To the Graduate Council:

I am submitting herewith a dissertation written by Mohammad Mehdi Sepehri entitled "The Symmetric Generalized Traveling Salesman Problem." I have examined the final copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Management Science.

Charles E. Noon, Major Professor

We have read this dissertation
and recommend its acceptance:

Lori Kaplan

Kenneth Gilbert

Bruce A. Ralston

Accepted for the Council:

Vice Provost
and Dean of The Graduate School

THE SYMMETRIC GENERALIZED TRAVELING SALESMAN PROBLEM

A Dissertation

Presented for the

Doctor of Philosophy

Degree

The University of Tennessee, Knoxville

Mohammad Mehdi Sepehri

August 1991

To the memory of my brother

Amir Abolfazl Sepehri

ACKNOWLEDGEMENTS

I would like to express my deep gratitude to my chairman and mentor, Professor Charles E. Noon. This endeavor was not possible without his guidance and support. I am indebted to him because of his advice not only during the completion of this dissertation, but also throughout my graduate study at Knoxville. I am also indebted to the other members of my committee, Professors Kenneth Gilbert, Lori Kaplan and Bruce Ralston, for their valuable suggestions and timely feedback.

Several others helped in particular ways: Giovanni Rinaldi provided his subtour elimination code; Don Broach of The University of Tennessee Computing Center provided consulting support for LINDO; colleagues, Rekha Pillai, Scott Carl, and Cheryl Hild for their constant support and enduring friendship. My sincere thanks to each and every one of them.

To my mother and father, I wish to say thanks for all the love, confidence and encouragement you have always given me.

Finally, I extend my deepest appreciation to my wife and my children, Nasrin, Adnan, Hadis, and Amir, without whose inspiration, encouragement and loving patience this research would never have been completed.

ABSTRACT

The focus of this research is a mathematical model known as the Generalized Traveling Salesman Problem (GTSP). The GTSP is a generalization of a well known problem in the field of operations research known as the Traveling Salesman Problem (TSP). The GTSP can be simply described as follows. A salesman is given a list of cities that have been pregrouped into sets and a matrix of intercity distances. The salesman must decide a route of minimum total distance which begins at a home city, visits one city from each set, and then return to the home city. The GTSP is a difficult problem in the area of discrete optimization.

The GTSP is useful for modeling real-world problems which involve simultaneous decisions of selection and sequence. One such problem is the vehicle routing problem. Other applications of the GTSP include order-picking and location/routing problems. The GTSP has received little attention in the literature perhaps due to the longstanding difficulty in solving its special case, the TSP.

Our aim is to investigate a linear programming (LP) based cutting plane procedure for bounding the symmetric version of the GTSP. In the symmetric version of the problem, the distance associated with travel from city i to city j is the same as from city j to city i. The cutting plane procedure utilizes linear programming to solve a relaxed version of the problem and then uses specialized algorithms to identify violated constraints. When violated constraints are identified, they are added to

the LP formulation and the problem is re-solved. This is repeated until either the LP yields an integer solution or until no more violated constraints can be found.

In this dissertation, we analytically describe four classes of valid constraints for the symmetric GTSP. These constraints play a role in the development of the cutting plane procedure for computing lower and upper bounds on the total cost of the optimal solution. Our procedure includes algorithms for identifying violated constraints and heuristics for constructing feasible symmetric GTSP tours. We summarize the procedure's computational performance on 120 test problems. The results indicate that the cutting plane procedure is a viable method for producing optimal solutions or tight bounds on symmetric GTSP's of moderate size.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION AND PRELIMINARIES

In this chapter, we first introduce the symmetric generalized traveling salesman problem. We then provide the reader with a basic set of definitions and notation that will be used throughout this research. Next, an integer programming formulation of the problem is presented. We conclude the chapter with a discussion of the research objectives and give an overview of the organization of this dissertation.

1.1 Introduction and Problem Statement

Distribution, transportation, or logistics of goods and services are costly activities that have become diffused throughout every part of industries in both public and private sectors. The importance of these activities is due to the magnitude of the costs associated with them. Several authors have discussed this matter, among them Bodin et al. [1983], Fisher and Jaikumar [1981], and Magee et al. [1985]. Operations Research and Management Science (OR/MS) have played key roles in improving productivity and operations of these activities through fundamental and

applied research. Network models are one particular area of OR/MS that has many applications in distribution, transportation, routing and scheduling, facility location, and other related subjects. The traveling salesman problem (TSP) is an analytical and practical tool in network models that has been widely applied to tackle a variety of problems in OR/MS.

The objective of this research is to study a general form of the TSP known as the Generalized Traveling Salesman Problem (GTSP). Before introducing the GTSP we discuss its special case, the TSP. Consider a salesman, starting from his home city, who must visit each of a specified group of cities exactly once, and then return to his home city. The TSP is a problem of *sequencing* the cities to be visited so that the total distance traveled in his tour is minimized. Though the exact origin of this problem is not known, it is believed that the Icosian Game is the earliest initiation. A history of the TSP can be found in Hoffman and Wolfe [1985].

The TSP is of great importance to practitioners and researchers for two reasons. First, there are a number of direct applications of the TSP in a variety of problem settings (see Garfinkel [1985]). Second, the TSP is representation of other problems of its genre, combinatorial optimization.

The GTSP can be stated that, for a given collection of *sets* of cities, a salesman starting from his home city, must visit one city from each set and then return to his home city. Besides *sequencing* the sets to be visited, *selection* of one city from each set is the other decision that must be made. These two components of the GTSP, sequencing and selection, allow the GTSP to model a broad class of problems, thereby yielding numerous potential applications.

1.2 Definitions and Notation

The definitions and notational conventions used in this research are presented in this section. In addition, a glossary of mathematical symbols is given in Appendix A.

1.2.1 Basic Definitions and Notation

A *graph* $\mathcal{G} = (\mathcal{V},\mathcal{E})$ consists of a finite, nonempty set $\mathcal{V}$ of vertices or *nodes* given as $\mathcal{V} = \{ 1, 2, ..., n \}$, and a set $\mathcal{E}$ of *edges* where $\mathcal{E} = \{ (i;j) \mid i \in \mathcal{V}, j \in \mathcal{V}, i \neq j \}$. Each edge corresponds to a pair of distinct nodes called *end nodes* and is depicted by a line joining its end nodes. Therefore, a graph defines a set of nodes and their connecting lines. Every line joining two end nodes i and j can be either in the direction from i to j, or from j to i. We use the unordered pair (i;j) to denote the edge between i and j. Let e_{ij} and $x_{ij} \in \Re$ denote the edge (i;j) and an associated variable, respectively. Furthermore, we assume $i < j$ as convention and rewrite $\mathcal{E}$ as $\mathcal{E} = \{ (i;j) \mid i \in \mathcal{V}, j \in \mathcal{V}, i < j \}$.

We characterize an attribute for each edge, namely *edge value* which is denoted by $\bar{x}_{ij}$. In solution procedures, we assign a binary value to each edge variable, that is, $x_{ij} \in \{0,1\}$. We say $\bar{x}_{ij} = 1$ if edge (i;j) is selected to be traversed, and $\bar{x}_{ij} = 0$ if not. By relaxing the condition that the variable takes an integer value of either 0 or 1, we have $x_{ij} \in [0,1]$, that is $0 \leq x_{ij} \leq 1$. Note that the $\bar{x}_{ij}$ is the value of variable x_{ij}. The variables can be thought of as the

elements of an $|\mathcal{E}|$- dimensional vector x. For a given set of edges, F, let x(F) and $\bar{x}(F)$ be abbreviations for $\sum_{(i;j)\in F} x_{ij}$ and $\sum_{(i;j)\in F} \bar{x}_{ij}$ respectively.

Each edge (i;j) has an associated *edge cost*, $c_{ij} \in \Re$, where c_{ij} represents the distance or travel cost for traveling between nodes i and j. We assume that the cost from i to j or from j to i is identical, and hence it is *symmetric*. Let c be the cost vector associated with edges in $\mathcal{G}$; then c is of order of $|\mathcal{E}|$, that is, $c \in \Re^{|\mathcal{E}|}$ and can be written as $c = \{ c_{ij} \mid (i;j) \in \mathcal{E} \}$.

The graph $\mathcal{G} = (\mathcal{V},\mathcal{E})$ is considered an *undirected graph* since it has no implied direction on its edges. If the order of the nodes of an edge (i;j) is of importance, ordered pairs (i,j) and (j,i) are the two possible ways to consider. An ordered pair (i,j), which is depicted by a directed line from node i to node j, is called an *arc*. Usually the node i is called the *tail* of the arc (i,j) and j is called the *head* of the arc. A directed graph or *digraph* $\mathcal{D} = (\mathcal{N}, \mathcal{A})$ is a structure which consists of a finite and nonempty set $\mathcal{N}$ of nodes and a set $\mathcal{A}$ of arcs where $\mathcal{A} = \{ (i,j) \mid i \in \mathcal{N}, j \in \mathcal{N}, i \neq j \}$. A directed graph represents a set of nodes and their directed connections. The cost associated with a connection from i to j, arc (i,j), is denoted by $\vec{c}_{ij} \in \Re$ and may represent any possible cost from node i to node j. In this system, a cost vector associated with a digraph $\mathcal{D}$ is denoted by $\vec{c}$ and is of order of $|\mathcal{A}|$, let $\vec{c} = \{ \vec{c}_{ij} \mid (i,j) \in \mathcal{A} \}$. We may use a_{ij} to refer to the arc (i,j). We denote the arc variable and the arc value of an arc (i,j) by y_{ij} and $\bar{y}_{ij}$ respectively. We should note that any undirected graph $\mathcal{G} = (\mathcal{V},\mathcal{E})$ with $c = \{ c_{ij} \mid (i;j) \in \mathcal{E} \}$ can be represented as a directed graph $\mathcal{D} = (\mathcal{N}, \mathcal{A})$ with $\vec{c} = \{ \vec{c}_{ij} \mid (i,j) \in \mathcal{A} \}$, where every edge e_{ij} in $\mathcal{E}$ corresponds to a pair of arcs a_{ij} and a_{ji} in $\mathcal{A}$.

For $W \subseteq \mathcal{V}$, E(W) defines the set of edges with both end nodes in W, that is $E(W) = \{ (i;j) \mid (i;j) \in \mathcal{E}, i \in W, j \in W \}$. A *partial graph* of $\mathcal{G}$ is a graph $\mathcal{G}' = (\mathcal{V}',\mathcal{E}')$ with $\mathcal{V}' \neq \emptyset$, $\mathcal{V}' \subset \mathcal{V}$ and $\mathcal{E}' \subseteq \mathcal{E}$, where $\mathcal{E}' = E(\mathcal{V}')$, meaning $\mathcal{E}'$ is the set of all edges in $\mathcal{G}$ that have both end nodes in $\mathcal{V}'$.

Let d(i) denote the set of edges incident to node i and let its cardinality, $|d(i)|$, be referred to as the *degree of node* i. We have $0 \leq |d(i)| \leq n-1 \ \forall \ i \in \mathcal{V}$. If $|d(i)| = n-1 \ \forall \ i \in \mathcal{V}$ of a graph $\mathcal{G} = (\mathcal{V},\mathcal{E})$, it means that the graph has an edge for all possible pair of nodes and is called a *complete graph*. The number of edges in a complete graph is n(n-1)/2.

A *path* between nodes s and t is a finite sequence of edges of the form $(s;i_1), (i_1;i_2), \ldots,(i_k;t)$. If the nodes s, i_1, i_2, ..., i_k, t are distinct, we say that the path is a *simple path*. The nodes i and j are said to be *connected* if there exists a path between i and j. A graph is called a *connected graph*, if there is a path between all pairs of nodes of the graph. A *cycle* is a path having the same beginning and ending nodes, that is s = t, and a cycle is a *simple cycle* if no node is repeated in it. A *Hamiltonian cycle* or a *tour* of a connected graph is a simple cycle consisting of $|\mathcal{V}|$ edges that contains all of the nodes in $\mathcal{V}$. We denote a tour by listing its edges sequenced in a bracket. For example $[(i_1;i_2), (i_2;i_3), \ldots, (i_{k-1};i_k), (i_k;i_1)]$ shows a tour in a graph with k nodes. If a simple cycle does not contain all of the nodes in $\mathcal{V}$, it is called a *subtour*. The TSP is the problem of finding minimum cost Hamiltonian cycle in a connected graph or digraph. The symmetric TSP (STSP) is usually related to a graph, whereas the asymmetric TSP (ATSP) corresponds to a digraph.

Given $Ax \leq b$ as a finite number of inequalities, and $x \in \Re^n$, we implicitly assume compatibility of sizes of A, x, and b. A set P of points $x \in \Re^n$ is called

a *polyhedron* if $P = \{ x \in \Re^n \mid Ax \leq b \}$. A polyhedron that is bounded is known as a *polytope*. The inequality $\alpha x \leq \alpha_\circ$,denoted by $(\alpha, \alpha_\circ)$, is called a *valid inequality* (or *cutting plane*) for P if the inequality is satisfied by all the points in P.

1.2.2 GTSP Definitions and Notation

The Symmetric Generalized Traveling Salesman Problem (SGTSP) in its *canonical form* is defined on a graph $\mathcal{G} = (\mathcal{V},\mathcal{E})$ where $\mathcal{V} = \{ 1, 2, \ldots, n \}$ is a union of m nonempty and mutually exclusive *nodesets* S_I for $I \in M = \{ 1, 2, \ldots, m \}$ $m \geq 2$, that is $S_I \neq \emptyset$ and $S_I \cap S_J = \emptyset$ $\forall$ I,J$\in$M, I $\neq$ J. It is assumed that no intraset edges exist, meaning that for every edge (i;j), i,j$\notin S_I$ for all I$\in$M. This implies that all edges in $\mathcal{E}$ are interset edges, that is $\forall$ (i;j)$\in \mathcal{E}$ i$\in S_I$, j$\in S_J$, I $\neq$ J, and I,J$\in$M. We refer to the preceding graph, $\mathcal{G}$, as a canonical GTSP graph or simply a GTSP graph. Figure 1.1 shows an example of a GTSP graph in its canonical form with 5 sets, 15 nodes, and 20 edges. Unless otherwise specified, we assume that an SGTSP is defined on a GTSP graph in canonical form.

For W$\subseteq\mathcal{V}$, an index set S(W), called *covering nodesets* of W, is the set of indices of nodesets each of which has at least one node in W. So, $S(W) = \{ I \mid |S_I \cap W| \geq 1, I \in M \}$. For example, $S(\mathcal{V}) = \{ 1, 2, \ldots, m \}$ with $|S(\mathcal{V})| = m$.

We define D(I), in a GTSP graph, as the set of edges which have one of their end nodes in the nodeset S_I, that is, $D(I) = \{ (i;j) \mid (i;j) \in \mathcal{E}$, either i$\in S_I$ or j$\in S_I \}$. The cardinality of a set D(I) is called the *degree of nodeset* I and is

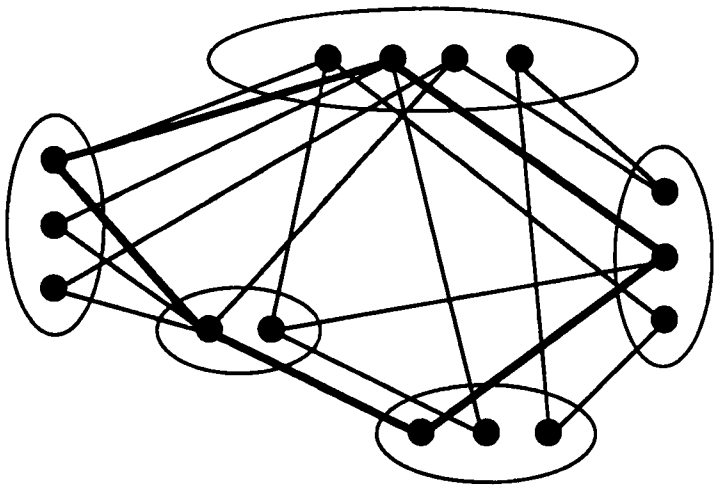


Figure 1.1: A GTSP graph in its canonical form.

shown by $|D(I)|$. Every edge contributes 2 to the sum of the degree of nodesets, since each edge is incident to two nodesets. Thus we have $\sum_{I=1}^{m} |D(I)| = 2|\mathcal{E}|$. If in a GTSP graph $|D(I)| = |S_I|(n-|S_I|)$ for all $I \in M$, the graph is a *complete GTSP graph*. Based on the preceding formulas the number of edges in a complete GTSP graph is equal to:

$$\frac{1}{2}\sum_{I=1}^{m} |S_I|(n-|S_I|) \equiv \sum_{I=1}^{m-1} |S_I|(|S_{I+1}| + \cdots + |S_m|).$$

In the case of having an equal number of p nodes in every nodeset, the total number of edges is $p^2m(m-1)/2$.

We define a *tour* of the SGTSP as a simple cycle that contains exactly one node from each nodeset of the GTSP graph. The SGTSP is the problem of finding minimum cost tour in a GTSP graph. The bold lines in Figure 1.1 illustrate a GTSP tour for the given graph. The symmetric TSP is a special case of the SGTSP with every nodeset of the SGTSP containing exactly one node. That is, a GTSP graph $\mathcal{G} = (\mathcal{V},\mathcal{E})$ with $\mathcal{V} = \{ 1, 2, ...,n \}$ and n nodesets

is a symmetric TSP of size n. The condition n > m for a GTSP graph indicates that the graph has at least one nodeset with more than one node in it.

Previously, we defined a subtour of a graph as a simple cycle which does not contain all of the nodes in the graph. By applying this definition to a GTSP graph, a *GTSP subtour* is a cycle which does not contain all of the nodesets of a GTSP graph, where each nodeset is visited through exactly one of its nodes. In a GTSP graph with n > m, a disjointed tour passing through all the m nodesets of the graph by way of m edges and incident to at least m+1 nodes is called a *cleaved tour*. By the same token, a GTSP disjointed subtour with m′ nodesets and m′ edges is called a *cleaved subtour* if the number of visited nodes in the subtour is at least m′+1, where m′ ≤ m-2. Figure 1.2 illustrates GTSP subtours, a cleaved tour, and cleaved subtours of a GTSP graph respectively.

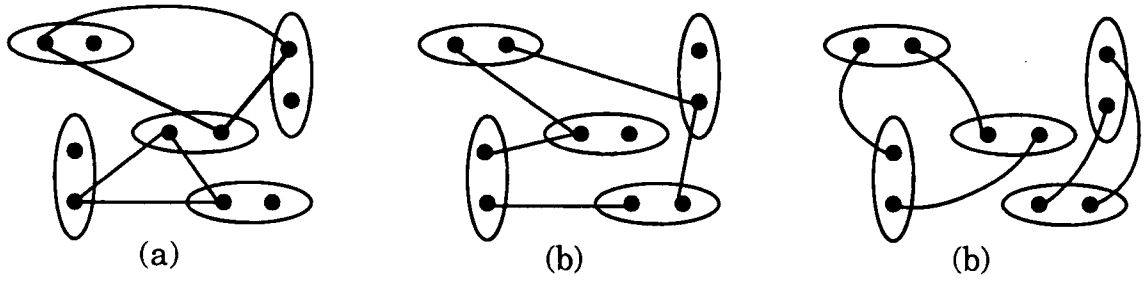


Figure 1.2: (a) GTSP subtours,
(b) A GTSP cleaved tour, (c) GTSP cleaved subtours.

