
<td>6.64%</td>
<td>6.80</td>
</tr>
<tr>
<td>bier127</td>
<td>118282</td>
<td>147130.9</td>
<td>19.61%</td>
<td>2.44</td>
<td>132071.8</td>
<td>10.44%</td>
<td>6.32</td>
</tr>
</tbody>
</table>

Continued on next page
8.3 The Pedigree Extraction Heuristic

Considering the small gap with the optimal integer solution of the LP relaxation of the MI formulation, it is interesting to see if some pedigrees with small gaps with the optimal solution can be extracted from fractional solutions of the MI formulation. In this section we
8.3 The Pedigree Extraction Heuristic

We give an algorithm for a heuristic which we refer to as the **pedigree extraction (PE) heuristic**. Using Algorithm 8 in Section 5.3 we can extract all the pedigrees given a solution to the LP relaxation of the MI formulation. Because the number of all pedigrees corresponding to a given solution can be very large, we select some pedigrees and then use them in this heuristic. First we give two examples of some possible scenarios during the pedigree extraction process.

For each \(4 \leq k \leq n\), let \(E_k\) be the set of \((i, j)\) edges for which \(x_{ijk}\) is greater than zero in the LP solution. When extracting edges from a given LP solution if more than one edge is available for the \((k - 3)\)rd component of the pedigree, we take the edge \((i, j)\) that corresponds to the largest \(x_{ijk}\) value in the LP solution.

**Example 8.3.1.** Consider the solution given in Table 8.2 for \(n = 6\). Starting with \(k = 4\), we have \(E_4 = \{(1, 2)\}\) therefore we can have pedigree \(W_4 = ((1, 2))\). Continuing with \(k = 5\) we have \(E_5 = \{(2, 3)\}\), we extend the pedigree to \(W_5 = ((1, 2), (2, 3))\). For \(k = 6\) we have \(E_6 = \{(2, 5), (1, 3)\}\) therefore, we can extend pedigree \(W_5\) either as \(((1, 2), (2, 3), (2, 5))\), or \(((1, 2), (2, 3), (1, 3))\), but we give the preference to the second pedigree extension as edge \((1, 3)\) corresponds to a greater \(x\) value than edge \((2, 5)\).

**Example 8.3.2.** Consider the MI relaxation solution given in Table 8.3 for \(n = 7\). For \(k = 6\), we can extract two pedigrees \(((1, 2), (2, 4), (2, 5))\), and \(((1, 3), (3, 4), (3, 5))\) from the

![Figure 8.3: The CPU Seconds for the NN and the NP Heuristics](image-url)

<table>
<thead>
<tr>
<th>(k)</th>
<th>(x_{ijk})</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>(x_{124} = 1)</td>
</tr>
<tr>
<td>5</td>
<td>(x_{235} = 1)</td>
</tr>
<tr>
<td>6</td>
<td>(x_{256} = 0.25), (x_{136} = 0.75)</td>
</tr>
</tbody>
</table>

Table 8.2: The MI Relaxation Solution in Example 8.3.1
8.4 The 1-Discordant Local Search

Table 8.3: The MI Relaxation Solution in Example 8.3.2

<table>
<thead>
<tr>
<th>k</th>
<th>$x_{ijk}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>$x_{124} = 0.5$, $x_{134} = 0.5$</td>
</tr>
<tr>
<td>5</td>
<td>$x_{245} = 0.5$, $x_{345} = 0.5$</td>
</tr>
<tr>
<td>6</td>
<td>$x_{256} = 0.5$, $x_{356} = 0.5$</td>
</tr>
<tr>
<td>7</td>
<td>$x_{247} = 0.5$, $x_{347} = 0.5$</td>
</tr>
</tbody>
</table>

solution. For $k = 7$ we have $E_7 = \{(2,4),(3,4)\}$, but we cannot extend any of the two pedigrees using $x_{247}$ or $x_{347}$, since the generators for edges $(2,4)$ and $(3,4)$ do not exist in any of these two pedigrees. In this case we can extend the pedigrees using other available edges in the two pedigrees.

As shown in Example 8.3.2, if during the pedigree extraction process it is not feasible to use any of the suggested edges in $E_k$ for the $(k-3)^{rd}$ component of the pedigree, we either

1. select a random edge from the unused edges of the pedigree, or

2. select an unused edge in the pedigree for which the insertion cost of node $k$ is the smallest.

We use two methods of extracting pedigrees from the LP solution based on these two options. In Method I we use the first option and in Method II we use the second option. The computational results of applying Methods I and II on some TSPLIB instances are shown in Table 8.4 and illustrated in Figures 8.4 and 8.5.

Despite the small gap between the LP relaxation solution and the optimal solution, in most of the cases these two methods do not yield good solutions. Method II performs better than Method I in terms of solution quality, however, in comparing these approaches with the NP heuristic, the NP heuristic provides significantly better solutions.

Next we give three STSP improvement heuristics based on the pedigree structure.

8.4 The 1-Discordant Local Search

Given a pedigree, we define a neighbourhood for the pedigree which we refer to as its 1-discordant (1-disc) neighbourhood. Given an initial pedigree, we use this neighbourhood rule as part of a local search approach to find better solutions.

Definition 8.4.1. Given some pedigree $W$, a 1-disc neighbour of $W$ is a pedigree that can be formed by attaching a vertex node $k$, with no other vertex node descendants in the pedigree tree of $W$, to another feasible edge node, i.e. an edge node $(i,j)$ where $j < k$. 
### Table 8.4: Computational Results on Pedigree Extraction Using Methods I and II