1.2.3 Aggregated Form of a GTSP Graph

Given a GTSP graph $\mathcal{G} = (\mathcal{V},\mathcal{E})$, we define $\mathcal{G}_a = (\mathcal{V}_a,\mathcal{E}_a)$ with $\mathcal{V}_a = \{ I \mid I \in M \}$ and $\mathcal{E}_a = \{ (I;J) \mid I \in \mathcal{V}_a, J \in \mathcal{V}_a, I<J \}$ as the *aggregated form* of $\mathcal{G}$ having the

following properties:

a. Each node in $\mathcal{G}_a$ is a representation of a nodeset in $\mathcal{G}$ and, hence, $|\mathcal{V}_a| = m$.

b. Let S_I and S_J be a pair of nodesets in $\mathcal{G}$, and let I and J be a pair of nodes in $\mathcal{G}_a$ representing nodesets S_I and S_J respectively. Define $E(I;J) = E(S_I \cup S_J)$. Then an edge $(I;J) \in \mathcal{E}_a$, called aggregated edge, corresponds to the edges in E(I;J). Let X_{IJ} and $\overline{X}_{IJ}$ denote the variable and value associated with the aggregated edge (I;J).

c. If there are values corresponding to every edge in $\mathcal{G}$, the value of an edge (I;J) in $\mathcal{G}_a$ is the summation of the value of edges which have been represented by the edge (I;J). Meaning, $\overline{X}_{IJ} = \sum_{(i;j) \in E(I;J)} \overline{x}_{ij}$.

d. The maximum number of edges in $\mathcal{G}_a$ is equal to m(m-1)/2, that is, $|\mathcal{E}_a| \leq m(m-1)/2$.

Figure 1.3 shows an example of a GTSP graph and its aggregated form.

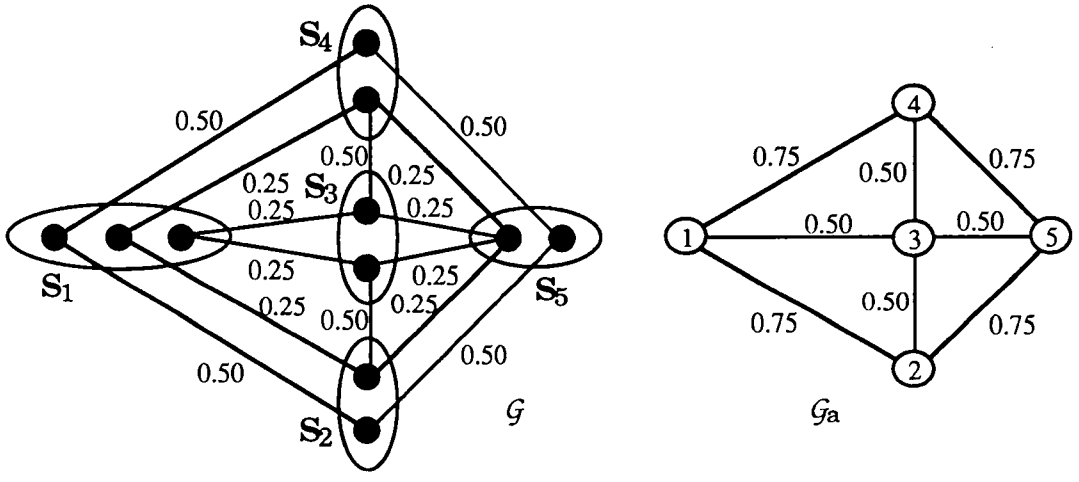
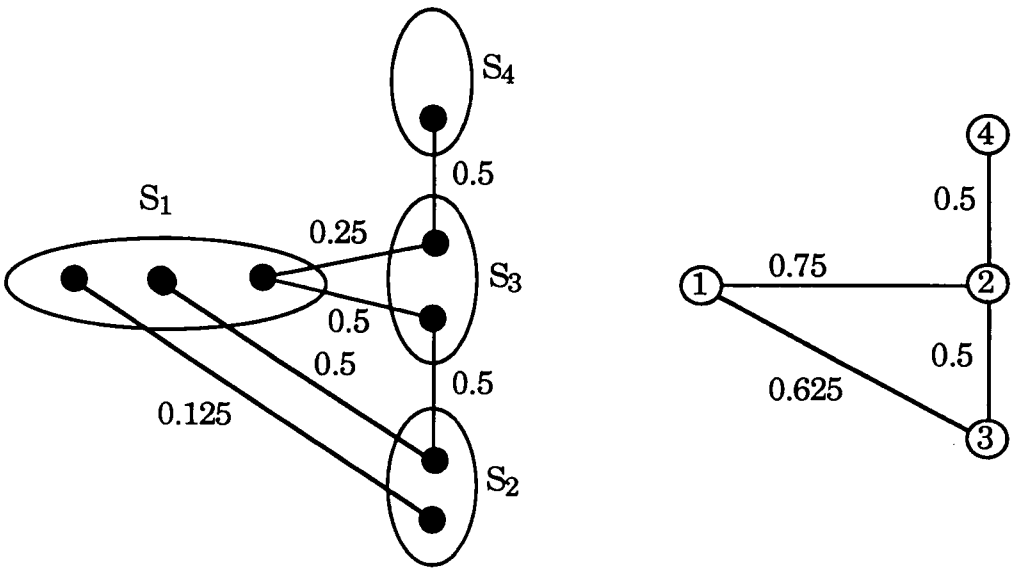


Figure 1.3: An example of a GTSP graph and its aggregated form.

For $W_a \subseteq \mathcal{V}_a$, $E_a(W_a)$ defines the set of aggregated edges with both of their end nodes in W_a, that is $E_a(W_a) = \{ (I;J) \mid (I;J) \in \mathcal{E}_a, I \in W_a, J \in W_a \}$. Given a set of aggregated edges, F_a, we define

$$X(F_a) = \sum_{(I;J) \in F_a} X_{IJ} \text{ and } \overline{X}(F_a) = \sum_{(I;J) \in F_a} \overline{X}_{IJ}.$$

We extend, also, the concept of aggregated form over a partial GTSP graph. For example the aggregated form of the partial GTSP graph shown in Figure 1.4-(a) is depicted in Figure 1.4-(b).



(a) (b)

Figure 1.4: A partial GTSP graph and its aggregated form.

1.3 Canonical and General SGTSP

As mentioned earlier, the canonical SGTSP is defined with mutually exclusive nodesets and no intraset edges. In this section we provide a

formulation for the canonical SGTSP and discuss its relation to a more general form of the SGTSP.

1.3.1 SGTSP Canonical Formulation

Given our previous definition of the canonical SGTSP, we can formulate the problem as an integer programming problem.

Assuming that a SGTSP graph $\mathcal{G} = (\mathcal{V},\mathcal{E})$ contains at least one GTSP tour, the objective is to find a tour of minimum cost. Those edges which constitute an optimal tour obtain the value 1 and the edges not in the tour each obtain value of 0. The objective function is shown by the expression (1.1). The constraint set (1.2), referred to as the *degree constraints*, ensure that each nodeset is incident to exactly two edges in a solution tour. The second constraint set, (1.3), called the *disjoint prevention constraints*, prevents disjointedness in SGTSP tours or subtours. Cleaved tours and cleaved subtours are examples of SGTSP tours and subtours with disjointedness. In this set, $NC_{iJ} = E(i \cup S_J)$ and is called the *node cone* of node i and nodeset J, and $NC'_{iJ} = E(i \cup \mathcal{V} \setminus S_J)$ and is called the *complement of node cone* of node i and nodeset J. In chapter 3, we will elaborate on this class of constraints. The constraint set (1.4), prevents subtours and the constraints are referred as *subtour elimination constraints*. The last two set of constraints enforce the binary nature of the variables.

(P): Minimize $\sum_{(i;j)\in E} c_{ij}x_{ij}$ (1.1)

Subject to,

$x(D(I)) = 2$ $\forall\ I \in M = \{1, 2, \ldots, m\}$ (1.2)

$x(NC'_{iJ}) - x(NC_{iJ}) \geq 0$ for all $i \in \mathcal{V}$, $i \in S_I$, and $J \in M \setminus \{I\}$ (1.3)

$x(E(T)) \leq |S(T)| - 1$ $\forall$ subsets of nodes T, $2 \leq |S(T)| \leq m-2$ (1.4)

$0 \leq x_{ij} \leq 1$ $\forall\ (i;j) \in \mathcal{E}$ (1.5)

x_{ij} integer $\forall\ (i;j) \in \mathcal{E}$. (1.6)

1.3.2 General Form to Canonical Form Transformation

When the SGTSP is defined on a GTSP in canonical form, solution approaches take advantage of characteristics of canonical form. These characteristics are, 1) mutually exclusive and exhaustive nodesets, and 2) absence of intraset edges, and visitation of exactly one node of every nodeset in a tour. Nonetheless, this highly structured form is somewhat limited in that it might exclude certain applications. The goal of this section is to introduce a general form of the SGTSP and discuss its relationship to the canonical form.

The general form SGTSP is defined on a graph $\mathcal{G} = (\mathcal{V},\mathcal{E})$ with edge cost vector c, $-\infty \leq c_{ij} \leq \infty$, for every edge (i;j) in $\mathcal{E}$. The nodes in $\mathcal{V}$ belong to m predefined nodesets $S_1, S_2, \ldots, S_m$, $m \geq 1$, where a node must belong to at least one and at most m-1 nodesets. A feasible solution is a cycle which passes through at least one node of each nodeset, and the number of

solution edges incident to each nodeset is exactly two, that is, the cycle passes through each nodeset exactly once. Figure 1.5 illustrates a general form SGTSP with a feasible cycle.

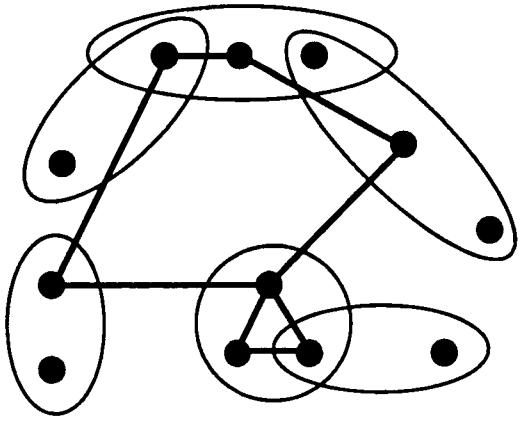


Figure 1.5: A general form SGTSP with a feasible cycle.

As can be seen, the general form SGTSP allows overlapping nodesets and intraset edges.

Given a general form SGTSP on a graph $\mathcal{G} = (\mathcal{V},\mathcal{E})$, we can model it as a general form asymmetric GTSP (AGTSP) on a digraph $\mathcal{D} = (\mathcal{N}, \mathcal{A})$ by replacing each edge (i;j) with a pair of arcs (i,j) and (j,i) in $\mathcal{A}$ with $\vec{c}_{ij} = \vec{c}_{ji} = c_{ij}$. Noon and Bean [1989b] present an efficient method for transforming any general form AGTSP into a canonical form AGTSP. The canonical form AGTSP obtained in this process can be back transformed to a canonical form SGTSP by replacing every pair of arcs of the form (i,j) and (j,i) with an edge (i;j).

1.4 Research Objectives

The first attempt to solve a TSP using a linear description of the TSP polytope was due to Dantzig, Fulkerson, and Johnson [1954]. With the TSP being perhaps the most prominent of the unsolved combinatorial optimization problems, few attempts have being made to tackle its generalized form, the GTSP. Although the initial appearance of the GTSP in OR/MS literature occurred in the late sixties, no attempt has been made to tackle the GTSP from its linear description point of view.

The primary goal of this research is to define classes of valid inequalities for the SGTSP, and to develop methods for their identifications. Ultimately, we aim to test the effectiveness of these inequalities from a computational point of view.

In order to achieve the aforementioned objectives, the three following tasks constitute the outline of the research:

1- Analytical description of new classes of valid inequalities.
2- Development of a linear programming (LP) based cutting plane procedure for the SGTSP.
3- Empirical investigation of final bound quality on the test problems.

1.5 Organization of Dissertation

In Chapter 2, we review the literature relevant to this research and present several GTSP applications. Chapter 3 details four classes of valid inequalities for the symmetric GTSP. Chapter 4 presents algorithms and routines for violated constraints identification, heuristic methods for

finding upper bounds, and the specifics of a cutting plane procedure. The chapter also examines the computational performance of the procedure. In the final chapter, we discuss conclusions and directions for future research.

CHAPTER 2

DISCUSSION OF LITERATURE AND APPLICATIONS OF GTSP

The focus of this chapter is to discuss the GTSP literature and applications. We first discuss the background of the GTSP from the viewpoint of literature. Then, we preview and introduce a number of actual and potential applications of the GTSP.

2.1 Literature Review

The first published definition of the symmetric generalized traveling salesman problem is by Srivastava, et al. [1969]; and the first application is due to Saksena [1970] who used the GTSP concept to solve the problem of scheduling clients through welfare agencies and called this application as "an extension of traveling salesman problem". Both of the above authors proposed a dynamic programming approach analogous to that of Gonzalez [1962] for the TSP. Another author who has applied a dynamic programming approach to solve a GTSP application is Henry-Labordere [1969]. This author presents an application of the asymmetric GTSP called

the record balancing problem. The goal of the problem is to find the optimal sequence of record fields organized in a file for indexed access.

As is generally the case with dynamic programming, the practical limit on the size of the largest problem that can be solved is likely due to storage space rather than the amount of computation. For example, in the case of an asymmetric TSP with 20 cities, approximately 925,000 storage locations are necessary to store the ninth step of the dynamic programming, (see Dreyfus and Law, [1977]). This situation is worse in a SGTSP instance with 20 sets of 10 cities each, namely over 10^{13} states are needed at the tenth step of the dynamic programming, (see Noon [1988]).

In a paper by Laporte and Nobert [1983], the authors present an integer programming formulation similar to that given in Chapter 1, except for the inclusion of fixed node costs with additional corresponding node variables. Although these node costs could have been embedded into the edge costs, they were kept separately for the purpose of studying their computational impacts. The solution approach adapted to solve this integer programming formulation takes advantage of a Land and Doig [1960] type branch-and-bound scheme with subtour elimination constraints introduced as needed. Computational results showed that problems of intermediate size having up to 50 nodes ($n = 50$) and at most 11 nodesets ($6 \leq m \leq 11$, and $|S_1| = 1$) could be solved in less than 600 seconds of mainframe CPU time.

Laporte, Mercure, and Nobert [1985] formulated and presented a branch-and-bound approach for the AGTSP. One version of the branch-and-bound algorithm makes use of a network flow routine for solving a relaxation of the problem; other versions of the algorithm employ Lagrangean relaxation. Computational results show that problems with

up to 40 nodes and 11 nodesets could be solved successfully. In another paper, Laporte, Mercure, and Nobert [1987] again considered the AGTSP. In this article, the problem is formulated as an integer linear program. Then the program is relaxed and solved by a modified version of the Carpaneto and Toth [1980] branch-and-bound algorithm for the asymmetric TSP. In the paper, the authors reported optimally solving problems with up to 100 nodes and 8 nodesets.

An extensive study of the AGTSP is given in Noon [1988]. This research aimed to describe the AGTSP modeling applications and to present both optimal and heuristic solution approaches. One of the highlights of this study is the process of transforming any problem in the general form to a problem in the canonical form. In addition, a two phase approach for determining an optimal solution to a problem in the canonical form is presented. In the first phase a lower bound is obtained by solving an assignment problem, and calculating upper bound using one of the proposed heuristics. A procedure makes use of this bound to reduce the size of a problem. In the second phase of the approach an efficient branch-and-bound procedure is used to solve the problem to optimality. The computational results show success on a wide range of problems with up to 300 nodes configured as 50 nodesets with 6 nodes per set.

Noon and Bean [1989a] have described the approach presented in Noon [1988] more rigorously. The authors showed that a combined approach of bounding, elimination, and enumeration is a practical method for solving the AGTSP. In a computational comparison, the authors showed that their approach was superior to the approach used in Laporte, Mercure, and Nobert [1987].

The transformation of a general form GTSP into an asymmetric TSP problem is the subject of another paper by Noon and Bean [1989b]. In this paper, the authors describe as a class of problems that can be formulated as a GTSP's and show how any problem in this class can be transformed into a standard TSP.

2.2 Applications

The GTSP enjoys a number of actual and potential applications. In general, the GTSP is useful for modeling problems which involve simultaneous decisions of selection and sequence. Several such problems are discussed in this section.

2.2.1 Routing Through Alternative Locations

The problem of optimally scheduling clients of a community clinic through a set of welfare agencies is the subject of a paper by Saksena [1970]. A client has a list of m needs which can be satisfied by a set of n agencies, n > m. The agencies have separate locations and each agency is capable of satisfying some but not all of the client's needs. The problem is to schedule each client through a number of agencies to meet his/her needs with the objective of minimizing the travel and service costs. This problem can be easily modeled as a GTSP where each nodeset contains nodes which correspond to the location of agencies that service a particular need. The number of nodesets is equal to the number of distinct needs of the client.

Noon [1988] has called this problem a "problem of routing through alternative locations".

The problem of optimally sequencing record fields organized in a file for direct access is a GTSP application discussed in Henry-Labordere [1969]. Kershenbaum et al. [1976] described the telephone network routing as an application of the GTSP. The travel agent problem presented in Laporte and Nobert [1983] is similar to the welfare agencies problem. In this instance a travel agent wants to organize tours passing through different cities with tourism attractions. Each tour must be coordinated in such a way that travelers are gratified by visiting each type of attractions once at the lowest possible total cost.

2.2.2 Postal Activities Applications

Two applications related to postal activities are the problem of locating post boxes in an urban environment, and the routing and scheduling of postal vans. The first problem is presented by Bovet [1983] and is based on statistical analyses of mail volume. In this problem n-1 potential locations of post boxes in m-1 different urban ares (n > m) are considered. Consider an extra nodeset m that has one node and corresponds to the sorting office. Each of the m-1 other urban areas corresponds to a nodeset containing nodes that represent potential location of post boxes. The problem is a GTSP with n nodes and m nodesets. A GTSP solution of the problem provides the individual post box location for each of the m-1 urban areas.

The second postal application has initially presented by Rousseau in January 1985 (see Laporte et al. [1985] and [1987]). The application is later

discussed and called the "Postal Delivery Problem" by its author in Rousseau [1988]. The problem is concerned with picking up the contents of a large number of mail boxes located in a relatively small urban region. The collection of mail is performed by a postal van driver who is constrained by union requirements. To comply with union demands, the driver must pick up the mail from the right hand side of the van and not cross any street on foot. With this regulation, the mail boxes located at the intersections of streets create less flexibility for the driver. To aid in our discussion, Figure 2.1-(a) depicts a mail box located at an intersection of two one-way streets.

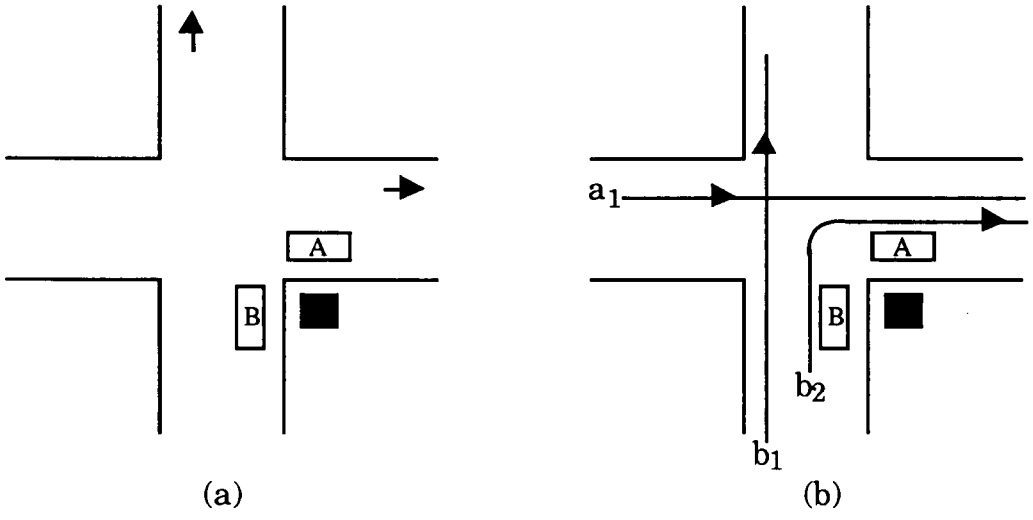


Figure 2.1: An example of postal delivery problem.

The driver must pick up mail from one of the two possible lanes adjacent to the mail box, position A or position B. When the driver goes in the direction of a_1, he picks up the mail at position A, and while going in the direction of b_1 or b_2 he picks up mail at position B (see Figure 2.1-(b)). The driver, in view of the above facts, can access a post box located at a corner in

two distinct directions. The distinct directions have different distances to the next post box. This problem is modeled as an AGTSP, where each post box is represented by a nodeset having two nodes. The two nodes represent the two potential positions from which the mail can be picked up.

2.2.3 Order Picking Problem

The order picking problem is one of the fundamental problems associated with automated storage/retrieval systems (AS/RS), material handling transporters in automated manufacturing, and automated guided vehicles (AGV). Among problems that can be modeled as GTSP are some special cases of the order picking problem.

2.2.3.1 Multiple Stock Locations

A special case of the order picking problem is when items have multiple stock locations (see Noon [1988], and Noon and Bean [1989b]). This problem can be viewed as a problem of routing through alternative locations. The problem can be modeled as a GTSP where a nodeset is associated with each ordered item and contains nodes which correspond to the item's different stock locations. It should be pointed out that the single stock location is a classic application of the TSP.

2.2.3.2 Clustered TSP

Another GTSP application considered in Noon [1988] is the clustered traveling salesman problem. This problem appears as an order picking problem when many orders (clusters) are placed and the assumption is

that only one order can be picked at a time. A sequence in which to pick up the orders and the items within each order is a solution to the problem. A method suggested by Noon [1988] to model the problem as a GTSP is given below.

Let an order for several items be represented by a nodeset and let each node in the nodeset correspond to an ordered pair (i,j) of items i and j that are in the order. Let the cost associated with each node (i,j) be the cost of the minimum intracluster Hamiltonian path that begins from i and ends at j. The arc cost associated with any interset arc from node $(i,j) \in S_I$ to node $(k,l) \in S_J$ reflects the cost from item j to item k. Selecting a node (i,j) from a nodeset S_I is interpreted as picking up the items of the order I beginning with item i and ending with item j.

2.2.3.3 Carousel Conveyor Problem

The carousel conveyor problem is an application of GTSP to the order picking problem. A carousel conveyor is a rotatable round shelf that turns in either direction under computer control. A carousel conveyor is a useful tool for storing and retrieval of small items in automated warehouses. The main property of a carousel conveyor is that rather than having its picker retrieve an item, the item can travel to the picker. Figure 2.2 depicts a representation of a carousel conveyor. In Figure 2.2, storage locations of items are shown by small circles around the conveyor. The shaded circles in the figure are the storage locations of items ordered within an arbitrary order.

Bartholdi and Platzman [1986] studied the carousel conveyor problem and suggested an optimal algorithm for a single order. The algorithm

requires on the order of n steps beyond a preprocessing sort. In addition, six heuristic algorithms for solving single and multiple orders are discussed by the authors.

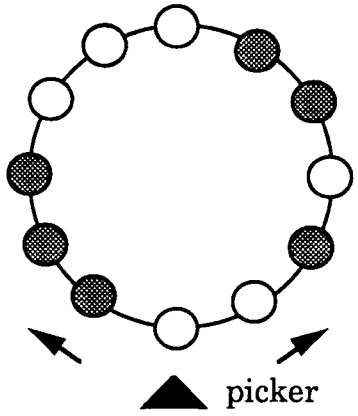


Figure 2.2: A representation of a carousel conveyor.

A method similar to that of presented in Noon [1988] for clustered TSP can be adapted to solve the carousel conveyor problem. In the case of multiple order, each order is modeled as a nodeset containing $n_I(n_I\text{-}1)$ nodes where n_I is the number of items ordered within the Ith order. Each node represents an ordered pair (i,j) in the same fashion as was in section 2.2.3.2.

2.2.4 Covering Salesman Problem

We say that a set of nodes is *covered* by a node i, if the nodes in the set are located within a prespecified distance S of the node i. With the above definition, the covering salesman problem (CSP) is the problem of determining a minimum cost tour over a subset of n given nodes in such a way that all n nodes be covered by the nodes in the subset. The CSP is

studied by Current and Schilling [1989]. The delivery of overnight mail and the routing of health care delivery teams in developing countries are potential applications of the CSP that are discussed by the authors. Noon and Bean [1989b], however, modeled the CSP as a GTSP application. The proposed method is to replace each arc cost $\vec{c}_{ij}$ with the shortest path cost from node i to node j. Then, every node and its covered nodes within the radius of S constitute a nodeset. In other words, a nodeset S_I represents the set of nodes that are covered by node i with respect to a radius S, $S_I = \{ j \mid \vec{c}_{ij} \leq S \} \cup \{i\}$. It should be pointed out that the result is a general form GTSP.

2.2.5 Traveling Salesman Problem with Time Windows

Consider a salesman who must visit a number of dispersed customers exactly once. Each customer must be visited and served fully within a specified time interval known as *time window*. The traveling salesman problem with time windows (TSPTW) is a problem of sequencing the customers to be visited and selecting a specified time interval to visit each customer so that the total distance traveled in his tour is minimized. We assume that no violation of the time windows constraints is allowed.

The use of a state space relaxation for finding optimal solution to the TSPTW is discussed in Christofides, Mingozzi, and Toth [1981]. We propose an application of the AGTSP to this problem. The use of the AGTSP to the problem is based on the assumption that time windows are discrete. We designate one nodeset to each customer. Every node within a nodeset defines a small time interval. Thus, a wide time window has to be partitioned into several smaller time windows based on the minimum

possible time interval. For example, suppose every node represents a one hour time interval, and time intervals 8-9, 9-10, ..., 3-4 in a working day from 8 a.m. to 4 p.m. are represented by nodes that are indexed from 1 to 8. A customer I with time windows 9-12, and 2-3 is represented by a nodeset S_I with nodes i_2, i_3, i_4, and i_7. An interset arc $(i_p;j_q)$ joins the node $i_p \in S_I$ and $j_q \in S_J$ if $p < q$, that is, the node i_p is precedence compatible to the node j_q.

2.2.6 Vehicle Routing Problem

The above applications suggest that the GTSP is a useful model for problems involving decisions of selection and sequencing. A significant problem which incorporates the elements of selection and sequencing is the vehicle routing problem (VRP).

The VRP is the problem of finding a minimum cost set of vehicle routes for a given vehicle fleet. The fleet is stationed at a central depot and delivers goods from the depot to a group of customers. In Figure 2.3 a set of vehicles routes for a VRP with 3 vehicles and 8 customers is depicted.

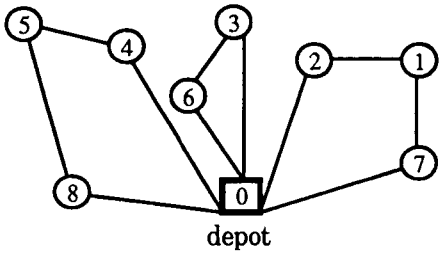


Figure 2.3: A set of vehicles routes for a VRP with 3 vehicles and 8 customers.

If vehicles are constrained by their capacities, the problem is referred to as a *capacitated VRP.*

2.2.6.1 Modeling the VRP as a SGTSP with Additional Constraints

We consider a capacitated VRP with k vehicles stationed in a depot and serving m customers geographically scattered in an area. Let $q_I \geq 0$ be the demand associated with customer I, for $I \in M = \{ 1, 2, ..., m \}$. Define Q_r as capacity of vehicle r for $r \in T = \{ 1, 2, ..., k \}$. It is assumed that $q_I \leq Q_r$, for all I and r. Let d^r_{IJ}, $I < J$, be the symmetric travel cost for vehicle r to travel between customers I and J for all $I,J \in M$. Let d^r_{0J} be the travel cost for vehicle r to travel between the depot and customer J, $J \in M$.

This problem can be modeled as a canonical SGTSP with some additional constraints. Each customer is represented by a nodeset. The model has m nodesets, km nodes and the following characteristics: m nodesets are indexed from 1 to m, with $|S_I| = k$ for all $I \in M$. We denote node $I_r \in S_I$ to represent customer I being visited by vehicle r. The nodesets then are given as $S_1 = \{ 1_1, 1_2, ..., 1_k \}$, $S_2 = \{ 2_1, 2_2, ..., 2_k \}$, and so forth. An edge $(I_r;J_r)$ corresponds to vehicle r traveling between customer I and J and has edge cost $c_{I_rJ_r}$. The edge cost associated with edge $(I_r;J_r)$ is equal to d^r_{IJ} for all I, J and r. An edge $(I_r;J_s)$, $r \neq s$, corresponds to the combined activities of vehicle r concluding its route at the depot after visiting customer I and vehicle s starting its route from the depot with a visit to customer J. Thus $c_{I_rJ_s}$, the edge cost associated with the edge $(I_r;J_s)$, $r \neq s$, is calculated by adding the traveling cost for vehicle r from customer I to the depot and the traveling cost for vehicle s from the depot to customer J. Therefore, $c_{I_rJ_s} = d^r_{0I} + d^s_{0J}$, $r \neq s$, for all customers $I,J \in M$, and all vehicles

r,s∈T. We denote the resulting GTSP graph as $\mathcal{G}_v = (\mathcal{V}_v, \mathcal{E}_v)$ with $\mathcal{V}_v = \bigcup_{I \in M} S_I$, $\mathcal{E}_v = \{ (I_r;J_s) \mid I_r \in \mathcal{V}_v, J_s \in \mathcal{V}_v, I < J, r \text{ and } s \in T \}$, and the edge cost vector $c_v = \{ c_{I_rJ_s} \mid (I_r;J_s) \in \mathcal{E}_v \}$.

2.2.6.2 Formulation

To ensure that a feasible solution of the above constructed SGTSP is a VRP feasible solution, we must add two new sets of constraints to the canonical formulation of the problem. The first set guarantees that every vehicle departs from the depot and returns exactly once, and as a consequence guaranteeing there are k vehicle routes. This set can be formulated as the following constraints:

$$\sum_{\substack{(I_r;J_s) \in \mathcal{E}_v \\ r \neq s}} x_{I_rJ_s} = 2 \text{ for every } r \in T \qquad (2.1)$$

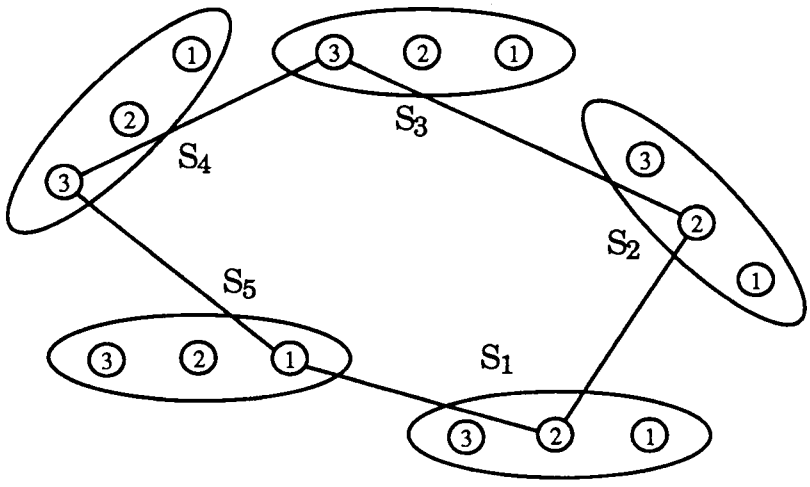
The second set of constraints is associated with maintaining feasibility with respect to vehicle capacity Q_r for r∈T. These constraints are as follows:

$$\sum_{\substack{(I_r;J_r) \in \mathcal{E}_v \\ I, J \in M, I < J}} (q_I + q_J) x_{I_rJ_r} + \sum_{\substack{(I_r;J_s) \in \mathcal{E}_v \\ r \neq s,\ I \in M}} q_I x_{I_rJ_s} \leq 2Q_r \text{ for every } r \in T. \qquad (2.2)$$

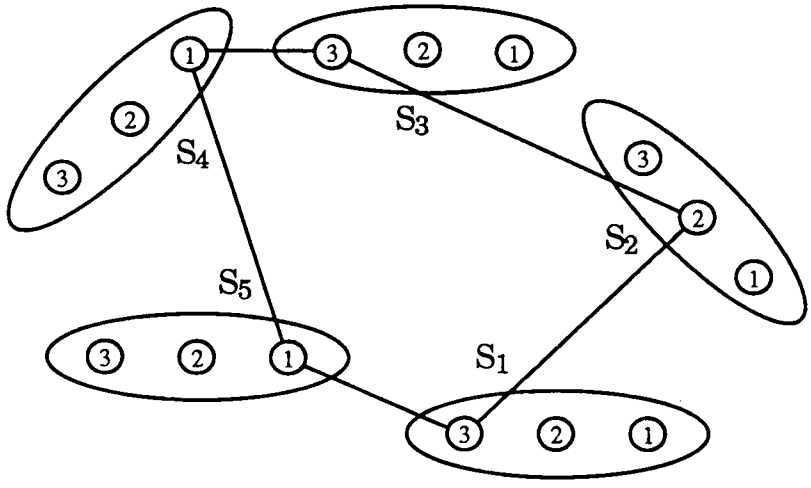
The two constraint sets, (2.1) and (2.2), and any other problem specific constraints can be added to the canonical formulation of SGTSP as side constraints.

2.2.6.3 Example

Figure 2.4 shows two SGTSP solutions for a VRP with 5 customers and 3 vehicles.



(a)



(b)

Figure 2.4: Two SGTSP solutions for a VRP.

Suppose that both solutions, the SGTSP tours in Figure 2.4, have satisfied the vehicle capacity constraints, (2.2). It is clear that the solution given in Figure 2.4-(b) is not feasible with respect to the constraint set (2.1). To clarify the claim, we make use of Figure 2.5. Figure 2.5 (a) and (b) depict the vehicle routes corresponding to the solutions given in Figure 2.4 (a) and (b), respectively.

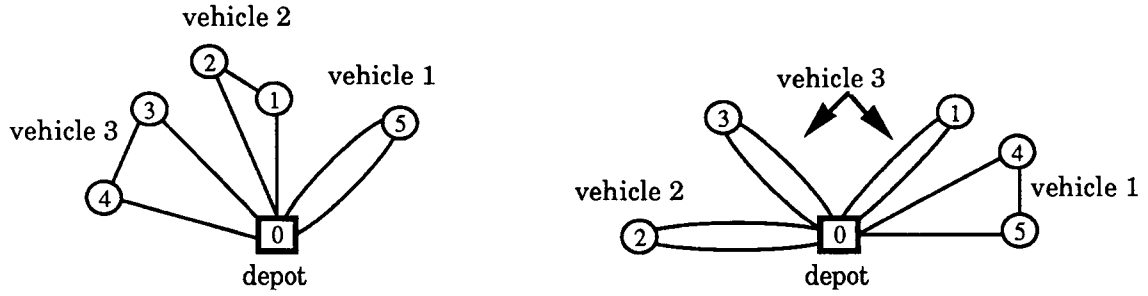


Figure 2.5: Vehicle routes corresponding to the solutions given in Figure 2.4.

It is evident that the vehicle routes given in Figure 2.5-(b) are not valid, since vehicle 3 has been routed twice.

CHAPTER 3

VALID INEQUALITIES FOR THE SYMMETRIC GTSP

In this chapter, we investigate four classes of valid inequalities for the SGTSP. 1) disjoint prevention constraints, 2) generalized TSP valid inequalities, 3) kite inequalities, and 4) inversely transformed TSP inequalities. Each class is described analytically, and its mathematical substantiation is provided.

Through the use of valid inequalities, we are able to construct progressively tighter linear programming (LP) descriptions of the problem. The LP descriptions of the problem are generated within an iterative process called a cutting plane procedure whose goal is to solve the integer problem (P) with an LP solver. In Chapter 4, we will discuss a cutting plane procedure which relies on the contents presented in this chapter.

3.1 Disjoint Prevention Constraints

In this section we investigate a class of SGTSP valid inequalities known as *disjoint prevention* (DP) constraints. We describe this class of

inequalities and provide proofs of their validity.

3.1.1 Description of the Disjoint Prevention Constraints

The name "disjoint prevention" for this class of inequalities comes from the fact that they prevent, to a certain degree, the disjointedness which may occur in a solution of the LP problem (P_1) defined below:

$$(P_1): \quad \text{Minimize} \sum_{(i;j)\in E} c_{ij}x_{ij} \tag{3.1}$$

Subject to,

$$x(D(I)) = 2 \qquad \forall\ I \in M = \{1, 2, \ldots, m\} \tag{3.2}$$

None or some valid (3.3)

SGTSP inequalities

$$0 \le x_{ij} \le 1 \qquad \forall\ (i;j) \in \mathcal{E} \tag{3.4}$$

The problem (P_1) is a general form of any LP-based SGTSP problem that can be generated for the SGTSP. We refer to the optimal solution of this formulation as the P_1 solution of its corresponding problem.

If we consider a node as a mechanism for joining two edges in a SGTSP tour, *disjointedness* refers to the case where a tour enters a nodeset through one node and leaves the nodeset from another node. Cleaved tours or cleaved subtours provide examples of disjointedness. Figure 1.2, in Chapter 1, shows some example of cleaved tour and cleaved subtours. Preventing *imbalance* among the values of edges that are incident to a node is another aspect of DP constraints. By imbalance we mean that DP

constraints associated with some of the edges incident to the node are violated. We will discuss the two characteristics, disjointedness and imbalance, in a later part of this section.

3.1.2 Mathematical Expression and Substantiation

A disjoint prevention constraint in a GTSP graph $\mathcal{G} = (\mathcal{V},\mathcal{E})$ corresponds to an edge and one of the two nodes associated with the edge. For a given edge $(i;j) \in \mathcal{E}$, at most two DP constraints can be identified, one associated with node i and the other corresponding to node j. Consider two nodesets S_I and S_J and let an edge e_{ij} be the edge connecting the pair of nodes $i \in S_I$ and $j \in S_J$. Let

$$b_i = \{ (i;k) \mid k \in S_J \setminus \{j\} \},$$
$$a_i = d(i) \setminus (b_i \cup e_{ij}),$$
$$b_j = \{ (j;k) \mid k \in S_I \setminus \{i\} \},$$
$$a_j = d(j) \setminus (b_j \cup e_{ij}).$$

Figure 3.1 shows the above definitions for the edge e_{ij}.

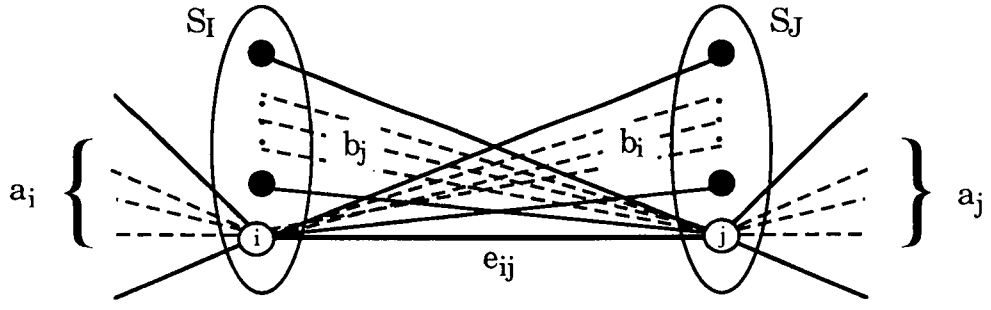


Figure 3.1: Pictorial definitions of terms used in the DP constraints.

For a given edge e_{ij}, we define a pair of constraints shown below as the DP constraints for the edge e_{ij}, where each constraint is associated with one end node of the edge.

$$x(a_i) - x(b_i) - x_{ij} \geq 0, \qquad (3.5\text{-a})$$

$$x(a_j) - x(b_j) - x_{ij} \geq 0. \qquad (3.5\text{-b})$$

The constraint (3.5-a) corresponds to end node i of e_{ij}, and (3.5-b) corresponds to end node j of e_{ij}.