<table>
<thead>
<tr>
<th>Problem</th>
<th>Optimal Value</th>
<th>MI Gap</th>
<th>Relaxation</th>
<th>Gap</th>
<th>Average with the IP Value</th>
<th>CPU Seconds</th>
<th>Method II</th>
<th>Gap</th>
<th>Average with the IP Value</th>
<th>CPU Seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>bayg29</td>
<td>1610</td>
<td>0.12%</td>
<td></td>
<td>1646.0</td>
<td>2.19%</td>
<td>0.01</td>
<td>1646.0</td>
<td>2.19%</td>
<td>0.01</td>
<td></td>
</tr>
<tr>
<td>bays29</td>
<td>2020</td>
<td>0.32%</td>
<td></td>
<td>2039.0</td>
<td>0.93%</td>
<td>0.01</td>
<td>2039.0</td>
<td>0.93%</td>
<td>0.01</td>
<td></td>
</tr>
<tr>
<td>dantzig42</td>
<td>699</td>
<td>0.29%</td>
<td></td>
<td>2311.3</td>
<td>69.76%</td>
<td>0.03</td>
<td>2090.0</td>
<td>65.21%</td>
<td>0.03</td>
<td></td>
</tr>
<tr>
<td>swiss42</td>
<td>1273</td>
<td>0.08%</td>
<td></td>
<td>1286.0</td>
<td>1.01%</td>
<td>0.03</td>
<td>1286.0</td>
<td>1.01%</td>
<td>0.03</td>
<td></td>
</tr>
<tr>
<td>att48</td>
<td>10628</td>
<td>0.23%</td>
<td></td>
<td>10685.0</td>
<td>0.53%</td>
<td>0.08</td>
<td>10685.0</td>
<td>0.53%</td>
<td>0.08</td>
<td></td>
</tr>
<tr>
<td>gr48</td>
<td>5046</td>
<td>1.72%</td>
<td></td>
<td>6823.2</td>
<td>26.05%</td>
<td>0.07</td>
<td>6425.0</td>
<td>21.46%</td>
<td>0.07</td>
<td></td>
</tr>
<tr>
<td>hk48</td>
<td>11461</td>
<td>0.14%</td>
<td></td>
<td>11955.0</td>
<td>4.13%</td>
<td>0.03</td>
<td>11955.0</td>
<td>4.13%</td>
<td>0.03</td>
<td></td>
</tr>
<tr>
<td>eil51</td>
<td>426</td>
<td>0.82%</td>
<td></td>
<td>1634.7</td>
<td>73.94%</td>
<td>0.05</td>
<td>720.0</td>
<td>40.83%</td>
<td>0.06</td>
<td></td>
</tr>
<tr>
<td>brazil58</td>
<td>25395</td>
<td>0.19%</td>
<td></td>
<td>25395.0</td>
<td>0.00%</td>
<td>0.07</td>
<td>25395.0</td>
<td>0.00%</td>
<td>0.07</td>
<td></td>
</tr>
<tr>
<td>st70</td>
<td>675</td>
<td>0.59%</td>
<td></td>
<td>3648.0</td>
<td>81.50%</td>
<td>0.05</td>
<td>1388.0</td>
<td>51.37%</td>
<td>0.05</td>
<td></td>
</tr>
<tr>
<td>eil76</td>
<td>538</td>
<td>0.19%</td>
<td></td>
<td>2426.9</td>
<td>77.83%</td>
<td>0.11</td>
<td>1019.0</td>
<td>47.20%</td>
<td>0.11</td>
<td></td>
</tr>
<tr>
<td>pr76</td>
<td>108159</td>
<td>2.81%</td>
<td></td>
<td>520842.9</td>
<td>79.23%</td>
<td>0.06</td>
<td>187386.0</td>
<td>42.28%</td>
<td>0.06</td>
<td></td>
</tr>
<tr>
<td>rat99</td>
<td>1211</td>
<td>0.41%</td>
<td></td>
<td>3361.1</td>
<td>63.97%</td>
<td>0.23</td>
<td>1387.0</td>
<td>12.69%</td>
<td>0.23</td>
<td></td>
</tr>
<tr>
<td>kroA100</td>
<td>21282</td>
<td>1.62%</td>
<td></td>
<td>169470.3</td>
<td>87.44%</td>
<td>0.10</td>
<td>54094.0</td>
<td>60.66%</td>
<td>0.11</td>
<td></td>
</tr>
<tr>
<td>kroB100</td>
<td>22141</td>
<td>1.39%</td>
<td></td>
<td>167730.0</td>
<td>86.80%</td>
<td>0.10</td>
<td>60131.0</td>
<td>63.18%</td>
<td>0.11</td>
<td></td>
</tr>
<tr>
<td>kroC100</td>
<td>20749</td>
<td>1.33%</td>
<td></td>
<td>157101.6</td>
<td>86.79%</td>
<td>0.38</td>
<td>67035.0</td>
<td>69.05%</td>
<td>0.40</td>
<td></td>
</tr>
<tr>
<td>kroD100</td>
<td>21294</td>
<td>0.72%</td>
<td></td>
<td>151350.7</td>
<td>85.93%</td>
<td>0.58</td>
<td>50654.0</td>
<td>57.96%</td>
<td>0.62</td>
<td></td>
</tr>
<tr>
<td>kroE100</td>
<td>22068</td>
<td>1.22%</td>
<td></td>
<td>146707.4</td>
<td>84.96%</td>
<td>0.18</td>
<td>69587.0</td>
<td>68.29%</td>
<td>0.19</td>
<td></td>
</tr>
<tr>
<td>rd100</td>
<td>7904</td>
<td>0.09%</td>
<td></td>
<td>53211.8</td>
<td>85.15%</td>
<td>0.21</td>
<td>19058.0</td>
<td>58.53%</td>
<td>0.22</td>
<td></td>
</tr>
<tr>
<td>eil101</td>
<td>629</td>
<td>0.24%</td>
<td></td>
<td>636.0</td>
<td>1.10%</td>
<td>0.49</td>
<td>636.0</td>
<td>1.10%</td>
<td>0.50</td>
<td></td>
</tr>
<tr>
<td>lin105</td>
<td>14379</td>
<td>0.06%</td>
<td></td>
<td>114563.3</td>
<td>87.45%</td>
<td>0.10</td>
<td>21938.0</td>
<td>34.46%</td>
<td>0.10</td>
<td></td>
</tr>
<tr>
<td>gr120</td>
<td>6942</td>
<td>0.44%</td>
<td></td>
<td>51875.2</td>
<td>86.62%</td>
<td>0.31</td>
<td>14096.0</td>
<td>50.75%</td>
<td>0.32</td>
<td></td>
</tr>
<tr>
<td>pr124</td>
<td>59030</td>
<td>1.63%</td>
<td></td>
<td>597792.5</td>
<td>90.13%</td>
<td>0.15</td>
<td>99010.0</td>
<td>40.38%</td>
<td>0.16</td>
<td></td>
</tr>
<tr>
<td>bier127</td>
<td>118282</td>
<td>0.72%</td>
<td></td>
<td>618326.2</td>
<td>80.87%</td>
<td>0.17</td>
<td>245000.0</td>
<td>51.72%</td>
<td>0.17</td>
<td></td>
</tr>
</tbody>
</table>
8.4 The 1-Discordant Local Search

**Figure 8.4**: The Gap with the Optimal Solution for Methods I and II

**Figure 8.5**: CPU Seconds for Methods I and II
Size of the 1-disc neighbourhood for a given pedigree is \( O(n^2) \). The following example shows some of the 1-disc neighbours the pedigree in Example 8.1.1.

**Example 8.4.1.** Consider the pedigree \( W = ((1, 2), (1, 4), (4, 5), (2, 3)) \) given in Example 8.1.1. We can attach vertex nodes 6 and 7 that have no other vertex node descendants, to other available edge nodes in the pedigree tree. Some of these 1-disc neighbours of \( W \) are shown in Figure 8.6. The pedigrees of some possible 1-disc neighbours of \( W \) are as follows:

\[
\begin{align*}
W_1 & = ((1, 2), (1, 4), (4, 5), (1, 3)) \\
W_2 & = ((1, 2), (1, 4), (4, 5), (4, 6)) \\
W_3 & = ((1, 2), (1, 4), (4, 5), (5, 6)) \\
W_4 & = ((1, 2), (1, 4), (4, 5), (1, 5)) \\
W_5 & = ((1, 2), (1, 4), (1, 3), (2, 3))
\end{align*}
\]

Next we give Algorithm 13 that searches amongst all the 1-disc neighbours of a given pedigree \( W \), given its \( n \times n \) cost matrix \( C \). The algorithm returns the best pedigree in the 1-disc neighbourhood of \( W \). Algorithm 13 runs in \( O(n^2) \) time.

**Algorithm 13** Finding the Best 1-Disc Neighbour of a Pedigree

\[
\begin{align*}
\text{least} & \leftarrow 0 \\
\text{change} & \leftarrow M \\
\text{for } k = n \text{ down to } 3 \text{ do} \\
& \quad \text{if vertex node } k \text{ has no other vertex node descendants then} \\
& \quad \quad \text{Find an unused edge node } e \text{ descending from vertex nodes smaller than } k \text{ for which} \\
& \quad \quad \quad \text{change} = \text{CostChange}(W, C, k, \text{parent}(k), e) \text{ is the smallest} \\
& \quad \quad \text{if } \text{change} < \text{least} \text{ then} \\
& \quad \quad \quad \text{least} \leftarrow \text{change} \\
& \quad \quad \quad W' \leftarrow \text{Modify}(W, k, e) \\
& \quad \quad \text{end if} \\
& \quad \text{end if} \\
\text{end for} \\
\text{return } W'
\end{align*}
\]

The subroutines CostChange and Modify that are used in this algorithm are given as Algorithms 14 and 15 respectively. The CostChange subroutine calculates the resulting change in the branch cost of a vertex node that is inserted in some edge node \( e_1 \), if it is to be inserted into some new edge node \( e_2 \). This subroutine is used for vertex nodes that don’t have any other vertex node descendants. Algorithm 14 returns the value by which the cost of some pedigree \( W \) has increased, given \( W \), a cost matrix \( C \), a vertex node \( k \), and edge nodes \( e_1 \) and \( e_2 \).

The Modify subroutine attaches a given vertex node to its given new edge node and returns the modified pedigree with updated branch costs. Algorithm 15 returns an updated
Figure 8.6: Some 1-disc Neighbours of the Pedigree in Example 8.4.1
8.5 The 2-Discordant Local Search

Algorithm 14 The CostChange Subroutine

\[
\text{cost} \leftarrow C(e_2(1), k) + C(e_2(2), k) - C(e_2) - C(e_1(1), k) - C(e_1(2), k) + C(e_1)
\]

\text{return cost}

Pedigree in which some given vertex node \( k \geq 4 \) is inserted into a new given edge node number \( i \) of vertex node \( j \). This algorithm runs in \( O(n) \) time.

Algorithm 15 The Modify Subroutine

\[
W' \leftarrow W
\]
\[
\text{In pedigree } W', \text{ parent}(k) \leftarrow \text{desc}(j, i)
\]
\[
desc(k, 1) \leftarrow (\text{desc}(j, i, 1), k)
\]
\[
desc(k, 2) \leftarrow (\text{desc}(j, i, 2), k)
\]

Update the branch cost values for vertex nodes \( k \) and \( j \).

Update the branch cost values of the direct and indirect parent nodes of vertex node \( j \) up to node 3.

Update the branch cost values of the previous direct and indirect parent nodes of vertex node \( k \) up to node 3.

\text{return } W'

In practice, we just run the Modify subroutine, when we want to switch from the current pedigree to a new pedigree to search a new neighbourhood. Some computational results on applying the 1-disc local search on TSPLIB instances are given in Section 8.7. Next we give another neighbourhood rule called the 2-discordant local search.

8.5 The 2-Discordant Local Search

Extending the idea of the 1-disc neighbourhood, we define another neighbourhood rule we refer to as the 2-discordant (2-disc) neighbourhood rule. By reattaching two vertex nodes of a given pedigree \( W \) to two new edge nodes, we can get a 2-disc neighbour of \( W \). These two vertex nodes are selected either from the set of all the vertex nodes that have no other vertex node descendants, or from the set of vertex node couples where one node is the only vertex node descendant of the other and has no vertex node descendants itself. The size of the 2-discordant neighbourhood for a given pedigree is \( O(n^4) \). The following example shows some 2-disc neighbours of a given pedigree.

Example 8.5.1. Consider pedigree \( W \) given in Example 8.1.1. To get 2-disc neighbours of \( W \), we can relocate vertex nodes 6 and 7, and also vertex nodes 5 and 6 and attach them to new edge nodes. Some of these possible pedigrees are illustrated in Figure 8.7.

Next we give Algorithm 16 which receives a pedigree \( W \) and its \( n \times n \) cost matrix \( C \), and searches in the 2-disc neighbours of the pedigree and returns the best neighbour. In
Figure 8.7: Some 2-disc Neighbours of the Pedigree in Example 8.5.1
this algorithm we use a subroutine called Modify\textsubscript{2} that is given as Algorithm 17. Algorithm 16 runs within $O(n^4)$ time. Since in this algorithm we just consider and relocate the vertex nodes at the end of trees, both for relocating and for attaching new vertex nodes to, the actual running time is less than this.