Let A_{iJ} be the set of edges emanating from a nodeset S_J and having a node $i \notin S_J$ as a common node. We can show that all edges in A_{iJ} have identical DP constraints of the form (3.5-a). Consider two nodesets S_I and S_J, and let i be a distinct node in S_I. Then A_{iJ} can be stated as: $A_{iJ} = \{ (i;k) \mid$ i is distinct and $i \in S_I$, $k \in S_J \}$. Let $j \in S_J$ and $(i;j) \in A_{iJ}$, then by definition of $b_i = \{ (i;k) \mid k \in J \setminus \{j\} \}$ we have

$$A_{iJ} = b_i \cup e_{ij}. \qquad (3.6)$$

The DP constraint corresponding to end node i of the edge e_{ij} is

$$x(a_i) - x(b_i) - x_{ij} \geq 0,$$

and can be restated as

$$x(a_i) - x(b_i \cup e_{ij}) \geq 0. \qquad (3.7)$$

From (3.6) and (3.7) we get

$$x(a_i) - x(A_{iJ}) \geq 0. \qquad (3.8)$$

Therefore for any edge in A_{iJ}, (3.8) is the DP constraint corresponding to end node i of the edge.

Let S_I and S_J be two nodesets of a complete GTSP graph and i be a distinct node in S_I. By the previous argument, only one DP constraint is required to cover all edges in A_{iJ}, where $A_{iJ} = \{ (i;k) \mid$ i is distinct and $i \in S_I$,

$k \in S_J$ }. Since there exist (m-1) nodesets besides S_I, (m-1) DP constraints are required to cover all edges in A_i, where

$$A_i = \bigcup_{J \in M \setminus \{I\}} A_{iJ}.$$

Therefore, for any nodeset, say S_K, with $|S_K|$ nodes, $|S_K|(m-1)$ DP constraints are needed. Consequently, summation over all nodesets gives the desired result. That is,

$$\sum_{K=1}^{m} |S_K|(m-1) = (|S_1| + |S_2| + \ldots + |S_m|)(m-1).$$

We know that $n = (|S_1| + |S_2| + \ldots + |S_m|)$, hence,

$$\sum_{K=1}^{m} |S_K|(m-1) = n(m-1).$$

Therefore for a complete GTSP graph with n nodes and m nodesets, the maximum number of DP constraints is n(m-1).

In an integer programming formulation for the SGTSP, Noon [1988] presented a class of constraints similar to the DP constraints. These constraints are given below.

$$\left.\begin{array}{ll} \sum_{i<j} x_{ij} + \sum_{j<l,k\neq l} x_{jl} - x_{jk} \geq 0 & (3.9\text{-a}) \\ \sum_{i<k,i\neq j} x_{ik} + \sum_{k<l} x_{kl} - x_{jk} \geq 0 & (3.9\text{-b}) \end{array}\right\} \text{for } (j;k) \in \mathcal{E},\ j<k.$$

The translation of (3.9-a) and (3.9-b) to our notation gives (3.10-a) and (3.10-b) respectively,

$$\left.\begin{array}{ll} x(a_j) + x(b_j) - x_{jk} \geq 0 & (3.10\text{-a}) \\ x(a_k) + x(b_k) - x_{jk} \geq 0 & (3.10\text{-b}) \end{array}\right\} \text{for every } (j;k) \in \mathcal{E},$$

and the maximum number of constraints required is $2|\mathcal{E}|$. For example, for a GTSP graph with 5 nodesets and 3 nodes in each one, the maximum number of DP constraints is 60; for the formulation of the forms (3.10-a) and (3.10-b), 180 DP constraints are the maximum requirement. It should be

noted that the DP constraints, in essence, are stronger than the ones given in (3.10-a) and (3.10-b)

3.1.3 Characteristics of DP Constraints

Previously, we mentioned that the main aspects of the DP constraints are that they prevent disjointedness among the edges and imbalance among the values of edges that are incident to a node. In this section we discuss these issues in depth.

These aspects are interrelated in that by preventing disjointedness of edges at a node, we are promoting balance at the node with respect to edge value. Consider a segment of a cleaved tour or subtour in Figure 3.2, where $x_{ij} = 1$ and $x_{kl} = 1$.

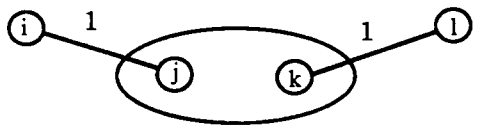


Figure 3.2: An example of disjointedness.

In this example, neither node j nor node k joins two edges with positive values. The DP constraint for the edge (i;j) from the end node j is violated, with $x(a_j) - x(b_j) - x_{ij} \not\geq 0$, since $\overline{x}(a_j) = 0$, $\overline{x}(b_j) = 0$, and $\overline{x}_{ij} = 1$. The same is true for the edge (k;l) from the end node k.

To discuss the aspect of establishing balance by DP constraint, we use the example given in Figure 3.3, where balance at the node j does not exist.

Assuming that the nodes i and l are not members of the same nodeset, consider the DP constraint associated with end node j of the edge (i;j). We have $x(a_j) - x(b_j) - x_{ij} \not\geq 0$, since $\bar{x}(a_j) = 0.5$, $\bar{x}(b_j) = 0$, and $\bar{x}_{ij} = 1$.

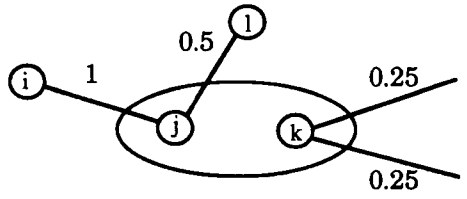


Figure 3.3: An example with imbalance.

Although no disjointedness occurs at the node j, the constraint is violated. This is due to the imbalance in values of edges that are incident to the node j. Another example, shown in Figure 3.4, gives imbalance in values of edges incident to the node i.

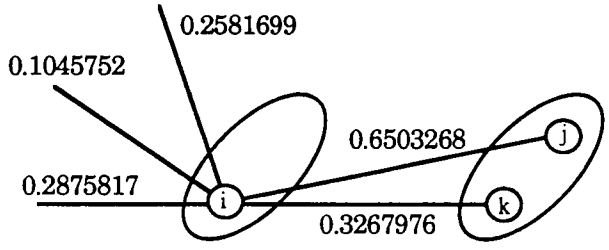


Figure 3.4: Another example with imbalance.

In this example, the DP constraint for end node i of the edge (i;j) is violated,

$$x(a_j) - x(b_j) - x_{ij} \not\geq 0,$$

$$(0.2875817 + 0.1045752 + 0.2581699) - 0.3267976 - 0.6503268 \not\geq 0.$$

Note that the DP constraint associated with end node i of the edge (i;k) is identical to that of the edge (i;j). The underlying reason is that both nodes j and k are in a same nodeset and the node i is in common for both edges.

3.1.4 A Simplified Form of DP Constraints and its Verification

Previously, we defined A_{iJ} as the set of all edges emanating from a nodeset S_J and incident to a node $i \notin S_J$. In addition, a_i can be restated as $a_i = d(i) \backslash A_{iJ}$. In order to come up with a better way of representing DP constraints, we refer to A_{iJ} as the *Node Cone* iJ, and a_i as the *Complement of Node Cone* iJ.

Node Cone iJ, denoted by NC_{iJ}, is the set of all edges each incident to a distinct node $i \in S_I$ from one end, and incident to a node in a nodeset S_J from the other end. Complement of Node Cone iJ denoted by NC'_{iJ} is the set of all edges incident to the node $i \in S_I$ except the edges in NC_{iJ}. That is,

$$NC_{iJ} = \{ (i;j) \in d(i) \mid j \in S_J \} = A_{iJ},$$

$$NC'_{iJ} = \{ (i;j) \in d(i) \mid j \notin S_J \} = a_i,$$

$$NC_{iJ} \cup NC'_{iJ} = A_{iJ} \cup a_i = d(i).$$

Figure 3.5 depicts these definitions.

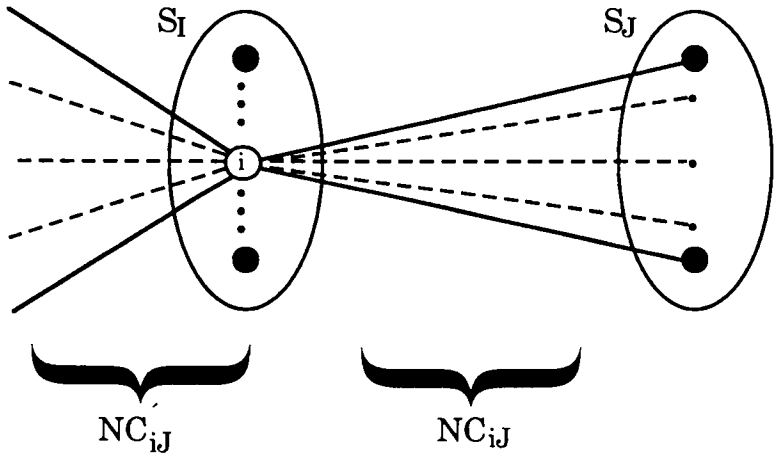


Figure 3.5: A node cone and its complement.

Using the above notation, we can relate (3.8) to the general definition of the DP constraints,

$$x(NC'_{iJ}) - x(NC_{iJ}) \geq 0 \quad \forall\, i \in \mathcal{V},\, i \in S_I,\, \forall\, J \in M \setminus \{I\}. \qquad (3.11)$$

The following theorem proves that any disjoint prevention constraint is a valid inequality for the symmetric GTSP.

Theorem 3.1. For a given node i and a given nodeset S_J, $i \notin S_J$, the constraint $x(NC'_{iJ}) - x(NC_{iJ}) \geq 0$ is valid for the symmetric GTSP.

Proof. There are only two possible cases for node i. Either a) node i is not included in a feasible GTSP tour, or b) node i is included in a feasible GTSP tour.

Case a:

If node i is not in a tour, $\bar{x}_{ij} = 0$ for all $(i;j) \in d(i)$ and the inequality holds.

Case b:

If node i is included in a feasible tour yet no edge connects it to a node in S_J, then $\bar{x}_{ij} = 0$ for all $j \in S_J$. Thus $\bar{x}(NC_{iJ}) = 0$, and $\bar{x}(NC'_{iJ}) = 2$. Therefore the constraint is valid. If node i is is included in a feasible tour and there is an edge connecting the node i to a node in S_J, $\bar{x}(NC'_{iJ}) = 1$, and $\bar{x}(NC_{iJ}) = 1$ and the inequality holds. This establishes the proof. ■

3.2 Generalization of TSP Valid Inequalities

In this section we introduce the second class of valid inequalities for the symmetric GTSP. We establish this class based on the fact that any constraint that is valid for an m-city STSP can be generalized to a valid inequality for an m-nodeset SGTSP. Whereas the class of DP constraints

contained at most n(m-1) constraints, the class we now investigate has an exponential number of constraints.

3.2.1 Description of the Inequalities

In the first chapter we defined $\mathcal{G}_a = (\mathcal{V}_a, \mathcal{E}_a)$ as the aggregated form of a GTSP graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. Each node I of $\mathcal{G}_a$ corresponds to a nodeset S_I of $\mathcal{G}$, and each edge (I;J) of $\mathcal{G}_a$ represents all edges between two nodesets S_I and S_J of $\mathcal{G}$. The variable corresponding to an edge (I;J) between nodes I and J in $\mathcal{V}_a$, denoted by X_{IJ}, is equal to the summation of the variable of all edges between nodesets S_I, and S_J, that is,

$$X_{IJ} = \sum_{(i;j) \in E(I;J)} x_{ij},$$

and consequently,

$$\overline{X}_{IJ} = \sum_{(i;j) \in E(I;J)} \overline{x}_{ij}. \qquad (3.12)$$

Figure 3.6 shows an example of a GTSP graph and its aggregated form. The edges shown on both graphs are those with nonzero values. The edge value of each edge is given beside the edge.

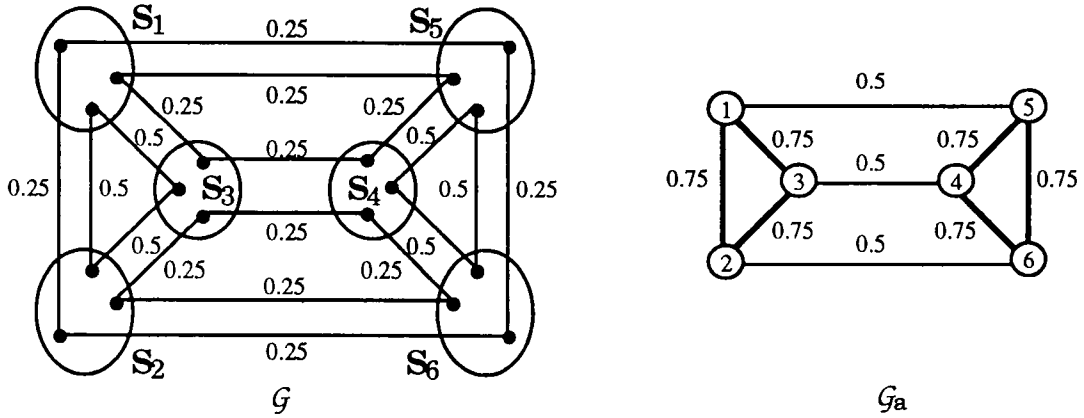


Figure 3.6: A GTSP graph and its aggregated form.

3.2.2 Finding Valid Inequalities Through Aggregated Form of GTSP

For a given solution $\bar{x} = (\bar{x}_{ij})$ of a SGTSP with graph $\mathcal{G}$, we first use (3.12) to calculate its aggregated solution, $\overline{X} = (\overline{X}_{IJ})$, of a STSP with graph $\mathcal{G}_a$. With $\overline{X}$ as a solution of the STSP over $\mathcal{G}_a$, any method for identifying violated valid inequalities for the STSP can be applied. We can then derive a violated inequality of SGTSP if a violated inequality for STSP is identified. This is done by simply substituting equivalent GTSP edges for every edge in a valid inequality over $\mathcal{G}_a$. That is, for any edge $(I;J) \in \mathcal{E}_a$, all edges in E(I;J) are being substituted if the edge (I;J) is in a valid inequality for the STSP over $\mathcal{G}_a$. The resulting valid inequality after is a valid inequality (VI) for the SGTSP. Figure 3.7 shows a flow diagram of the above process.

For instance, two valid inequalities can be identified over the aggregated form of the GTSP graph shown in Figure 3.6. These inequalities which are shown below, are subtour elimination constraints over the nodes 1, 2, 3, and 4, 5, 6 in $\mathcal{G}_a$.

$$X_{12} + X_{13} + X_{23} \leq 2,$$

$$X_{45} + X_{46} + X_{56} \leq 2.$$

The following valid inequalities over the given GTSP graph are derived from the above inequalities, respectively.

$$x_{15} + x_{24} + x_{28} + x_{37} + x_{48} + x_{69} \leq 2,$$

$$x_{10,14} + x_{11,15} + x_{11,16} + x_{12,17} + x_{13,18} + x_{15,16} \leq 2.$$

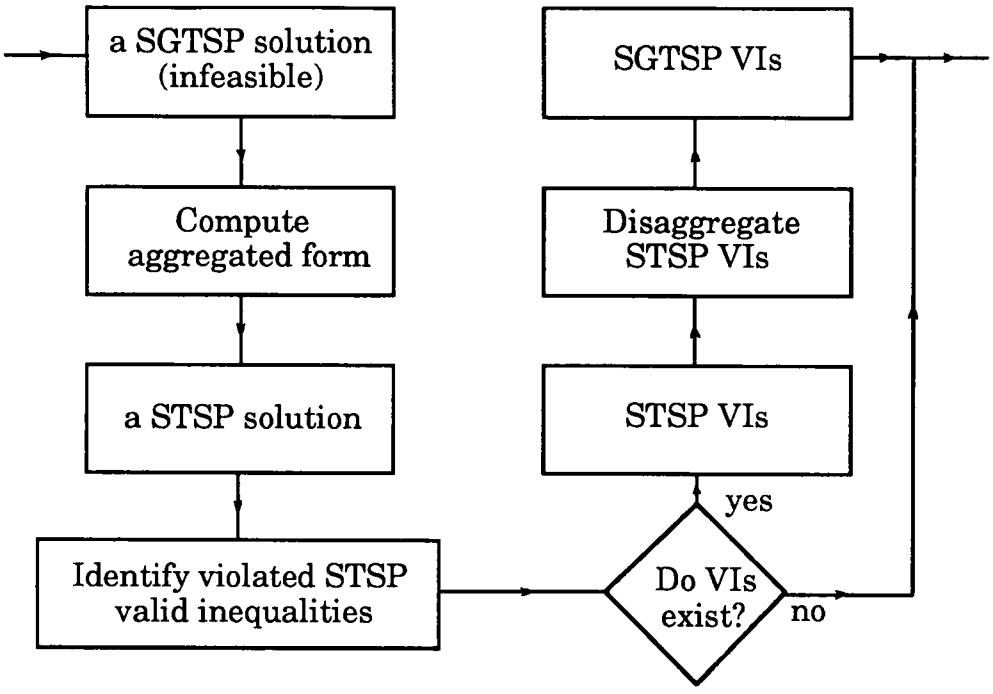


Figure 3.7: A flow diagram for deriving violated SGTSP inequalities from violated STSP inequalities.

3.2.3 Mathematical Substantiation

Lemma 3.1. A symmetric GTSP tour over a GTSP graph G is a symmetric TSP tour over G_a, the aggregated form of G.

Proof. A symmetric GTSP tour is a cycle that contains exactly one node of every nodeset of a GTSP graph. Viewing every nodeset as a node, in conformity with the way that we represent each node in aggregated form of GTSP graph, the symmetric GTSP tour is a cycle over the nodes of the graph G_a. Hence the result. ■

Theorem 3.2. Any constraint that is valid for a symmetric TSP over m cities can be generalized to a symmetric GTSP over m nodesets.

Proof. The proof follows directly from Lemma 3.1 and the relation (3.12). ■

Recall the following definitions from chapter 1, where, for $W_a \subseteq \mathcal{V}_a$, $E_a(W_a)$ defines the set of aggregated edges with both of their end nodes in W_a. That is, $E_a(W_a) = \{ (I;J) \mid (I;J) \in \mathcal{E}_a, I \in W_a, J \in W_a \}$. In addition, recall that for $W \subseteq \mathcal{V}$, $S(W)$ is the set of all nodesets each of which has at least one node in W. So, $S(W) = \{ S_I \mid |S_I \cap W| \geq 1, I \in M \}$, $M = \{1, 2, ..., m\}$. Now the subtour elimination constraints for graph $\mathcal{G}_a$ can be stated as

$$X(E_a(W_a)) \leq |W_a| - 1 \qquad \text{for all } W_a \subseteq \mathcal{V}_a,\ 2 \leq |W_a| \leq m-2.$$

The next corollary follows from Theorem 3.2 with regard to the above definitions.

Corollary 3.1. The generalized subtour elimination constraints over a collection of nodesets S(W) in $\mathcal{G}$ with $W \subseteq \mathcal{V}$ is given as

$$x(E(W)) \leq |S(W)| - 1 \qquad \text{for all } W \subseteq \mathcal{V},\ 2 \leq |S(W)| \leq m-2,$$

and is a valid inequality for the SGTSP.

Our rationale for focusing on the subtour elimination constraints among known valid inequalities for the TSP is that 1) the subtour elimination constraint defines a facet for traveling salesman polytope (see Nemhauser and Wolsey [1988]), 2) there are polynomial algorithms for identification of them, and 3) we use the generalized subtour elimination constraints to prove validity of kite constraints that we will discuss in the next section.

3.3 Kite Inequalities

In this section we study the third class of SGTSP valid inequalities known as kite inequalities. We discuss two groups of valid SGTSP constraints namely *simple kite inequalities* and *kite chain inequalities*. Both belong to the class of kite inequalities and, in fact, the first group is a special case of the latter one.