Algorithm 16 Find the Best 2-Disc Neighbour of a Pedigree

\begin{algorithm}
\begin{algorithmic}
\State \textproc{least} $\leftarrow$ 0
\State \textproc{change} $\leftarrow$ $M$
\For {$k_1 = (n - 1)$ down to 4}
\For {$k_2 = (k_1 + 1)$ to $n$}
\If {(vertex nodes $k_3$ has no other vertex node descendants) \textbf{and} ($k_2$ is the only vertex node descendant of node $k_1$)}
\For {all edge couples $[e_1, e_2] = [\text{desc}(j_1, i_1), \text{desc}(j_2, i_2)]$}
\If {(both edges are unused) \textbf{and} ($((j_1 \neq j_2) \text{ or } (i_1 \neq i_2))$) \textbf{and} ($j_1 < k_1$) \textbf{and} ($j_2 < k_2$)}
\State \textproc{change} $\leftarrow$ CostChange($W, C, k_1, \text{parent}(k_1), \text{desc}(j_1, i_1)$)
\State \textproc{change} $\leftarrow$ \textproc{change} + CostChange($W, C, k_2, \text{parent}(k_2), \text{desc}(j_2, i_2)$)
\If {\textproc{change} < \textproc{least}}
\State \textproc{least} $\leftarrow$ \textproc{change}
\EndIf
\EndIf
\EndFor
\EndIf
\EndFor
\EndFor
\Return $W'$
\end{algorithmic}
\end{algorithm}

Algorithm 17 receives a pedigree, two vertex nodes $k_1$ and $k_2$, and two edge nodes $e_1$ and $e_2$ and returns a pedigree where vertex node $k_1$ and $k_2$ are inserted into $e_1$ and $e_2$ respectively. This algorithm runs in $O(n)$ time.

The computational results on applying the 2-disc local search on some TSPLIB instances in comparison with other heuristics are given in Section 8.7. Next we give another improvement heuristic called the pedigree branch heuristic.

### 8.6 The Pedigree Branch Local Search

In this section we define a neighbourhood rule we refer to as the \textit{pedigree branch}. To get a pedigree branch neighbour of a given pedigree, we connect an entire branch of a vertex node $k$ to a new \textit{feasible} edge node $(i,j)$. The edge node $(i,j)$ is considered feasible if
Algorithm 17 The Modify\(_2\) Subroutine

\[
\begin{align*}
W' & \leftarrow W \\
\text{In pedigree } W'; \quad \text{parent}(k_1) & \leftarrow e_1 \\
\text{desc}(k, 1) & \leftarrow (e_1(1), k) \\
\text{desc}(k, 1) & \leftarrow (e_1(2), k) \\
\text{parent}(k_2) & \leftarrow e_2 \\
\text{desc}(k, 2) & \leftarrow (e_2(1), k) \\
\text{desc}(k, 2) & \leftarrow (e_2(2), k)
\end{align*}
\]

Update the branch cost values for vertex nodes \(k_1, k_2, e_1(1), \text{ and } e_2(1)\)

Update the branch cost values of the direct and indirect parent nodes of vertex nodes \(k_1\) and \(k_2\) up to node 3

Update the branch cost values of the previous direct and indirect parent vertex nodes of nodes \(e_1(1)\) and \(e_2(1)\) up to node 3

return \(W'\)

it does not have any vertex node descendants, and also if \(j < k\). When connecting the branch of a vertex node to a new edge node, we do not change the configuration of the branch but only the \(i\) and \(j\) values of the edge nodes in that branch that may have been affected by this change. The following example shows the pedigree branch neighbours of a given pedigree, and also illustrates the modifications that are made to the edge nodes of a branch after attaching the branch to a new edge node.

**Example 8.6.1.** Consider pedigree \(W\) given in Example 8.1.1 (Figure 8.8, top left). We can attach the branches of vertex nodes 4, 5, 6, and 7 to new edge nodes. Considering vertex node 4, since edge (2, 3) is used by node 6, the branch of node 4 can only be attached to edge (1, 3). The edge nodes descending from vertex nodes 4, 5, and 7 are modified accordingly (Figure 8.8, bottom left). As for vertex node 5, the branch of the node can be attached to edges (1, 3) and (1, 4) (Figure 8.8, top right). Similarly the branch of node 6, which includes only vertex node 6, can be attached to edges (1, 3), (1, 4), and (4, 5) (Figure 8.8, bottom right). Reattaching branches with only one vertex node is equivalent to creating 1-disc neighbours of a pedigree.

Size of the pedigree branch neighbourhood is \(O(n^2)\), but it is larger in size than the 1-disc neighbourhood. The reason for this is that in the pedigree branch neighbourhood, we reconnect all the vertex nodes to new edge nodes whereas in the 1-disc neighbourhood we only relocate vertex nodes without any vertex node descendants. In Example 8.6.2 we compare the size of these two neighbourhoods for a given pedigree.

**Example 8.6.2.** Consider the pedigree in Example 8.1.1. In Table 8.5 we list the edge nodes that each vertex node can be connected to under 1-disc and pedigree branch neighbourhood rules.
Figure 8.8: Some Pedigree Branch Neighbours of the Pedigree in Example 8.6.1
8.7 Computational Results

We use this idea in a local search algorithm where we consider each vertex node at a time to connect its branch to a new feasible edge node which would decrease the branch cost and therefore the total cost of the pedigree.

In the pedigree branch local search we use a subroutine called the BranchChange. This subroutine calculates the new cost of some branch from vertex node $k$, that is to be attached to some edge $e$. This subroutine updates the branch costs of the vertex nodes that are affected by this branch relocation. The subroutine is given as Algorithm 18. Algorithm 18 receives a pedigree $W$, a cost matrix $C$, a vertex node $k$, and an edge node $e$, and returns the change in the cost of the pedigree after attaching vertex node $k$ and its branch to edge $e$. This algorithm runs in $O(n)$ time.

Algorithm 18 The BranchChange Subroutine

```
change ← $C(e(1), k) + C(e(2), k) - C(e)$
for all the nodes in the branch of node $k$ that are affected by the relocation do
    add the cost of the new node insertions to change
end for
return change
```

We also use subroutines BranchCost and Modify that are given in Section 8.4. We give Algorithm 19 that receives a pedigree $W$, and its $n \times n$ cost matrix $C$, and finds the best pedigree branch neighbour for a given pedigree. This algorithm runs in $O(n^3)$ time.

In the next section we give the computational results of applying the heuristics given in the previous sections, on some STSP instances.

8.7 Computational Results

We compared the performance of the different improvement heuristics by applying them on some feasible integer solutions to some TSPLIB instances. These improvement heuristics
Algorithm 19 Finding the Best Pedigree Branch Neighbour of a Pedigree

\[
\text{least} \leftarrow 0
\]
\[\text{for } k = n \text{ down to } 4 \text{ do} \]
\[\text{for } j = 3 \text{ down to } k \text{ do} \]
\[\text{for } \text{leaf} = 1 \text{ to } 2 \text{ (or } 3 \text{ if } j = 3) \text{ do} \]
\[\text{if } e = \text{leaf}(j, i) \text{ is not used then} \]
\[\text{set } \text{change} = \text{BranchChange}(W, C, k, e) - \text{BranchCost}(W, k) \]
\[\text{if } \text{change} < \text{least} \text{ then} \]
\[\text{least} = \text{change} \]
\[W' = \text{Modify}(W, C, k, j, i) \]
\[\text{end if} \]
\[\text{end if} \]
\[\text{end for} \]
\[\text{end for} \]
\[\text{return } W' \]

are the 2-opt, the 3-opt, the 1-disc, the 2-disc, and the pedigree branch heuristics. Each one of these heuristics is applied on the initial pedigrees provided by the NP, and the PE heuristics. We use the pedigree extraction method II for finding initial solutions for the improvement heuristics. We compare these heuristic combinations in terms of solution quality, solution time, and the number of iterations. The performance results of the 2-opt, 3-opt, pedigree branch, 1-disc, and 2-disc heuristics are given in Tables \[8.6-8.10\] respectively. The heuristics’ performances in terms of the gap with the optimal solution, solution time, and number of iterations are illustrated in Figures \[8.9-8.14\] respectively.

In terms of solution quality, all the improvement heuristics that are using the initial solution by the PE method are providing significantly weaker results than the heuristics using the solution by the NP method. The only exception is the combination of the PE and the 2-opt heuristics which performs better than all other PE heuristic combinations. The gap with the optimal solution for the heuristics that are combined with the NP method are almost equal. However, the combinations with the 2-opt and the pedigree branch heuristics perform slightly better than the rest.

In terms of solution times, the pedigree branch and the 2-opt method both using the NP initial solutions require the least amount of CPU time. On the other hand, the 3-opt, 1-disc, and 2-disc heuristics all using the initial PE solutions, require significantly larger amount of time. In general the heuristics using the initial solution from the NP method, require fewer iterations compared to those using the PE method. The number of iterations for the 2-discordant and the 3-opt heuristics based on the NP method are smaller than other heuristics.
Table 8.6: Computational Results for the 2-opt Heuristic

<table>
<thead>
<tr>
<th>Problem</th>
<th>Optimal Value</th>
<th>Pedigree Extraction</th>
<th>Nearest Pedigree</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CPU Seconds</td>
<td>Final Value</td>
<td>Gap</td>
</tr>
<tr>
<td>bayg29</td>
<td>1610</td>
<td>0.01</td>
<td>1620.0</td>
</tr>
<tr>
<td>bays29</td>
<td>2020</td>
<td>0.02</td>
<td>2039.0</td>
</tr>
<tr>
<td>dantzig42</td>
<td>699</td>
<td>0.06</td>
<td>715.8</td>
</tr>
<tr>
<td>swiss42</td>
<td>1273</td>
<td>0.04</td>
<td>1286.0</td>
</tr>
<tr>
<td>att48</td>
<td>10628</td>
<td>0.10</td>
<td>10653.0</td>
</tr>
<tr>
<td>gr48</td>
<td>5046</td>
<td>0.10</td>
<td>5314.2</td>
</tr>
<tr>
<td>hk48</td>
<td>11461</td>
<td>0.06</td>
<td>11461.0</td>
</tr>
<tr>
<td>eil51</td>
<td>426</td>
<td>0.21</td>
<td>458.5</td>
</tr>
<tr>
<td>berlin52</td>
<td>7542</td>
<td>0.00</td>
<td>7542.0</td>
</tr>
<tr>
<td>brazil58</td>
<td>25395</td>
<td>0.09</td>
<td>25395.0</td>
</tr>
<tr>
<td>st70</td>
<td>675</td>
<td>0.62</td>
<td>724.9</td>
</tr>
<tr>
<td>eil76</td>
<td>538</td>
<td>0.81</td>
<td>576.6</td>
</tr>
<tr>
<td>pr76</td>
<td>108159</td>
<td>0.81</td>
<td>114297.1</td>
</tr>
<tr>
<td>rat99</td>
<td>1211</td>
<td>1.09</td>
<td>1269.4</td>
</tr>
<tr>
<td>kroA100</td>
<td>21282</td>
<td>3.36</td>
<td>22951.7</td>
</tr>
<tr>
<td>kroB100</td>
<td>22141</td>
<td>2.69</td>
<td>24452.7</td>
</tr>
<tr>
<td>kroC100</td>
<td>20749</td>
<td>1.33</td>
<td>21698.8</td>
</tr>
<tr>
<td>kroD100</td>
<td>21294</td>
<td>2.69</td>
<td>22667.6</td>
</tr>
<tr>
<td>kroE100</td>
<td>22068</td>
<td>0.91</td>
<td>23003.6</td>
</tr>
<tr>
<td>rd100</td>
<td>7904</td>
<td>2.02</td>
<td>8235.7</td>
</tr>
<tr>
<td>eil101</td>
<td>629</td>
<td>0.61</td>
<td>633.0</td>
</tr>
<tr>
<td>lin105</td>
<td>14379</td>
<td>0.57</td>
<td>14504.8</td>
</tr>
</tbody>
</table>

Figure 8.9: The Gap with the Optimal Solution for Heuristic Combinations (1)
### Table 8.7: Computational Results for the 3-opt Heuristic