The kite inequalities are needed since introducing the two preceding classes of inequalities are not sufficient to ensure an integer solution. For instance, the example given in Figure 3.8 shows such a case. There are no violated DP constraints and there are no violated valid symmetric TSP inequalities over the aggregated form of $\mathcal{G}$, $\mathcal{G}_a$. The solution given on $\mathcal{G}_a$, in fact, is associated with a point inside of the TSP convex hull.

The solution is actually a convex combination of three feasible TSP tours. These tours are T_1 = [(1;2), (2;3), (3;4), (1;4)], T_2 = [(1;3), (2;3), (2;4), (1;4)], and T_3 = [(1;2), (2;4), (3;4), (1;3)]. Let $\overline{X}$ = ($\overline{X}_{12}$, $\overline{X}_{13}$, $\overline{X}_{14}$, $\overline{X}_{23}$, $\overline{X}_{24}$, $\overline{X}_{34}$) be the solution vector associated with a TSP over $\mathcal{G}_a$. The following solutions are associated with the three tours respectively.

$$\begin{bmatrix}1\\0\\1\\1\\0\\1\end{bmatrix} \text{ for } T_1, \quad \begin{bmatrix}0\\1\\1\\1\\1\\0\end{bmatrix} \text{ for } T_2, \text{ and } \begin{bmatrix}1\\1\\0\\0\\1\\1\end{bmatrix} \text{ for } T_3.$$

The convex combination is as follows.

$$0.25\begin{bmatrix}1\\0\\1\\1\\0\\1\end{bmatrix} + 0.25\begin{bmatrix}0\\1\\1\\1\\1\\0\end{bmatrix} + 0.5\begin{bmatrix}1\\1\\0\\0\\1\\1\end{bmatrix} = \begin{bmatrix}0.75\\0.75\\0.5\\0.5\\0.75\\0.75\end{bmatrix}.$$

This means that the given solution, (0.75, 0.75, 0.5, 0.5, 0.75, 0.75), over $\mathcal{G}_a$ is

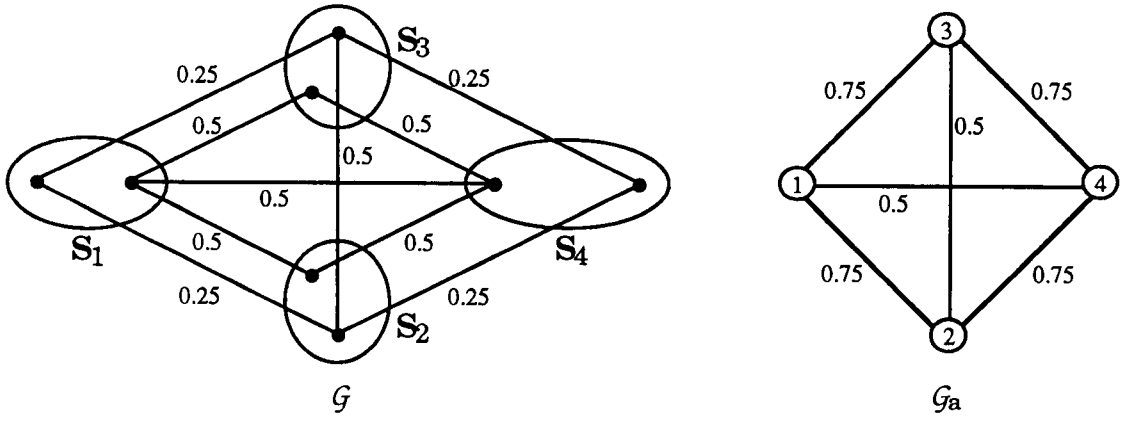


Figure 3.8: A SGTSP example with no violated
DP constraints and generalization of STSP valid inequalities.

a point inside of the traveling salesman polytope over $\mathcal{G}_a$.

3.3.1 Kite Frame and Kite Chain

In Chapter 1, we defined the concept of the *covering nodesets* of a collection of nodes in a GTSP graph $\mathcal{G} = (\mathcal{V},\mathcal{E})$. To facilitate our discussion of kite frame, we repeat the definition in this section. It should be noted that for a given problem all nodesets are predefined and the nodes in each nodeset are known. In other words, in any SGTSP with n nodes and m nodesets, nodes are partitioned into m nodesets and no repartitioning can occur.

Let W be a collection of nodes in a GTSP graph $\mathcal{G} = (\mathcal{V},\mathcal{E})$. For $W \subseteq \mathcal{V}$, an index set S(W), called *covering nodesets* of W, is the set of indices of nodesets each of which has at least one node in W. So, $S(W) = \{ I \mid |S_I \cap W| \geq 1, I \in \{ 1, 2, ..., m \} \}$. For example $S(\mathcal{V}) = \{ 1, 2, ..., m \}$, and $|S(\mathcal{V})| = m$. We say that a nodeset S_I is a *covering nodeset* for a collection of nodes W, if

$I \in S(W)$. In our examples that follows we use bold numbers to identify the indices of nodesets. The set of all edges that have both their end nodes in W is denoted by E(W), that is, $E(W) = \{ (i;j) \mid (i;j) \in \mathcal{E}, i \in W, j \in W \}$.

A *kite frame* K_i in a GTSP graph $\mathcal{G} = (\mathcal{V},\mathcal{E})$ is a collection of nodes satisfying the following properties.

a. The number of covering nodesets for K_i is at least two, that is,

$|S(K_i)| \geq 2$.

b. The kite frame K_i is a proper subset of all nodes in its covering

nodesets, that is, $K_i \subset \bigcup_{J \in S(K_i)} S_J$.

For example, consider a SGTSP with 20 nodes - 8 nodesets, where S_1 = {1}, S_2 = {2, 3, 4}, S_3 = {5, 6, 7}, S_4 = {8, 9, 10}, S_5 = {11, 12, 13}, S_6 = {14, 15, 16}, S_7 = {17, 18}, and S_8 = {19,20}. A collection of nodes K_i = {3, 11, 13, 14} satisfies the properties of a kite frame. Since $S(K_i)$ = {**2**, **5**, **6**}, $|S(K_i)| \geq 2$. In addition,

$$\bigcup_{J \in S(K_i)} S_J = S_2 \cup S_5 \cup S_6 = \{2, 3, 4, 11, 12, 13, 14, 15, 16\}.$$

Therefore $K_i \subset \bigcup_{J \in S(K_i)} S_J$.

Suppose we are given a collection of $k \geq 2$ distinct kite frames denoted by $K_1, K_2, ..., K_k$. For the given kite frames, we can distinguish $r \geq 0$ nodesets each of which is a covering nodeset for more than one kite frame. These nodesets are referred to as the *connective nodesets* corresponding to the collection of k kite frames. More formally, if K_i and K_j are two distinct kite frames of a given collection of kite frames and $I \in S(K_i)$ and $I \in S(K_j)$, then we identify S_I to be a connective nodeset for the given collection. For a given

collection of kite frames, all nodesets are partitioned into two groups, connective nodesets and *nonconnective nodesets*. That is, in a collection of kite frames any nodeset which is not a connective nodeset is a nonconnective nodeset. Note that by definition, a nodeset S_I with $|S_I| = 1$ would not be eligible to be a connective nodeset.

Suppose for our previous 20 nodes - 8 nodesets example, a collection of three kite frames, K_1, K_2, and K_3, where $K_1 = \{1, 2, 10, 14\}$, $K_2 = \{4,11,15,18\}$, and $K_3 = \{6,13,16\}$. For this collection of kite frames we can identify three connective nodesets. The connective nodesets are S_2, S_5, and S_6. Note that $\mathbf{2} \in S(K_1) \cap S(K_2)$, $\mathbf{5} \in S(K_2) \cap S(K_3)$, and $\mathbf{6} \in S(K_1) \cap S(K_2) \cap S(K_3)$. The nodesets S_1, S_3, S_4, and S_7 are the nonconnective nodesets with respect to the given collection of kite frames.

A *kite chain C* is a collection of $k \geq 2$ kite frames satisfying the following properties.

a. No two kite frames intersect, meaning $K_i \cap K_j = \emptyset$ for $i \neq j$.

b. Any kite frame K_i in the kite chain must have at least one and at most $|S(K_i)| - 1$ connective nodesets among its covering nodesets.

Property b implies that every kite frame K_i must have at least one and at most $|S(K_i)| - 1$ nonconnective nodesets among its covering nodesets.

3.3.1.1 Description of the Simple Kite

Before discussing the simple kite in detail, we pictorially introduce it by Figure 3.9.

A *simple kite* is a kite chain consisting of two kite frames K_t and K_i and exactly one connective nodeset. Further, the kite frame K_t has exactly two covering nodesets, $|S(K_t)| = 2$. We shall refer to K_t as the *tail* of the simple

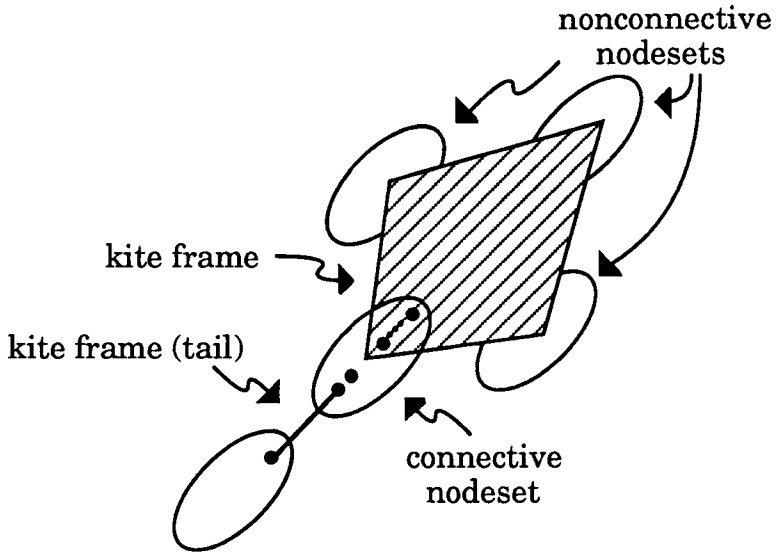


Figure 3.9: The simple kite.

kite. Figure 3.10 shows four different examples of simple kite node configurations. Note the variety of tail configurations in the figure.

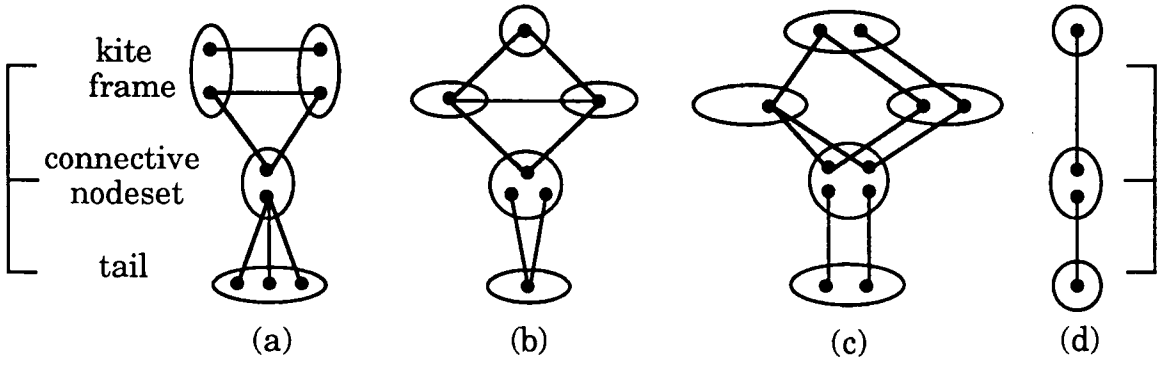


Figure 3.10: Four different examples of the simple kite.

Figure 3.10-(d) depicts the simplest form of the simple kite.

3.3.1.2 Simple Kite Inequality

A simple kite consists of two kite frames K_t and K_1 with $|S(K_t)| = 2$. The following inequality is associated with the simple kite:

$$x(E(K_t)) + x(E(K_1)) \leq |S(K_1)| - 1.$$

In Figure 3.11, a simple kite defined over a P_1 solution of a 40 node - 10 nodeset SGTSP is depicted. The problem is associated with a complete graph.

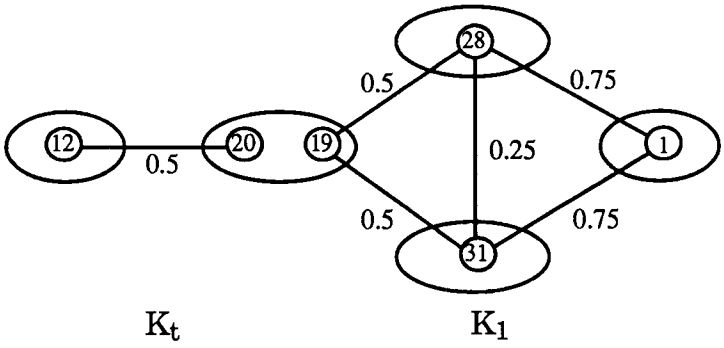


Figure 3.11: A example of the simple kite.

In this example,

$$x(E(K_t)) = x_{12,20},$$

$$x(E(K_1)) = x_{1,19} + x_{1,28} + x_{1,31} + x_{19,28} + x_{19,31} + x_{28,31},$$

$$|S(K_1)| = 4.$$

The inequality below is the simple kite inequality associated with this example.

$$x(E(K_t)) + x(E(K_1)) \leq |S(K_1)| - 1.$$

This constraint is violated, since the total value of the left hand side of the inequality is 3.25 and the right hand side value is 3.

3.3.1.3 Description of Kite Chain

Consider a kite chain *C* with k kite frames and assume that we have identified r connective nodesets with respect to the chain. Let the sets K = { 1, 2, ..., k } and R = { 1, 2, ..., r } be the sets of indices of kite frames and

connective nodesets of C, respectively. If S_I is the nodeset corresponding to the lth connective nodeset of the kite chain, $l \leq r$, then we define S_{C_l} as another name for S_I and in fact $S_I \equiv S_{C_l}$. An index set C_l defines the set of indices of all kite frames having the lth connective nodeset, S_{C_l}, in common, that is $C_l = \{ i \mid S_{C_l} \cap K_i \neq \emptyset, i \in K \}$. Accordingly, $|C_l|$ gives the cardinality of the index set C_l for all $l \in R$. Note that, $|S_{C_l}| \geq |C_l|$ for all $l \in R$, and $S_{C_l} \cap K_i \neq \emptyset$, if S_{C_l} is a connective nodeset chaining kite frame K_i to some other kite frames in C. The set of nodes in common between a kite frame K_i and a connective nodeset S_{C_l} is denoted by $C_l(K_i)$, that is, $C_l(K_i) = S_{C_l} \cap K_i$. Therefore $C_l(K_i) = \emptyset$, if the connective nodeset S_{C_l} is not covering for the kite frame K_i.

In Figure 3.12, three different kite chain node configurations are shown. In the figure, each kite frame is contained within a shaded rectangle.

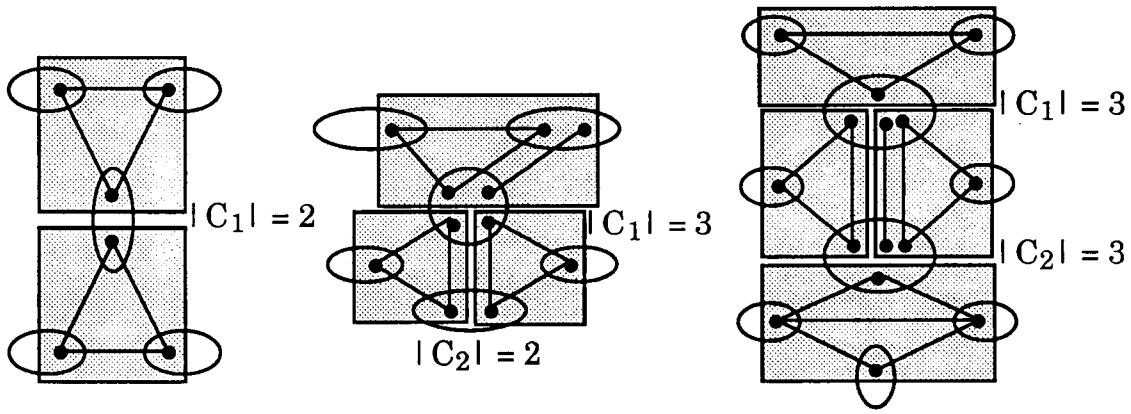


Figure 3.12: Three kite chain examples.

3.3.1.4 Kite Chain Inequality

For a kite chain C with k kite frames and r connective nodesets we propose a *kite chain inequality* as,

$$\sum_{i=1}^{k} x(E(K_i)) \leq \sum_{i=1}^{k} (|S(K_i)|-1) - \sum_{l=1}^{r} (|C_l|-1).$$

Figure 3.13 illustrates a kite chain defined over a P_1 solution of a SGTSP from a complete graph.

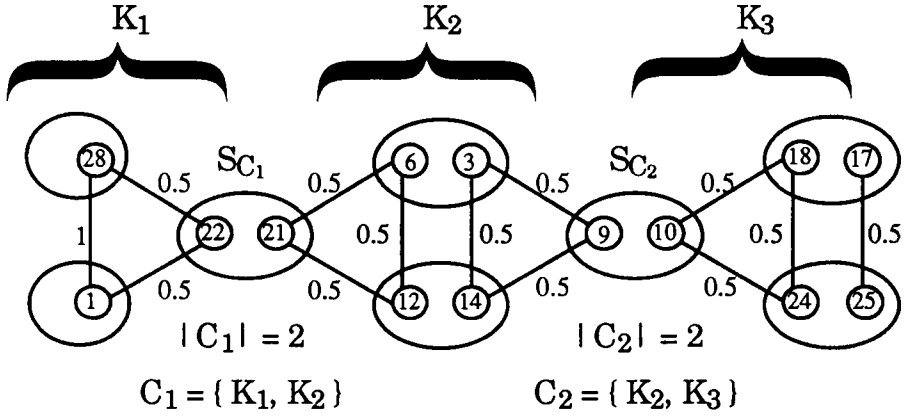


Figure 3.13: A kite chain example.

In this example,

$$x(E(K_1)) = x_{1,22} + x_{1,28} + x_{22,28},$$

$$x(E(K_2)) = x_{3,9} + x_{3,12} + x_{3,14} + x_{3,21} + x_{6,9} + x_{6,12} + x_{6,14} + x_{6,21} + x_{9,12} + x_{9,14} + x_{9,21} + x_{12,21} + x_{14,21},$$

$$x(E(K_3)) = x_{10,17} + x_{10,18} + x_{10,24} + x_{10,25} + x_{17,24} + x_{17,25} + x_{18,24} + x_{18,25},$$

$|S(K_1)| = 3$, $|S(K_2)| = 4$, $|S(K_3)| = 3$, $|C_1| = 2$, $|C_2| = 2$,

$k = 3$, and $r = 2$.

The kite chain inequality associated with the kite chain is,

$$\sum_{i=1}^{3} x(E(K_i)) \leq \sum_{i=1}^{3} (|S(K_i)| - 1) - \sum_{l=1}^{2} (|C_l| - 1) =$$

$$[(3 - 1) + (4 - 1) + (3 - 1)] - [(2 - 1) + (2 - 1)] = 5.$$

This inequality is a violated inequality for the given problem. The constraint is violated since $\sum_{i=1}^{3} \bar{x}(E(K_i)) = 7$.

As mentioned earlier, the simple kite inequality is a special case of the kite chain inequality. Consider a kite chain consisting two kite frames K_1, and K_2 with $|S(K_2)| = 2$, and one connective nodeset S_{C_1}. By applying the kite chain inequality to this instance, we get

$$x(E(K_1)) + x(E(K_2)) \leq [|S(K_1)| - 1 + |S(K_2)| - 1] - [|C_1| - 1].$$

We have, however, $|S(K_2)| = 2$, and $|C_1| = 2$ by the statement of the example. Substituting these values in the right hand side of the constraint yields

$$x(E(K_1)) + x(E(K_2)) \leq |S(K_1)| - 1,$$

which is identical to the simple kite inequality with K_2 designated as the tail frame.

3.3.2 Mathematical Substantiation

In this section we prove that the kite constraints are valid SGTSP inequalities. The development proceeds from simple cases of kite inequalities to more complex ones such as kite chain inequalities. First, we prove that the simple kite constraint (a special case of the kite chain constraint) is a valid inequality for the SGTSP. Then, inequalities associated with the following kite chain descriptions are proved to be valid

inequalities for the SGTSP. The kite chain descriptions are, 1) a kite chain containing two kite frames and one connective nodeset, 2) a kite chain containing more than two kite frames and only one connective nodeset, and 3) a kite chain containing two kite frames and two or more connective nodeset. The section concludes with a theorem which states inequality associated with a kite chain containing two or more kite frames and one or more connective nodesets is a valid inequality for the SGTSP.

Theorem 3.3. The simple kite inequality is a valid inequality for the SGTSP.

Proof. Consider a simple kite consisting of a kite frame K_t (as tail) and another kite frame K_1. In addition, let S_{C_1} be the connective nodeset of the simple kite. We want to prove that the expression (3.12),

$$x(E(K_t)) + x(E(K_1)) \leq |S(K_1)| - 1 \qquad (3.12),$$

is a valid inequality for the symmetric GTSP.

In the left hand side of (3.12), the term corresponding to the tail, $x(E(K_t))$, can obtain two possible values in a SGTSP tour, either 0 or 1. The term $x(E(K_t)) = 1$ if one of the edges in $E(K_t)$ is in the tour, and $x(E(K_t)) = 0$ otherwise.

If $x(E(K_t)) = 0$, we are left with

$$x(E(K_1)) \leq |S(K_1)| - 1 \qquad (3.13).$$

This inequality, (3.13), is the generalization of the STSP subtour elimination constraint over the covering nodesets of K_1. Therefore, by Corollary 3.1, the inequality (3.12) is valid.

Now, suppose $x(E(K_t)) = 1$, and recall two of our basic assumptions. First, there exist no intraset edges and a tour must enter and leave a

nodeset from the same node. Second, no two kite frames intersect. Since $x(E(K_t)) = 1$, this means that one of the edges in $E(K_t)$ is in the tour and this edge is incident to a node in $C_1(K_t)$, $C_1(K_t) = S_{C_1} \cap K_t$. Thus, in $E(K_1)$, all the edges incident to the nodes in $C_1(K_1)$, $C_1(K_1) = S_{C_1} \cap K_1$, must obtain the value zero. That is, all edges that are in $E(K_1)$ and are incident to the nodes in the connective nodeset S_{C_1} cannot obtain any values but zeros. Therefore, we can assume that the connective nodeset is omitted from the set of covering nodesets of K_1. In such a case, by applying Corollary 3.1 to the remaining $|S(K_1)|-1$ covering nodesets of K_1 we get,

$$x(E(K_1)) \leq [|S(K_1)| - 1] - 1 \qquad (3.14).$$

Simply add $x(E(K_t)) = 1$ to the both sides of the inequality (3.14). Simplifying the result yields (3.12). Hence the proof. ■

The following lemma concerns a special case of kite chain inequality that is slightly more general than the simple kite inequality.

Lemma 3.2. The kite chain inequality which corresponds to a kite chain with two kite frames K_i and K_j and one connective nodeset S_{C_1} is a valid inequality for the SGTSP.

Proof. We want to show that the inequality (3.15) is a valid SGTSP inequality.

$$x(E(K_i)) + x(E(K_j)) \leq |S(K_i)| - 1 + |S(K_j)| - 1 - (|C_1| - 1) \qquad (3.15).$$

Consider the kite frame K_i and its associated edges, $E(K_i)$. By Corollary 3.1, $x(E(K_i)) \leq |S(K_i)| - 1$ is a valid inequality. If an edge in $E(K_i)$ and incident to a node in S_{C_1} is in a SGTSP tour, then none of the edges in $E(K_j)$ and incident to the nodes in S_{C_1} can be in the tour. The reason is that the

opposite case causes disjointedness in the tour. Thus we can drop the connective nodeset S_{C_1} from the collection of covering nodesets of K_j. Hence $x(E(K_j)) \leq [|S(K_j)| - 1] - 1$ is valid. Combining the last two inequalities and the fact that $|C_1| = 2$ establishes that (3.15) is a valid inequality.

If none of the edges in the kite chain, $E(K_i \cup K_j)$, and incident to the nodes in S_{C_1} is in a tour, then we can assume that the connective nodeset is omitted from both $S(K_i)$ and $S(K_j)$. This omission yields the following valid inequalities for both kite frames K_i and K_j respectively.

$$x(E(K_i)) \leq [|S(K_i)| - 1] - 1,$$

$$x(E(K_j)) \leq [|S(K_j)| - 1] - 1.$$

Adding this two inequalities leads to,

$$x(E(K_i)) + x(E(K_j)) \leq [|S(K_i)| - 1] + [|S(K_j)| - 1] - 2 =$$

$$|S(K_i)| - 1 + |S(K_j)| - 1 - (|C_1| - 1),$$

in the view of the fact that $|C_1| = 2$. This establishes the proof. ■

In Figure 3.12-(a), a kite chain with two kite frames and one connective nodeset is depicted. We now extend the results of Lemma 3.2 to the case of more than two kite frames with only one connective nodeset.

Lemma 3.3. Any kite chain inequality corresponding to a kite chain with $k \geq 2$ kite frames and exactly one connective nodeset is a valid inequality for the SGTSP.

Proof. We prove this lemma by induction. Let S_{C_1} be the connective nodeset in the kite chain described by the statement of the theorem. Then, $|C_1| = k$, since S_{C_1} is the only connective nodeset in the kite chain. Now,

consider the inequality (3.16) that furnishes the kite chain inequality associated with the kite chain.

$$\sum_{i=1}^{k} x(E(K_i)) \le \sum_{i=1}^{k} (|S(K_i)|-1) - (k-1). \qquad (3.16)$$

Let C_k be the theorem that (3.16) is a SGTSP valid inequality associated with the kite chain. Then C_2 is the theorem that k = 2, which is certainly true by Lemma 3.3.

Suppose C_k is true. Chaining the k+1st kite frame with $S_{C_1} \cap K_{k+1} \neq \emptyset$ to the original kite chain, using S_{C_1} as the only means of connection, suggests the two following cases.

Case a. If an edge in $E(K_1 \cup K_2 ... \cup K_k)$ and incident to a node of S_{C_1} is in a tour, then none of the edges in $E(K_{k+1})$ and incident to the nodes of S_{C_1} can be in the tour. So, we can assume that S_{C_1} stops participating in the kite frame K_{k+1}. Because of the preceding assumption inequality (3.17) is valid by Corollary 3.1.

$$x(E(K_{k+1})) \le [|S(K_{k+1})|-1] - 1 \qquad (3.17)$$

Combining the inequalities (3.16) and (3.17) yields the inequality (3.18) which is C_{k+1}; hence, C_k implies C_{k+1}, and therefore (3.16) holds for all k.

$$\sum_{i=1}^{k+1} x(E(K_i)) \le \sum_{i=1}^{k+1} (|S(K_i)|-1) - (k+1-1). \qquad (3.18)$$

Case b. If none of the edges that are in $E(K_1 \cup K_2 ... \cup K_{k+1})$ and incident to the nodes of S_{C_1} are in a tour, again S_{C_1} can be prevented from taking part as a covering nodeset in K_{k+1}. Thus, (3.17) is valid and by the same argument that was pointed out in case a, (3.16) is valid for all k. ■

Figure 3.14 illustrates a kite chain with three kite frames and one connective nodeset.

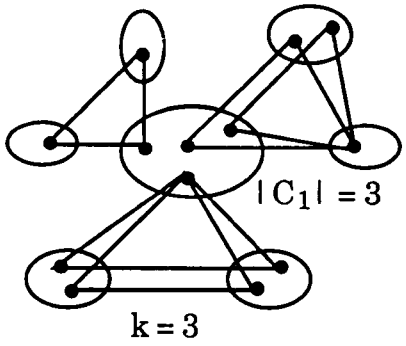


Figure 3.14: A kite chain with three
kite frames and one connective nodeset.

The following lemma discusses a special case of the kite chain inequality, where the associated kite chain contains two kite frames that are chained together by more than one connective nodeset.

Lemma 3.4. The kite chain inequality associated with a kite chain having two kite frames and r connective nodesets, $r \geq 1$, is a SGTSP valid inequality.

Proof. Suppose S_{C_1}, S_{C_2}, ..., and S_{C_r} are the connective nodesets of a kite chain with two kite frames, K_i, and K_j. By definition we have $|C_1| = |C_2| = \ldots = |C_r| = 2$, $|S(K_i)| > r$, $|S(K_j)| > r$, and the kite chain inequality associated with the kite chain is given by (3.19).

$$x(E(K_i)) + x(E(K_j)) \leq |S(K_i)| - 1 + |S(K_j)| - 1 - \sum_{l=1}^{r} (|C_l| - 1). \qquad (3.19)$$

Since $|C_1| = |C_2| = \ldots = |C_r| = 2$, the inequality (3.19) can be written as

$$x(E(K_i)) + x(E(K_j)) \leq |S(K_i)| - 1 + |S(K_j)| - 1 - r. \qquad (3.20)$$

Let N_r be the set of r nodes $n_1, n_2, \ldots, n_r$, each of which belongs to a connective nodeset in such a way that $n_1 \in S_{C_1}$, $n_2 \in S_{C_2}$, ..., $n_r \in S_{C_r}$. In

addition, assume that all the nodes in N_r are part of the selected nodes being included in a SGTSP tour. Let $r = r_i + r_j + r_o$ where $r_i, r_j, r_o \in Z_+^1$ and define

r_i as the number of nodes in N_r that are in K_i,

r_j as the number of nodes in N_r that are in K_j, and

r_o as the number of nodes in N_r that are not in $K_i \cup K_j$.

That is, $r_i = |N_r \cap K_i|$, $r_j = |N_r \cap K_j|$, and $r_o = |N_r \cap \{ \mathcal{V} \setminus [K_i \cup K_j] \}|$. Note that, $r \geq r_i$, $r \geq r_j$, $r \geq r_o$, $r \geq r_i + r_o$, $r \geq r_j + r_o$, and all r connective nodesets are part of covering nodesets of each of the two kite frames.

Now, in a SGTSP tour, consider kite frame K_i where no edges in $E(K_i)$ are incident to any nodes in $r_j + r_o$ connective nodesets of the kite frame. Thus $r_j + r_o$ covering nodesets of K_i are not included in the corresponding subtour elimination constraint. Hence by Corollary 3.1 we get the valid inequality (3.21) for K_i.

$$x(E(K_i)) \leq [|S(K_i)| - (r_j + r_o)] - 1 \qquad (3.21).$$

Using the same argument that we applied for K_i, the corresponding valid inequality for K_j is the inequality (3.22).

$$x(E(K_j)) \leq [|S(K_j)| - (r_i + r_o)] - 1 \qquad (3.22).$$

Combining the inequalities (3.21) and (3.22) produces valid inequality (3.23).

$$x(E(K_i)) + x(E(K_j)) \leq |S(K_i)| - 1 + |S(K_j)| - 1 - (r + r_o) \qquad (3.23).$$

Since $r_o \in Z_+^1$, the right hand side of (3.23) is less than or equal to

$$|S(K_i)| - 1 + |S(K_j)| - 1 - r.$$

This establishes the validity of inequality (3.20), hence the proof. ■

Consider a kite chain with k kite frames in a set K, and r connective nodesets in a set R. Inequality (3.24) is the kite chain inequality associated with the kite chain.

$$\sum_{i=1}^{k} x(E(K_i)) \le \sum_{i=1}^{k} (|S(K_i)|-1) - \sum_{l=1}^{r} (|C_l|-1). \qquad (3.24)$$

Now, we use a constructive method starting from a kite frame to build the kite chain iteratively. That is, we take a kite frame and then chain another kite frame, if it is linkable to the previous one(s) by means of connective nodesets. We repeat this step until all k kite frames are chained together.

In order to discuss the iterative process of kite chain growth, we use the following notation and definitions.

a. We say two kite frames are *linkable*, if they share at least one connective nodeset as one of their covering nodesets,
b. We say a kite frame is *scanned*, if it is selected to be chained to the others.
c. At iteration i, a kite frame selected to be scanned and will be indexed K_i.
d. The set $\overline{K}^i = \{ K_1, K_2, ...K_i \}$ is the set of all i scanned kite frames.
e. Any kite frame K_i, $i > 1$, selected for scanning must be linkable to at least one of the i-1 previously scanned kite frames.
f. The set S^i is set of all connective nodesets which are used *only* to chain the ith kite frame to the scanned kite frames in $\overline{K}^{i-1}$. We call these connective nodesets as *scanned connective nodesets*. The quantity $|S^i|$ gives the number of scanned connective nodesets at iteration i. Based on part e of notation and definitions, $|S^1| = \emptyset$ and $|S^i| > 0$ for all $i > 1$. Note that $S^i \subset S(K_i)$ and thus $|S^i| < |S(K_i)|$.

g. The set $\overline{S}^i$ is the set of all identified and scanned connective nodesets that have been used to chain all i scanned kite frames in $\overline{K}^i$. In other words, $\overline{S}^i$ is the set of all connective nodesets scanned during i iterations, that is,

$$\overline{S}^i = \bigcup_{j=1}^{i} S^j, \text{ and } |\overline{S}^i| \leq |\overline{S}^k| = r, \text{ for all } i \in \{ 1, 2, ..., k \}.$$

h. At every iteration i, when we select a linkable kite frame K_i to be chained to the previously scanned kite frames, $\overline{K}^i$, we encounter two types of connective nodesets in S^i. The first type are the ones that are scanned earlier and are in $\overline{S}^{i-1}$. The other type are the ones not in $\overline{S}^{i-1}$ and were unscanned until the start of the ith iteration. Let rs^i represent the cardinality of the first type of connective nodesets that are going to be rescanned at iteration i. The number of the second type connective nodesets is denoted by ns^i which stands for newly scanned connective nodesets at the ith iteration. Thus $|S^i| = rs^i + ns^i$. At the first iteration $|S^1| = 0$, hence $rs^1 = 0$ and $ns^1 = 0$. For the second iteration $|S^2| = rs^2 > 0$ and $ns^2 = 0$, since at this iteration no scanned connective nodesets exist to be rescanned.

i. The notation L^i denotes the set of indices of the connective nodesets in S^i.

j. The notation $\overline{L}^i$ denotes the set of indices of the connective nodesets in $\overline{S}^i$ as $\{ 1, 2, ..., |\overline{S}^i| \}$. Note that at the last iteration, k, $\overline{L}^k = \{ 1, 2, ..., |\overline{S}^k| = r \}$, since we have scanned all the r connective nodesets.

k. Previously we defined the index set C_l as the set of indices of all kite frames having the lth connective nodeset, S_{C_l}, in common. Now, we define C_l^i as the index set of all kite frames in $\overline{K}^i$ that share the connective nodeset S_{C_l}. That is, until the end of iteration i, $|C_l^i|$ kite

frames share the connective nodeset S_{C_l}. We call C_l^i the *covered kite frames* by S_{C_l} until the end of iteration i, and all $|C_l^i|$, the *number of covered kite frames* by S_{C_l} until the end of iteration i. Note that $C_l^k = C_l$ and $|C_l^k| = |C_l|$.

1. The connective nodesets in $\overline{S}^i$ are partitioned into three mutually exclusive and exhaustive subsets.

   i′. The connective nodesets that are rescanned at ith iteration.

   ii′. The connective nodesets that are just added to $\overline{S}^i$ and are newly scanned at iteration i.

   iii′. The connective nodesets that have been scanned prior to iteration i and are not scanned at iteration i.

   We denote the cardinality of the third subset by ps^i. For the cardinality of the first two subsets, we have already denoted rs^i and ns^i, respectively, in part h. Using our notation L^i, the set of indices of connective nodesets in S^i, and $\overline{L}^i$, the set of indices of connective nodesets in $\overline{S}^i$, enables us to extract the indices of connective nodesets in each subset of $\overline{S}^i$. They are as follows.

   i°. Set of indices of rescanned connective nodesets in $\overline{S}^i = \overline{L}^{i-1} \cap L^i$, $i > 1$. The number of this kind of connective nodesets is rs^i.

   ii°. Set of indices of newly scanned connective nodesets in $\overline{S}^i = L^i \setminus (\overline{L}^{i-1} \cap L^i)$, $i > 1$. There are ns^i connective nodesets of this type.

   iii°. Set of indices of previously scanned connective nodesets in $\overline{S}^i = \overline{L}^i \setminus L^i$, $i > 1$. There are ps^i connective nodesets are in this partition.

We use the notation, RS^i, NS^i, and PS^i, to denote the above index sets respectively. That is,

$$RS^i = \overline{L}^{i-1} \cap L^i, \ |RSi| = rs^i, \ i > 1,$$

$$NS^i = L^i \setminus (\overline{L}^{i-1} \cap L^i), \ |NSi| = ns^i, \ i > 1,$$

$$PS^i = \overline{L}^i \setminus L^i, \ |PSi| = ps^i, \ i > 1.$$

m. At every iteration i we introduce one kite frame to be scanned. therefore, we increase the cardinality of kite frames that share a rescanned connective nodeset S_{C_l} by one. That is,

$$|C_l^i| = |C_l^{i-1}| + 1, \text{ for } l \in RS^i, \ i > 1.$$

For any connective nodeset that is newly scanned at iteration i, the number of kite frames sharing it is clearly two, meaning,

$$|C_l^i| = 2, \text{ for } l \in NS^i, \ i > 1.$$

Covering nodesets of every kite frame scanned at iteration i do not contain any of the previously scanned connective nodesets. Thus, the cardinality of kite frames which share any of the previously scanned connective nodesets remains unchanged. Hence,

$$|C_l^i| = |C_l^{i-1}|, \text{ for } l \in PS^i, \ i > 1.$$

n. Let $N_{|S^i|}$ denote the collection of $|S^i|$ nodes that participate in a SGTSP tour and each of which belongs to a connective nodeset in S^i. Every node in $N_{|S^i|}$ has to be in only one of the three possible states.

i′. The nodes in $\overline{K}^{i-1}$.

ii′. The nodes in K_i.

iii′. The nodes that are not in $\overline{K}^i = \overline{K}^{i-1} \cup K_i$, that is, the nodes in $\mathcal{V} \setminus \overline{K}^i$.

We denote the number of nodes in each of the above three states as p^i, q^i, and r^i respectively. Therefore $|S^i| = p^i + q^i + r^i$, where,

i°. $p^i = |N_{|S^i|} \cap \overline{K}^{i-1}|$ as cardinality of $P^i = N_{|S^i|} \cap \overline{K}^{i-1}$.

ii°. $q^i = |N_{|S^i|} \cap K_i|$ as cardinality of $Q^i = N_{|S^i|} \cap K_i$.

iii°. $r^i = |N_{|S^i|} \cap \overline{K}^{i\prime}|$ as cardinality of $R^i = N_{|S^i|} \cap \overline{K}^{i\prime}$.

Note that $\overline{K}^{i\prime}$ is the complement of $\overline{K}^i$, and is equal to $\mathcal{V} \setminus \overline{K}^i$.

Lemma 3.4 proves the validity of an expression that is used in the iterative process to be discussed later.