<table>
<thead>
<tr>
<th>Problem</th>
<th>Optimal Value</th>
<th>Pedigree Extraction</th>
<th>Nearest Pedigree</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>CPU Seconds</td>
<td>Final Value</td>
</tr>
<tr>
<td>bayg29</td>
<td>1610</td>
<td>0.10</td>
<td>1646.0</td>
</tr>
<tr>
<td>bays29</td>
<td>2020</td>
<td>0.11</td>
<td>2039.0</td>
</tr>
<tr>
<td>dantzig42</td>
<td>699</td>
<td>9.01</td>
<td>878.3</td>
</tr>
<tr>
<td>swiss42</td>
<td>1273</td>
<td>0.38</td>
<td>1286.0</td>
</tr>
<tr>
<td>att48</td>
<td>10628</td>
<td>0.66</td>
<td>10685.0</td>
</tr>
<tr>
<td>gr48</td>
<td>5046</td>
<td>5.46</td>
<td>5405.2</td>
</tr>
<tr>
<td>hk48</td>
<td>11461</td>
<td>1.19</td>
<td>11525.0</td>
</tr>
<tr>
<td>eil51</td>
<td>426</td>
<td>27.40</td>
<td>510.4</td>
</tr>
<tr>
<td>berlin52</td>
<td>7542</td>
<td>0.00</td>
<td>7542.0</td>
</tr>
<tr>
<td>brazil58</td>
<td>25395</td>
<td>1.16</td>
<td>25395.0</td>
</tr>
<tr>
<td>st70</td>
<td>675</td>
<td>117.42</td>
<td>843.2</td>
</tr>
<tr>
<td>eil76</td>
<td>538</td>
<td>178.94</td>
<td>650.6</td>
</tr>
<tr>
<td>pr76</td>
<td>108159</td>
<td>174.17</td>
<td>135364.1</td>
</tr>
<tr>
<td>rat99</td>
<td>1211</td>
<td>189.07</td>
<td>1332.3</td>
</tr>
<tr>
<td>kroA100</td>
<td>21282</td>
<td>553.79</td>
<td>27165.9</td>
</tr>
<tr>
<td>kroB100</td>
<td>22141</td>
<td>610.83</td>
<td>27650.3</td>
</tr>
<tr>
<td>kroC100</td>
<td>20749</td>
<td>595.92</td>
<td>25364.6</td>
</tr>
<tr>
<td>kroD100</td>
<td>21294</td>
<td>643.13</td>
<td>26645.8</td>
</tr>
<tr>
<td>kroE100</td>
<td>22068</td>
<td>587.31</td>
<td>26921.7</td>
</tr>
<tr>
<td>rd100</td>
<td>7904</td>
<td>652.02</td>
<td>9929.3</td>
</tr>
<tr>
<td>eil101</td>
<td>629</td>
<td>14.23</td>
<td>633.0</td>
</tr>
<tr>
<td>lin105</td>
<td>14379</td>
<td>945.19</td>
<td>18526.3</td>
</tr>
</tbody>
</table>

**Figure 8.10:** The Gap with the Optimal Solution for Heuristic Combinations (2)
### Table 8.8: Computational Results for the Pedigree Branch Heuristic

<table>
<thead>
<tr>
<th>Problem</th>
<th>Optimal Value</th>
<th>Pedigree Extraction</th>
<th>Nearest Pedigree</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>CPU Seconds</td>
<td>Final Value</td>
</tr>
<tr>
<td>bayg29</td>
<td>1610</td>
<td>0.05</td>
<td>1620.0</td>
</tr>
<tr>
<td>bays29</td>
<td>2020</td>
<td>0.05</td>
<td>2020.0</td>
</tr>
<tr>
<td>dantzig42</td>
<td>699</td>
<td>0.81</td>
<td>716.7</td>
</tr>
<tr>
<td>swiss42</td>
<td>1273</td>
<td><strong>0.09</strong></td>
<td>1286.0</td>
</tr>
<tr>
<td>att48</td>
<td>10628</td>
<td>0.15</td>
<td>9625.0</td>
</tr>
<tr>
<td>gr48</td>
<td>5046</td>
<td>0.28</td>
<td>5326.4</td>
</tr>
<tr>
<td>hk48</td>
<td>11461</td>
<td>0.30</td>
<td>11684.0</td>
</tr>
<tr>
<td>eil51</td>
<td>426</td>
<td>0.95</td>
<td>518.0</td>
</tr>
<tr>
<td>berlin52</td>
<td>7542</td>
<td><strong>0.00</strong></td>
<td>7542.0</td>
</tr>
<tr>
<td>brazil58</td>
<td>25395</td>
<td>0.17</td>
<td>25395.0</td>
</tr>
<tr>
<td>st70</td>
<td>675</td>
<td>2.38</td>
<td>872.0</td>
</tr>
<tr>
<td>eil76</td>
<td>538</td>
<td>3.31</td>
<td>708.6</td>
</tr>
<tr>
<td>pr76</td>
<td>108159</td>
<td>7.76</td>
<td>119684.9</td>
</tr>
<tr>
<td>rat99</td>
<td>1211</td>
<td>7.09</td>
<td>1275.4</td>
</tr>
<tr>
<td>kroA100</td>
<td>21282</td>
<td>12.35</td>
<td>43844.2</td>
</tr>
<tr>
<td>kroB100</td>
<td>22141</td>
<td>12.66</td>
<td>34992.6</td>
</tr>
<tr>
<td>kroC100</td>
<td>20749</td>
<td>3.16</td>
<td>23198.0</td>
</tr>
<tr>
<td>kroD100</td>
<td>21294</td>
<td>7.39</td>
<td>28949.4</td>
</tr>
<tr>
<td>kroE100</td>
<td>22068</td>
<td>2.15</td>
<td>24844.1</td>
</tr>
<tr>
<td>rd100</td>
<td>7904</td>
<td>4.58</td>
<td>9177.7</td>
</tr>
<tr>
<td>eil101</td>
<td>629</td>
<td><strong>0.75</strong></td>
<td>636.0</td>
</tr>
<tr>
<td>lin105</td>
<td>14379</td>
<td>3.55</td>
<td>14416.8</td>
</tr>
</tbody>
</table>

Figure 8.11: CPU Seconds for Heuristic Combinations (1)
### Table 8.9: Computational Results for the 1-Disc Heuristic