Lemma 3.5. At the iterative process of kite chain growth, for any iteration $i > 1$,

$$\sum_{l \in \overline{L}^{i-1}} (|C_l^{i-1}| - 1) + |S^i| = \sum_{l \in \overline{L}^i} (|C_l^i| - 1).$$

Proof. We have $|S^i| = ns^i + rs^i$, by statement of part h of the notation and definitions. Since $|C_l^i| = 2$, for all $l \in NS^i$, it follows that,

$$(|C_l^i| - 1) = 1, \text{ for all } l \in NS^i,$$

and hence,

$$ns^i = \sum_{l \in NS^i} (|C_l^i| - 1). \qquad (3.25)$$

In addition, rs^i can be written as

$$rs^i = \sum_{l \in RS^i} 1. \qquad (3.26)$$

Meanwhile,

$$\sum_{l \in \overline{L}^{i-1}} (|C_l^{i-1}| - 1) = \sum_{l \in PS^i} (|C_l^{i-1}| - 1) + \sum_{l \in RS^i} (|C_l^{i-1}| - 1), \qquad (3.27)$$

since $\overline{L}^{i-1} = PS^i \cup RS^i$, and $PS^i \cap RS^i \neq \emptyset$. From (3.25), (3.26), and (3.27) we obtain

$$\sum_{l \in \overline{L}^{i-1}} (|C_l^{i-1}| - 1) + |S^i| =$$

$$\sum_{l \in PS^i} (|C_l^{i-1}| - 1) + \sum_{l \in RS^i} (|C_l^{i-1}| - 1) + \sum_{l \in RS^i} 1 + \sum_{l \in NS^i} (|C_l^i| - 1). \qquad (3.28)$$

Combining the second and the third terms of the right hand side of (3.28) yields

$$\sum_{l \in \overline{L}^{i-1}} (|C_l^{i-1}| - 1) + |S^i| =$$

$$\sum_{l \in PS^i} (|C_l^{i-1}| - 1) + \sum_{l \in RS^i} (|C_l^{i-1}| + 1 - 1) + \sum_{l \in NS^i} (|C_l^i| - 1). \qquad (3.29)$$

Using the results stated in part m of the notation and definitions,
$|C_l^i| = |C_l^{i-1}|$, for all $l \in PS^i$, and $|C_l^i| = |C_l^{i-1}| + 1$, for all $l \in RS^i$,

equality (3.29) can be modified to

$$\sum_{l \in \overline{L}^{i-1}} (|C_l^{i-1}| - 1) + |S^i| =$$

$$\sum_{l \in PS^i} (|C_l^i| - 1) + \sum_{l \in RS^i} (|C_l^i| - 1) + \sum_{l \in NS^i} (|C_l^i| - 1). \qquad (3.30)$$

Since the subsets PSi, RSi, and NSi are mutually exclusive and exhaustive partitions of $\overline{L}^i$, we have (3.31).

$$\overline{L}^i = PS^i \cup RS^i \cup NS^i,\ PS^i \cap RSi = \emptyset,\ PS^i \cap NSi = \emptyset, \text{ and } RS^i \cap NSi = \emptyset. \qquad (3.31)$$

Expression (3.31) in conjunction with equality (3.30) establishes

$$\sum_{l \in \overline{L}^{i-1}} (|C_l^{i-1}| - 1) + |S^i| = \sum_{l \in \overline{L}^i} (|C_l^i| - 1),$$

which completes the proof. ■

The iterative process of kite chain growth is as follows. In addition, an example of kite chain growth is provided in Appendix B.

Step 1: INITIALIZATION.

Let K be the collection of all nodes in $k \geq 2$ kite frames. Start with $i = 1$, pick a kite frame to be scanned and name it K_1. Set $\overline{K}^1 = \{ K_1 \}$, $S^1 = \emptyset$, $\overline{S}^1 = \emptyset$, $L^1 = \emptyset$, $\overline{L}^1 = \emptyset$, and $i = 2$.

Step 2: LINKABLE KITE FRAME SELECTION.

Select a kite frame from $K\backslash\overline{K}^{i-1}$, the set of unscanned kite frames, linkable to at least one of the kite frames in $\overline{K}^{i-1}$, and name it K_i.

Step 3: CONNECTIVE NODESET SCANNING.

Set S^i = { all connective nodesets which chain K_i to $\overline{K}^{i-1}$ }. If there are $\overline{n}$ newly scanned connective nodesets, index them from $|\overline{S}^{i-1}|+1$ to $|\overline{S}^{i-1}|+\overline{n}$.

Set $\overline{S}^i = S^i \cup \overline{S}^{i-1}$.

Extract L^i and $\overline{L}^i$ from S^i and $\overline{S}^i$ respectively. Clearly, $\overline{L}^i$ = { 1, 2, ..., $|\overline{S}^i|$ }.

Set $RS^i = \overline{L}^{i-1} \cap L^i$, $NS^i = L^i \backslash (\overline{L}^{i-1} \cap L^i)$, and $PS^i = \overline{L}^i \backslash L^i$.

Step 4: COVERED KITE FRAMES CALCULATION.

Calculate $|C_l^i|$ for all $l \in \overline{L}^i$, using the following assignments.

$|C_l^i| = 2$, if $l \in NS^i$,

$|C_l^i| = |C_l^{i-1}| + 1$, if $l \in RS^i$,

$|C_l^i| = |C_l^{i-1}|$, for $l \in PS^i$.

Step 5: INEQUALITIES DEVELOPMENT.

If i = 2:

Inequality (3.32) is the kite chain inequality corresponding to the kite chain over kite frames K_1 and K_2 with $|\overline{S}^i|$ connective nodesets in $\overline{S}^i$.

$$\sum_{j=1}^{i} x(E(K_j)) \le \sum_{j=1}^{i} (|S(K_j)| - 1) - \sum_{l \in \overline{L}^i} (|C_l^i| - 1). \qquad (3.32)$$

Since all connective nodesets chaining K_1 and K_2 are newly scanned, $|C_l^i| = 2$, for all $l \in \overline{L}^i$. Thus, by Lemma 3.4, inequality (3.32) is a valid SGTSP inequality.

If i = k, go to Step 6.

Set i = i + 1.

Go to Step 2.

If i > 2:

The inequality (3.33) is valid based on the inequality (3.39) from the iteration i-1.

$$\sum_{j=1}^{i-1} x(E(K_j)) \le \sum_{j=1}^{i-1} (|S(K_j)| - 1) - \sum_{l \in \overline{L}^{i-1}} (|C_l^{i-1}| - 1). \qquad (3.33)$$

In addition, inequality (3.34) which is associated with kite frame K_i is valid by Corollary 3.1.

$$x(E(K_i)) \le |S(K_i)| - 1. \qquad (3.34)$$

Now, consider the $|S^i|$ connective nodesets in S^i, where each of which chains the kite frame K_i to the kite frames in $\overline{K}^{i-1}$. Recalling our notation described in part n of the notation and definitions, suppose the q^i+r^i nodes in $Q^i \cup R^i$ are part of the nodes included in a SGTSP tour. Therefore no edges in $E(K_1 \cup K_2 \ldots \cup K_{i-1})$ can be incident to any nodes in the q^i+r^i connective nodesets $S(Q^i \cup R^i)$. Thus, with regard to the inequality (3.33), inequality (3.35) is valid.

$$\sum_{j=1}^{i-1} x(E(K_j)) \le \sum_{j=1}^{i-1} (|S(K_j)| - 1) - \sum_{l \in \overline{L}^{i-1}} (|C_l^{i-1}| - 1) - (q^i + r^i). \qquad (3.35)$$

Meanwhile, because of participation of the p^i+r^i nodes in $P^i \cup R^i$ as part of the nodes passed by the SGTSP tour, no nodes in p^i+r^i connective nodesets of kite frame K_i can be incident to any edges in $E(K_i)$. These connective nodesets are $S(P^i \cup R^i)$. Therefore based on the preceding discussion, the inequality (3.36), as a modification of inequality (3.34), is valid.

$$x(E(K_l)) \leq |S(K_l)| - 1 - (p^i + r^i). \qquad (3.36)$$

Combining the inequalities (3.35) and (3.36) yields the valid inequality (3.37).

$$\sum_{j=1}^{i} x(E(K_j)) \leq$$

$$\sum_{j=1}^{i} (|S(K_j)| - 1) - \sum_{l \in \overline{L}^{i-1}} (|C_l^{i-1}| - 1) - (p^i + q^i + r^i) - r^i. \qquad (3.37)$$

Since $r^i \in Z_+^1$ and $|S^i| = p^i + q^i + r^i$, (3.38) follows from (3.37).

$$\sum_{j=1}^{i} x(E(K_j)) \leq \sum_{j=1}^{i} (|S(K_j)| - 1) - \sum_{l \in \overline{L}^{i-1}} (|C_l^{i-1}| - 1) - |S^i|. \qquad (3.38)$$

By Lemma 3.5, however, equality (3.31) is valid. To obtain the valid inequality (3.39) we add equalities (3.31) and (3.38).

$$\sum_{j=1}^{i} x(E(K_j)) \leq \sum_{j=1}^{i} (|S(K_j)| - 1) - \sum_{l \in \overline{L}^{i}} (|C_l^{i}| - 1). \qquad (3.39)$$

If i = k, go to Step 6.

Set i = i + 1.

Go to Step 2.

Step 6: GROWTH TERMINATION.

We come to this step if i = k. This implies that there exists no kite frame to be chained to the growing kite chain, thus the growth has to be stopped. Hence, by the arguments stated in the Step 5, the valid inequality (3.32) (or (3.39) when k > 2) is the kite chain inequality corresponding to our given kite chain with k kite frames and r connective nodesets. The expressions (3.32) or (3.39) can be rewritten as (3.40) for i = k.

$$\sum_{j=1}^{k} x(E(K_j)) \leq \sum_{j=1}^{k} (|S(K_j)| - 1) - \sum_{l \in \overline{L}^{k}} (|C_l^{k}| - 1). \qquad (3.40)$$

In view of the facts that $\overline{L}^k = \{ 1, 2, ..., r \}$, and $| \, | C_l^k | \, = \, | C_l | \,$, (3.40) appears as (3.41), which is the kite chain inequality.

$$\sum_{j=1}^{k} x(E(K_j)) \le \sum_{j=1}^{k} (|S(K_j)| - 1) - \sum_{l=1}^{r} (|C_l| - 1). \qquad (3.41)$$

This leads to the following theorem.

Kite Chain Growth Theorem. The kite chain inequality associated with a kite chain having k kite frames and r connective nodesets is a valid inequality for the SGTSP.

3.3.3 Interrelationship Between Kite Inequalities and Valid Inequalities for the Symmetric Traveling Salesman Polytope

The purpose of this section is to discuss the relationship between kite inequalities and a known class of TSP inequalities called *clique tree inequalities.* In addition to subtour elimination constraints, clique tree or general comb inequalities and their special cases are known valid inequalities for the symmetric traveling salesman polytope (see Nemhauser and Wolsey [1988] or Grötschel and Padberg [1985]). Special cases of clique tree inequalities include comb, simple comb, Chvátal comb and 2-matching inequalities.

We first provide some discussions of these inequalities and then present a comparative study between kite inequalities and the inequalities for the symmetric traveling salesman (STS) polytope.

3.3.3.1 Valid Inequalities for the STS Polytope

Valid inequalities corresponding to clique tree and its special cases are well studied inequalities for the STS polytope. Consider a graph $G = (V,E)$. A nonempty set C of vertices in V is called a *clique* if for every pair of vertices in C there is an edge connecting the vertices.

Clique tree. A collection of cliques is called a *clique tree* if the following properties are satisfied.

a. Cliques are partitioned into two nonempty sets, the set of h handles, denoted by $H_1, H_2, ..., H_h$, and the set of t teeth denoted by $T_1, T_2, ..., T_t$.

b. $T_i \cap T_j = \emptyset$, for $i \neq j$.

c. $H_i \cap H_j = \emptyset$, for $i \neq j$.

d. $2 \leq |T_j| \leq |V| - 2$, for all $j \in \{ 1, 2, ..., t \}$.

e. $|T_j| - t_j \geq 1$ for all $j \in \{ 1, 2, ..., t \}$, where t_j denotes the number of handles which intersect T_j.

f. The number of teeth that each handle intersects is odd and at least three.

g. The clique tree becomes disconnected if $H_i \cap T_j$, for $|H_i \cap T_j| > 0$ is deleted from it. In other words, there is no cycle which is made of cliques.

Figure 3.15 illustrates an example of a clique tree. Each clique is depicted by an oval as a handle, or by a rectangle as a tooth.

Figure 3.16 shows two detailed examples. One of them, (a), is a clique tree with two handles and five teeth, and the other one, (b), consists of one handle and three teeth.

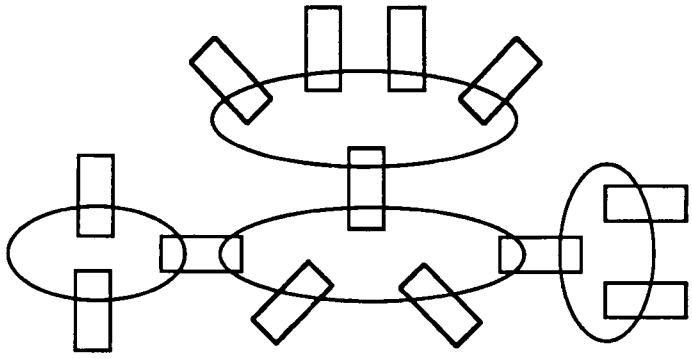
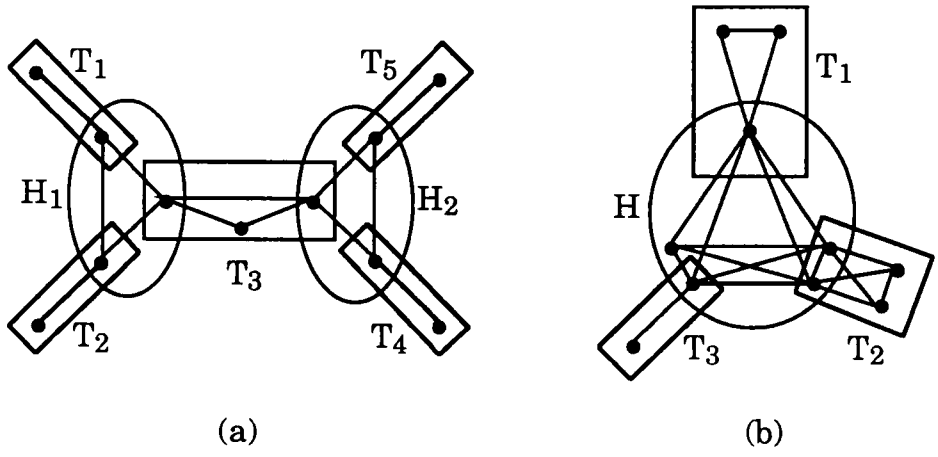


Figure 3.15: An example of a clique tree.



(a) (b)

Figure 3.16: Detailed examples of clique tree.

Clique Tree Inequality. The inequality (3.42) corresponds to a clique tree with h handles and t teeth.

$$\sum_{i=1}^{h} x(E(H_i)) + \sum_{j=1}^{t} x(E(T_j)) \le \sum_{i=1}^{h} |H_i| + \sum_{j=1}^{t} (|T_j| - t_j) - \frac{t+1}{2}. \qquad (3.42)$$

It has been proven that the clique tree inequality is valid for the STS polytope (see Nemhauser and Wolsey [1988] or Grötschel and Padberg [1985]).

Simple clique tree, comb, and simple comb are special cases of clique trees, and, consequently, their corresponding inequalities are special cases

of clique tree inequalities. A clique tree is simple if $|H_i \cap T_j| \leq 1$ for any handle H_i and tooth T_j. Figure 3.16-(a) shows a simple clique tree.

Comb Inequality. A clique tree with only one handle is called a *comb*. Likewise, a simple clique tree with only one handle is known as *simple comb* or *Chvátal comb*. In Figure 3.16-(a) the clique tree containing H_1, T_1, T_2, and T_3 is a Chvátal comb. In addition, the cliques H_2, T_3, T_4, and T_5 constitute another Chvátal comb. Figure 3.16-(b) shows a comb which is not simple. The inequality (3.43) is the comb inequality corresponding to a comb with t teeth.

$$x(E(H)) + \sum_{j=1}^{t} x(E(T_j)) \leq |H| + \sum_{j=1}^{t} (|T_j| - 1) - \frac{t+1}{2}. \qquad (3.43)$$

2-Matching Inequality. A 2-matching is a Chvátal comb with the condition that $|T_j| = 2$ for all $j \in \{ 1, 2, ..., t \}$, and $t \geq 1$. This condition ignores the property f of clique tree which expresses that t is odd and $t \geq 3$. The collection of H_2, T_4, and T_5 in Figure 3.16-(a) shows a 2-matching with two teeth. The inequality (3.44) is the 2-matching inequality corresponding to a 2-matching with t teeth.

$$x(E(H)) + \sum_{j=1}^{t} x(E(T_j)) \leq |H| + \left\lfloor \frac{t}{2} \right\rfloor. \qquad (3.44)$$

3.3.3.2 Comparative Structure of Kite Inequalities and Inequalities for the STS Polytope

The aggregated form of kite inequalities is the underlying tool employed to facilitate the comparative study of the kite inequalities and inequalities for the STS polytope. By aggregated form of an inequality we mean aggregated form of the partial graph associated with the left hand side of

the inequality. To recall the aggregated form over partial graphs, see section 1.2.3 and particularly Figure 1.4.

Simple Kite Inequality Versus 2-Matching Inequality. The aggregated form of any simple kite yields a figure analogous to a 2-matching with one tooth. This can be done by assuming the tail of simple kite as tooth, the kite frame as handle, and the node representing the connective nodeset as the node in common between tooth and handle. Figure 3.17 shows a simple kite and its aggregated form which resembles a 2-matching with one tooth. In this figure, only the edges with positive values are depicted, that is, in the simple kite $x_{ij} > 0$, and in the aggregated form $X_{IJ} > 0$.

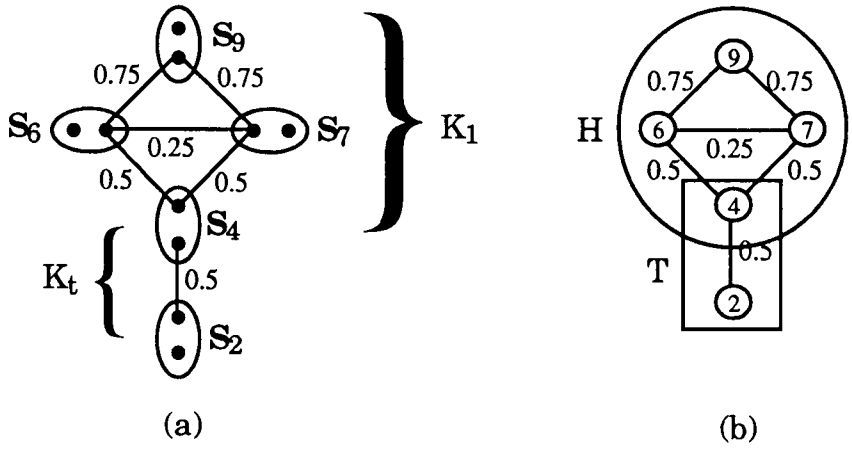


Figure 3.17: A simple kite, (a), and its aggregated form, (b).

Inequality (3.45) is the 2-matching inequality for a case of having one tooth T. Using aggregated form notation, (3.45) is the simplified form of (3.44) with t = 1.

$$X(E_a(H)) + X(E_a(T)) \leq |H|. \qquad (3.45)$$

The inequality corresponding to a simple kite with K_t, $|S(K_t)| = 2$, as the tail frame and K_1, $|S(K_1)| \geq 2$, as the kite frame is given by (3.46)

$$x(E(K_t)) + x(E(K_1)) \leq |S(K_1)| - 1. \qquad (3.46)$$

In our situation for study, we can assume that $|H| = |S(K_1)|$. Therefore, the right hand sides of both inequalities (3.45) and (3.46) are two different numbers. In addition, we can claim that there are differences between the left hand sides of the inequalities. These differences are due to nonhomogeneous variables in both left hand sides. Namely, X_{IJ}s are building blocks of (3.45), while (3.46) is composed of x_{ij}s elements.

For every X_{IJ} in the left hand side of (3.45) we substitute its equivalent

$$\sum_{(i;j)\in E(I;J)} x_{ij}$$

that indeed produces disaggregated form of (3.45). This substitution and the notion of *not-in-kite edges* help us to find differences in the left hand sides of (3.45) and (3.46).