<table>
<thead>
<tr>
<th>Problem</th>
<th>Optimal Value</th>
<th>Pedigree Extraction</th>
<th>Nearest Pedigree</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>CPU Seconds</td>
<td>Final Value</td>
</tr>
<tr>
<td>bayg29</td>
<td>1610</td>
<td>0.37</td>
<td>1642.0</td>
</tr>
<tr>
<td>bays29</td>
<td>2020</td>
<td>0.27</td>
<td>2020.0</td>
</tr>
<tr>
<td>dantzig42</td>
<td>699</td>
<td>5.95</td>
<td>1087.4</td>
</tr>
<tr>
<td>swiss42</td>
<td>1273</td>
<td>0.43</td>
<td>1286.0</td>
</tr>
<tr>
<td>att48</td>
<td>10628</td>
<td>1.09</td>
<td>10685.0</td>
</tr>
<tr>
<td>gr48</td>
<td>5046</td>
<td>3.31</td>
<td>5472.0</td>
</tr>
<tr>
<td>hk48</td>
<td>11461</td>
<td>1.88</td>
<td>11802.0</td>
</tr>
<tr>
<td>eil51</td>
<td>426</td>
<td>33.42</td>
<td>915.6</td>
</tr>
<tr>
<td>berlin52</td>
<td>7542</td>
<td>0.00</td>
<td>7542.0</td>
</tr>
<tr>
<td>brazil58</td>
<td>25395</td>
<td>1.77</td>
<td>25395.0</td>
</tr>
<tr>
<td>st70</td>
<td>675</td>
<td>168.85</td>
<td>1930.9</td>
</tr>
<tr>
<td>eil76</td>
<td>538</td>
<td>280.59</td>
<td>1285.7</td>
</tr>
<tr>
<td>pr76</td>
<td>108159</td>
<td>107.53</td>
<td>243359.4</td>
</tr>
<tr>
<td>rat99</td>
<td>1211</td>
<td>32.42</td>
<td>1802.7</td>
</tr>
<tr>
<td>kroA100</td>
<td>21282</td>
<td>1293.07</td>
<td>8653.65</td>
</tr>
<tr>
<td>kroB100</td>
<td>22141</td>
<td>1119.33</td>
<td>82532.6</td>
</tr>
<tr>
<td>kroC100</td>
<td>20749</td>
<td>1033.68</td>
<td>72275.6</td>
</tr>
<tr>
<td>kroD100</td>
<td>21294</td>
<td>1131.54</td>
<td>75935.3</td>
</tr>
<tr>
<td>kroE100</td>
<td>22068</td>
<td>894.26</td>
<td>63169.7</td>
</tr>
<tr>
<td>rd100</td>
<td>7904</td>
<td>1089.58</td>
<td>27111.2</td>
</tr>
<tr>
<td>eil101</td>
<td>629</td>
<td>19.05</td>
<td>636.0</td>
</tr>
<tr>
<td>lin105</td>
<td>14379</td>
<td>658.76</td>
<td>54576.4</td>
</tr>
</tbody>
</table>

**Figure 8.12:** CPU Seconds for Heuristic Combinations (2)
### 8.7 Computational Results

#### Table 8.10: Computational Results for the 2-Disc Heuristic

<table>
<thead>
<tr>
<th>Problem</th>
<th>Optimal Value</th>
<th>Pedigree Extraction</th>
<th>Nearest Pedigree</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>CPU Seconds</td>
<td>Final Value</td>
</tr>
<tr>
<td>bayg29</td>
<td>1610</td>
<td>0.20</td>
<td>1646.0</td>
</tr>
<tr>
<td>bayg29</td>
<td>2020</td>
<td>0.14</td>
<td>2039.0</td>
</tr>
<tr>
<td>dantzig42</td>
<td>699</td>
<td>10.23</td>
<td>1468.0</td>
</tr>
<tr>
<td>swiss42</td>
<td>1273</td>
<td>0.36</td>
<td>1286.0</td>
</tr>
<tr>
<td>att48</td>
<td>10628</td>
<td>1.00</td>
<td>10685.0</td>
</tr>
<tr>
<td>gr48</td>
<td>5046</td>
<td>3.34</td>
<td>5832.5</td>
</tr>
<tr>
<td>hk48</td>
<td>11461</td>
<td>0.75</td>
<td>11955.0</td>
</tr>
<tr>
<td>eil51</td>
<td>426</td>
<td>29.38</td>
<td>846.7</td>
</tr>
<tr>
<td>berlin52</td>
<td>7542</td>
<td>0.00</td>
<td>7542.0</td>
</tr>
<tr>
<td>brazil58</td>
<td>25395</td>
<td>1.68</td>
<td>25395.0</td>
</tr>
<tr>
<td>st70</td>
<td>675</td>
<td>172.40</td>
<td>1743.6</td>
</tr>
<tr>
<td>eil76</td>
<td>538</td>
<td>279.14</td>
<td>1144.2</td>
</tr>
<tr>
<td>pr76</td>
<td>108159</td>
<td>270.66</td>
<td>359692.7</td>
</tr>
<tr>
<td>rat99</td>
<td>1211</td>
<td>75.31</td>
<td>3035.9</td>
</tr>
<tr>
<td>kroA100</td>
<td>21282</td>
<td>1168.64</td>
<td>66367.5</td>
</tr>
<tr>
<td>kroB100</td>
<td>22141</td>
<td>960.25</td>
<td>68650.8</td>
</tr>
<tr>
<td>kroC100</td>
<td>20749</td>
<td>1203.78</td>
<td>59383.6</td>
</tr>
<tr>
<td>kroD100</td>
<td>21294</td>
<td>978.63</td>
<td>62988.5</td>
</tr>
<tr>
<td>kroE100</td>
<td>22068</td>
<td>977.25</td>
<td>49096.0</td>
</tr>
<tr>
<td>rd100</td>
<td>7904</td>
<td>1080.66</td>
<td>21438.2</td>
</tr>
<tr>
<td>eil101</td>
<td>629</td>
<td>35.67</td>
<td>635.0</td>
</tr>
<tr>
<td>lin105</td>
<td>14379</td>
<td>1433.42</td>
<td>96650.4</td>
</tr>
</tbody>
</table>

#### Figure 8.13: The Number of Iterations for Heuristic Combinations (1)
In Sections 8.2–8.6 we suggested some heuristics for the STSP using the pedigree structure. The NP heuristic given in Section 8.2 is suggested based on the idea of the nearest neighbour heuristic for STSP tours. This heuristic uses a greedy approach for creating pedigrees with respect to node insertion costs. The computational results of applying the NP and the NN heuristics on some of the TSPLIB instances showed that the NP heuristic performs better than the NN heuristic in terms of solution quality. Considering the good performance of the NP heuristic, it would be interesting to develop improved methods that use the node insertion costs based on the pedigree structure to create pedigrees of good quality.

We also suggested the PE heuristic in Section 8.3 that extracts pedigrees from given LP relaxation solutions of the MI formulation. Despite the significantly small gap of the LP solution with the optimal solution, in most of the cases the extracted pedigrees seem to be significantly inferior to the optimal solution. It seems that in the cases where the edges suggested by the LP solution are not feasible to be used in the extracted pedigree, the PE heuristic does not perform well. Future studies can include improved PE heuristics that address this issue, e.g. fixing these pedigree components based on the LP solutions first and then completing the rest of the extracted pedigree accordingly. It would also be interesting to find other methods that can benefit from the small gap of the LP relaxation of the MI formulation with the optimal solution.

We suggested three neighbourhood rules; the 1-disc, the 2-disc, and the pedigree branch. We applied these three rules in local search approaches on some TSPLIB instances with the initial solutions given by the NP or the PE heuristics. The combination of the local search methods with the NP heuristic seems to perform better than the PE heuristic. In comparing the suggested heuristics with the well known 2-opt and 3-opt methods, we found that the 2-opt and the pedigree branch methods using the solutions from the NP heuristic
provide the best results, and PE with the 2-disc and the 1-disc yield the worst results. Future studies can include improved heuristics based on the pedigree branch rules, that can create better pedigrees by swapping two node branches, or relocating more than one branch at the same time, or other similar approaches.
Conclusions

This thesis is focused on the MI formulation for the STSP designed by Arthanari (1983), and the MI formulation for the ATSP by Usha (1998). The MI formulation for the STSP is a compact model with \(O(n^2)\) constraints and \(O(n^3)\) variables. Some of the characteristics of the MI formulation and the pedigree polytope are shown through various studies by Arthanari (2005, 2006, 2007) and Arthanari and Usha (2000, 2001). In this thesis we report on some empirical studies performed on the MI formulation both for the STSP and the ATSP, and also for some related problems such as the membership problem in the pedigree polytope.

Grötschel et al. (1988) have shown how having an efficient algorithm for the membership or separation problems is promising for an efficient optimisation algorithm over a polytope. Arthanari (2008) has studied the membership problem in the pedigree polytope, and provided a necessary condition for the membership of an MI relaxation solution \(X/k + 1\) in the pedigree polytope \(\text{conv}(P_{k+1})\), given that \(X/k \in \text{conv}(P_k)\). This necessary condition is shown to be equivalent to having a multicommodity flow of one in some layered network, that is constructed using the MI relaxation solution. It is shown that the necessary condition for membership can be checked in polynomial time. Using a numerical illustration, we showed how this layered network can be constructed and the necessary condition be checked. We also showed the necessity of removing some arcs known as the dummy arcs, from the layered network. We answered the question of the necessary condition being sufficient in the negative, by providing a numerical example. We also suggested an LP model for checking the sufficient condition for membership in the pedigree polytope given an MI relaxation solution. It would be interesting to be able to improve the necessary condition for membership in the pedigree polytope in future studies. One possible way is to extract pedigree paths with rigid arcs from the network and assign fixed flow capacities to their entire path. Future studies can also include developing a separation algorithm for the pedigree polytope using the layered network.
We compared the performance of the LP relaxation of the MI formulation with other TSP models in solving some STSP and ATSP test problems in the TSPLIB, and some diamond instances by Papadimitriou and Steiglitz (1978). In solving the TSPLIB instances, the MI formulation performed better than other TSP models in terms of LP relaxation value or solution time in all the cases. Providing the same gap with the optimal solution as the formulations by Dantzig et al. (1954), Claus (1984), and Wong (1980), the MI formulation outperformed these models in terms of solution time and number of iterations. The MI formulation performed better than the formulation by Carr (1996) in terms of LP relaxation value for instances with 99 nodes or larger, and in terms of solution time for all the instances. Providing significantly better solutions compared to the models by Fox et al. (1980), Flood (1956), and Miller et al. (1960), the solution times for the MI formulation were slightly longer than these formulations. The MI formulations for the STSP and ATSP performed better than all the other TSP models in terms of number of Cplex iterations, except the MTZ model which provided significantly inferior solutions compared to the MI formulation. The gap with the optimal solution for the MCF formulation for some ATSP instances are smaller than those of the MI formulation; however, solution times required by this formulation are significantly greater than those of the MI formulation.

We solved diamond instances from size 14 to 200 by the LP relaxation of the MI formulation and found that in all the cases, the MI formulation yielded the optimal integer solution to the STSP diamond instances. Since the diamond instances have a repetitive structure, it is likely that the LP relaxation of the MI formulation can find the optimal solution to any diamond instance larger than 200 as well. Studying how the MI formulation can find an optimal solution to all diamond instances is an interesting topic that can be addressed in future studies.

The good performance of the LP relaxation of the MI formulation is promising for using the MI formulation in other solution methods. We developed some branching rules based on the structure of the MI formulation to be used in branch and bound solution methods. In comparing the performance of the MI formulation with other TSP models, the MI branching rules provided smaller branch and bound trees, and shorter solution times. Future studies can include improving these branching rules to achieve better solution methods.

Using the pedigree structure, we suggested some heuristics for the STSP. We designed a construction heuristic called the nearest pedigree using the similar ideas in the nearest neighbour heuristic. We tested this heuristic on some TSPLIB instances, and compared its performance with that of the nearest neighbour heuristic. We found the nearest pedigree heuristic to perform better than the nearest neighbour heuristic in terms of solution quality. Considering the good performance of the this heuristic, it would be interesting to develop improved methods that use the node insertion costs based on the pedigree structure in
construction heuristics. Observing the small gap with the optimal solution for the LP relaxation of the MI formulation, we suggested another heuristic that extracts pedigrees from a given LP relaxation solution of the MI formulation. We refer to this heuristic as the pedigree extraction heuristic. Despite the small gap of the LP solution with the optimal solution, we found that pedigree extraction heuristic does not perform well in cases where the edges suggested by the LP solution are not feasible to be included in the extracted pedigree. Future studies can include improved methods of constructing pedigrees given some LP relaxation solution of the MI formulation.

We suggested three neighbourhood rules based on the pedigree structure to be used in local search algorithms; the 1-discordant, the 2-discordant, and the pedigree branch. We applied these three rules in local search algorithms on some TSPLIB instances, with the initial solutions given by the nearest pedigree or the pedigree extraction heuristics. These methods seem to be sensitive to the initial solution and they performed better when used in combination with the nearest pedigree heuristic, compared to the pedigree extraction heuristic. We compared the suggested heuristics with the well known 2-opt and 3-opt methods, and found that the pedigree branch and the 2-opt methods combined with the nearest pedigree heuristic, provide the best results. Future studies can include improved heuristics based on the variations of the pedigree branch rules. Improving these neighbourhoods and checking whether they are exact neighbourhoods or not, are interesting problems that can be studied in future.

The good performance of the MI formulation seems promising for its application in future solution methods. Future studies can also include using the MI formulation as a part of other LP models e.g. vehicle routing problems. It is planned to study the performance of vehicle routing models when designed based on the MI formulation constraints.
Bibliography


BIBLIOGRAPHY


D. Naddef. The hirsch conjecture is true for (0, 1)-polytopes. *Mathematical Programming*, 45:109–110, 1989. 21


G. Reinelt. TSPLIB. [http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95](http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95), 1995.