Not-in-kite edges. Consider two kite frames Ki and Kj chained together by a connective nodeset S_{C_l} as a kite chain or part of a kite chain. With respect to these two kite frames consider a set of all edges having one of the two following properties:

a. An edge with one of its end nodes in $C_l(K_i)$ and the other end node in K_j.

b. An edge with one of its end nodes in $C_l(K_j)$ and the other end node in K_i.

We name this set as the set of *not-in-kite* edges between K_i and K_j connected by S_{C_l} and denote it by $E_{C_l}(K_i,K_j)$. Figure 3.18 displays an example of not-in-kite edges for the simple kite given earlier in Figure 3.17. In the example, connective nodeset S_{C_1} is equivalent to S_4, K_t is shown by hollow nodes, and K_1 is shown by solid nodes. The bold edges illustrate the not-in-kite edges $E_{C_1}(K_t,K_1)$.

In fact, the disaggregated form of (3.45) contains edge variables which are associated with all edges in the simple kite, $x(E(K_t \cup K_1))$, and edge variables

associated with not-in-kite edges in $E_{C_1}(K_t,K_1)$, $x(E_{C_1}(K_t,K_1))$. All thin and bold edges displayed in Figure 3.18 are edges associated with disaggregated form of the 2-matching example illustrated in Figure 3.17-(b).

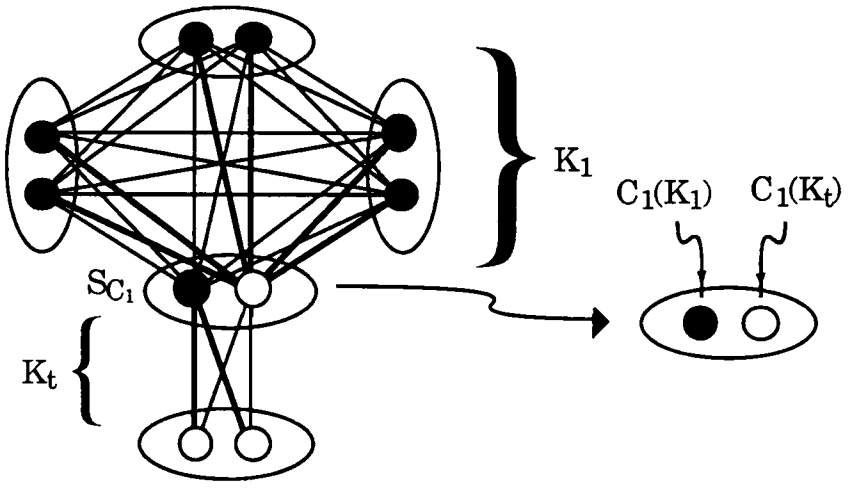


Figure 3.18: An example of not-in-kite edges for a simple kite.

The preceding discussion establishes that the left hand sides of (3.45) and (3.46) are not the same, and the extra terms in (3.45) are edge variables corresponding to the not-in-kite edges $E_{C_1}(K_t,K_1)$. The overall conclusion is that the simple kite inequality and disaggregated form of 2-matching inequality over aggregated form of the simple kite are two distinct inequalities for the SGTSP.

Kite Chain Inequality Versus Clique Tree Inequality. The aggregated form of kite chain might look like a clique tree, however, this is not always the case. In general, this is because of the restriction imposed by clique tree properties. For instance, consider a kite chain of three kite frames in aggregated form. Suppose the frames resemble a handle and two teeth, and some clique tree properties are satisfied. Certainly property f (of clique

tree) cannot be satisfied because of the even number of teeth in the aggregated form.

In the event that there is a resemblance between the aggregated form of a kite chain and a clique tree, the not-in-kite edges participate in disaggregated form of the clique tree, but are not included in the kite chain edges. The difference between the edges in a kite chain and the disaggregated form of a clique tree resembling the kite chain makes their corresponding inequalities different and distinct. Figure 3.19 shows a kite chain (each kite frame is contained within a shaded polygon) and its aggregated form which resemble a comb with three teeth.

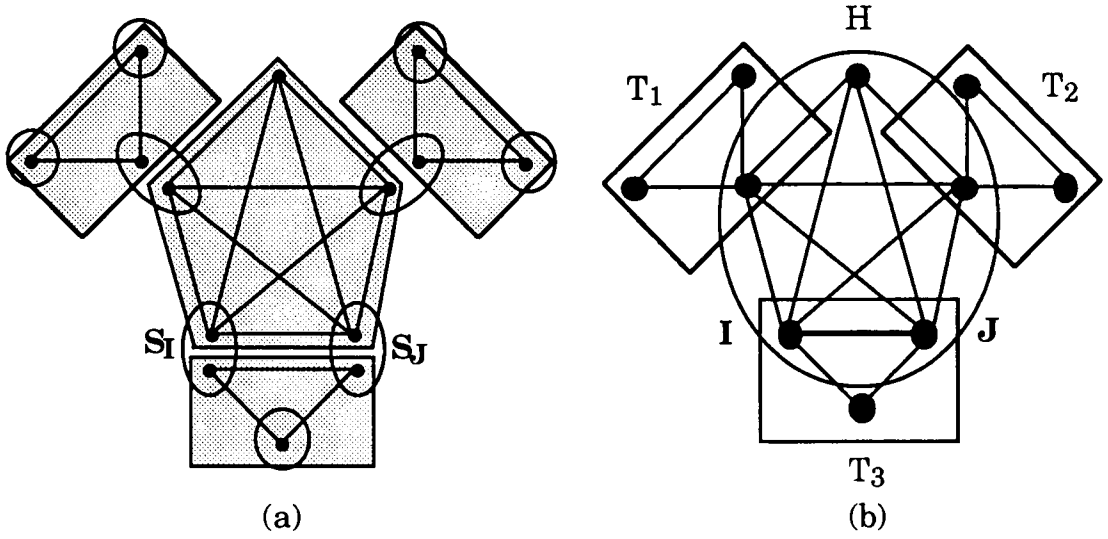


Figure 3.19: A kite chain and its aggregated form that resembles a comb.

Note that the coefficients on the left hand side of kite inequalities are one, while this is not always the case for clique tree inequalities. For example, any edge variable with a coefficient of 2 corresponds to an edge with both its end nodes in common between a handle H and a tooth T with $|H \cap T| \geq 2$.

In Figure 3.19-(b) $|H \cap T_3| \geq 2$, thus the coefficient of edge variable X_{IJ} is 2, and, as a result, the coefficients of edge variables associated with edges in E(I;J) are 2.

The result is that aggregated kite chain and clique tree inequalities have very little in common. In addition, applying clique tree inequalities over the aggregated form of the SGTSP bears little resemblance to kite chain inequalities. Figure 3.8, and the corresponding discussion at the beginning of section 3.3, implies that there are cases where no STSP inequality over the aggregated form of a GTSP graph is identifiable. For the example given in Figure 3.8, there are no identifiable generalized STSP inequalities, however, we can identify a violated kite chain inequality. The kite frames which make up the kite chain are shown in Figure 3.20. Kite frame K_1 is

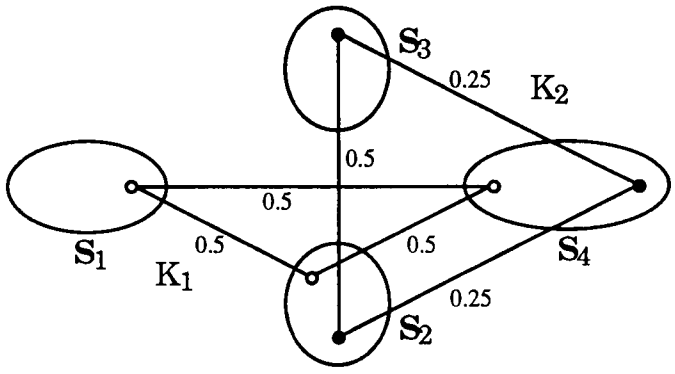


Figure 3.20: A kite chain associated with
the example given in Figure 3.8.

shown by the hollow nodes and K_2 is shown by the solid nodes. Nodesets S_2 and S_4 are the connective nodesets in the kite chain and it is easy to verify that its corresponding kite chain inequality is violated.

3.4 Inverse Transformation of ATSP Constraints

In this section, we discuss a fourth class of valid inequalities for the SGTSP. These inequalities can be generated by an inverse transformation of asymmetric TSP valid inequalities. Prior to inverse transformation, a P_1 solution of a SGTSP is transformed to an equivalent solution of an ATSP. Over the ATSP solution, we search for violated ATSP inequalities. We can show that, under certain conditions, an ATSP inequality can be transformed to a violated SGTSP constraint.

3.4.1 From SGTSP to AGTSP

In this section we show that how a P_1 solution of a SGTSP can be transformed to an equivalent solution of AGTSP, where the SGTSP and the AGTSP have equivalent node configurations.

In chapter 1, we introduced the formulation (P) for the SGTSP. For ease of discussion, we have rewritten this formulation below with slightly modified notation. Problem (P) is a SGTSP canonical formulation over a graph $\mathcal{G} = (\mathcal{V},\mathcal{E})$ of m nodesets S_I for $I \in \{ 1, 2, ..., m \}$, with $\mathcal{V} = \{ 1, 2, ..., n \}$, $\mathcal{E} = \{ (i;j) \mid i \in \mathcal{V}, j \in \mathcal{V}, i < j \}$, and a cost vector $c = \{ c_{ij} \mid (i;j) \in \mathcal{E} \}$.

Consider an asymmetric GTSP canonical digraph $\mathcal{D} = (\mathcal{N}, \mathcal{A})$ of m nodesets S_I for $I \in \{ 1, 2, ..., m \}$, with $\mathcal{N} = \{ 1, 2, ..., n \}$, $\mathcal{A} = \{ (i,j) \mid i \in \mathcal{N}, j \in \mathcal{N}, i \neq j \}$, and a cost vector $\vec{c} = \{ \vec{c}_{ij} \mid (i,j) \in \mathcal{A} \}$. Let $\vec{d}(i)$ be the set of all arcs in $\mathcal{A}$ that are incident *into* node $i \in \mathcal{N}$. Let $\overleftarrow{d}(i)$ be the set of all arcs in $\mathcal{A}$ that are incident *out of* node $i \in \mathcal{N}$. Let $\vec{D}(I)$ define the set of all arcs in $\mathcal{A}$ that are

incident into nodes in S_I, and $\overleftarrow{D}(I)$ defines the set of all arcs in $\mathcal{A}$ that are incident out of the nodes of nodeset S_I. For $W \subseteq \mathcal{N}$, A(W) defines the set of arcs with both end nodes in W, that is, $A(W) = \{ (i,j) \mid (i,j) \in \mathcal{A}, i \in W, j \in W \}$.

$$(P): \quad Z(x) = \text{Minimize} \sum_{(i,j) \in \mathcal{E}} c_{ij} x_{ij} \qquad (P.1)$$

Subject to,

$$x(D(I)) = 2 \qquad \forall\ I \in M = \{ 1, 2, \ldots, m \} \qquad (P.2)$$

$$x(NC'_{iJ}) - x(NC_{iJ}) \geq 0 \qquad \text{for all } i \in \mathcal{V},\ i \in S_I, \text{ and } J \in M \setminus \{I\} \qquad (P.3)$$

$$x(E(T)) \leq |S(T)| - 1 \qquad \forall \text{ subsets of node T, } 2 \leq |S(T)| \leq m-2 \qquad (P.4)$$

$$0 \leq x_{ij} \leq 1 \qquad \forall\ (i,j) \in \mathcal{E} \qquad (P.5)$$

$$x_{ij} \text{ integer} \qquad \forall\ (i,j) \in \mathcal{E}. \qquad (P.6)$$

Let y_{ij} be the arc variable associated with an arc (i,j) and let $\bar{y}_{ij}$ be the arc value of the arc (i,j) in a given solution. For a given set of arcs, F, let y(F) and $\bar{y}(F)$ be abbreviations for $\sum_{(i,j) \in F} y_{ij}$ and $\sum_{(i,j) \in F} \bar{y}_{ij}$ respectively. Problem $(\vec{P})$ represents an asymmetric GTSP canonical formulation over $\mathcal{D}$.

$$(\vec{P}): \quad \vec{Z}(y) = \text{Minimize} \sum_{(i,j) \in \mathcal{A}} \vec{c}_{ij} y_{ij} \qquad (\vec{P}.1)$$

Subject to,

$$\left.\begin{aligned} \vec{y}(\vec{D}(I)) &= 1 \\ \vec{y}(\overleftarrow{D}(I)) &= 1 \end{aligned}\right\} \qquad \forall\ I \in M = \{ 1, 2, \ldots, m \} \qquad (\vec{P}.2)$$

$$\vec{y}(\vec{d}(i)) - \vec{y}(\overleftarrow{d}(i)) = 0 \qquad \text{for all } i \in \mathcal{N} \qquad (\vec{P}.3)$$

$$y(A(T)) \le |S(T)| - 1 \qquad \forall \text{ subset of nodes } T, \quad 2 \le |S(T)| \le m-2 \qquad (\vec{P}.4)$$

$$0 \le y_{ij} \le 1 \qquad \forall (i,j) \in \mathcal{A} \qquad (\vec{P}.5)$$

$$y_{ij} \text{ integer} \qquad \forall (i,j) \in \mathcal{A}. \qquad (\vec{P}.6)$$

By replacing each symmetric edge (i;j) in $\mathcal{G}$ with two directed arcs (i,j) and (j,i) referred to as *twin arcs*, transforms $\mathcal{G} = (\mathcal{V},\mathcal{E})$ to $\mathcal{D} = (\mathcal{N}, \mathcal{A})$, where $|\mathcal{A}| = 2|\mathcal{E}|$. Defining a cost vector, $\vec{c} = \{ \vec{c}_{ij} \mid (i,j) \in \mathcal{A}, \vec{c}_{ij} = \vec{c}_{ji} = c_{ij} \}$, associated with arcs in $\mathcal{D}$ is the last step to make $\mathcal{D}$ equivalent to $\mathcal{G}$. Let $(\vec{P})$ be the GTSP canonical formulation over the newly modified $\mathcal{D}$.

Define (P^1) as the following formulation over the GTSP graph $\mathcal{G}$,

$$(P^1): \qquad Z^1(x) = \text{Minimize} \sum_{(i;j) \in \mathcal{E}} c_{ij} x_{ij}$$

Subject to, (P.2), (P.5), and (P.7),

where (P.7) is a set Π of v SGTSP valid inequalities that have been identified for $\mathcal{G}$. We assume that (P.7) includes all violated DP constraints.

$$\Pi = \{ (\pi^1,\pi_\circ^1), (\pi^2,\pi_\circ^2), \ldots, (\pi^v,\pi_\circ^v) \}. \qquad (P.7)$$

There is a slight difference between (P_1), the formulation discussed in section 3.3.1, and (P^1). This difference is between notation used in expressions (3.3) and (P.7).

Let $\bar{x}_f = \{ \bar{x}_{ij} \mid (i;j) \in \mathcal{E} \}$ be a feasible solution for (P^1), yet infeasible for (P). Given $\bar{x}_f$, we want to obtain $\bar{y}_f = \{ \bar{y}_{ij} \mid (i,j) \in \mathcal{A} \}$, a feasible solution for $(\vec{P}^1)$, where $(\vec{P}^1)$ is the following formulation over the GTSP digraph $\mathcal{D}$.

$$(\vec{P}^1): \qquad \vec{Z}^1(y) = \text{Minimize} \sum_{(i,j)\in \mathcal{A}} \vec{c}_{ij} y_{ij}$$

$$\text{Subject to, } (\vec{P}.2), (\vec{P}.3), (\vec{P}.5).$$

Since every pair of arcs (i,j) and (j,i) in $\mathcal{D}$ is associated with an edge (i;j) in $\mathcal{G}$, the relationship (3.47) must hold for every pair of arc values $\bar{y}_{ij}$ and $\bar{y}_{ji}$ in $\bar{y}_f$.

$$\bar{y}_{ij} + \bar{y}_{ji} = \bar{x}_{ij}. \text{ for every } (i;j)\in \mathcal{E}, \bar{y}_{ij}, \bar{y}_{ji}\in \bar{y}_f. \qquad (3.47)$$

(3.47) guarantees that $\vec{Z}^1(\bar{y}_f) = Z^1(\bar{x}_f)$, but is not enough to guarantee that $\bar{y}_f$ satisfies all constraints in $(\vec{P}^1)$. Thus $\bar{y}_f$ must belong to $F = \{ \bar{y}\in \Re^{|\mathcal{A}|} \mid (\vec{P}.2), (\vec{P}.3), (\vec{P}.5), \text{ and } (3.47) \}$. Since $0 \le x_{ij} \le 1$ for every $(i;j)\in \mathcal{E}$, and by (3.47), $\bar{y}_{ij} + \bar{y}_{ji} = \bar{x}_{ij}$ for every $(i;j)\in \mathcal{E}$, F can be written as $F = \{ \bar{y}\in \Re^{|\mathcal{A}|} \mid (\vec{P}.2), (\vec{P}.3), (3.47), \bar{y} \ge 0 \}$ or simply $F = \{ \bar{y}\in \Re^{|\mathcal{A}|} \mid B\bar{y} = b, \bar{y} \ge 0 \}$, where B is the coefficient matrix associated with coefficients of $(\vec{P}.2)$, $(\vec{P}.3)$, (3.47). Lemma 3.6 shows that for any given $\bar{x}_f$, a solution $\bar{y}_f\in F$ always exists.

Lemma 3.6. For any given $\bar{x}$ feasible for (P^1), $F = \{ \bar{y}\in \Re^{|\mathcal{A}|} \mid B\bar{y} = b, \bar{y} \ge 0 \}$ is nonempty and has at least one extreme point.

Proof. Given $\bar{x}_f = \{ \bar{x}_{ij} \mid (i;j)\in \mathcal{E} \}$, a feasible solution for (P^1), let

$$\bar{y}_f = \{ \bar{y}_{ij} \mid \bar{y}_{ij} = \frac{\bar{x}_{ij}}{2}, \text{ and } \bar{y}_{ji} = \frac{\bar{x}_{ij}}{2}, \text{ for every } (i;j)\in \mathcal{E} \}. \qquad (3.48)$$

We need to first show that $\bar{y}_f\in \Re^{|\mathcal{A}|}$ satisfies all constraints in $F = \{ \bar{y}\in \Re^{|\mathcal{A}|} \mid (\vec{P}.2), (\vec{P}.3), (3.47), \bar{y} \ge 0 \}$, and hence $\bar{y}_f\in F$.

Since $\bar{x}_f$ is a feasible solution for (P^1), $\bar{x}_f$ satisfies (P.2). Constraint set (P.2) can be written as,

$$\sum_{(i;j)\in D(I)} \bar{x}_{ij} = 2, \forall\, I\in M, \qquad (3.49)$$

and then simplified to,

$$\sum_{(i;j)\in D(I)} \frac{\bar{x}_{ij}}{2} = 1. \ \forall\ I \in M \qquad (3.50)$$

Any two arcs (i,j) and (j,i) between nodes i and j in $\mathcal{D}$, correspond to an edge (i;j) between nodes in $\mathcal{G}$ with the same indices i and j. Thus, one can apply the same index set used in (3.50) for ($\vec{P}$.2) taking into considerations the arc directions. That is, for any arc with index (i,j) or (j,i) we use edge index (i;j) with regard to the direction of the arc. Therefore the following substitutions of indices for ($\vec{P}$.2) are valid.

$$\left.\begin{aligned} \bar{y}(\vec{D}(I)) &= \sum_{(i,j)\in \vec{D}(I)} \bar{y}_{ij} = \sum_{(i;j)\in D(I)} \bar{y}_{ij}. \\ \bar{y}(\overleftarrow{D}(I)) &= \sum_{(i,j)\in \overleftarrow{D}(I)} \bar{y}_{ij} = \sum_{(i;j)\in D(I)} \bar{y}_{ij}. \end{aligned}\right\} \forall\ I \in M = \{\, 1, 2, \ldots, m \,\} \qquad (3.51)$$

We derive the expressions in (3.52) from (3.51) with respect to (3.48) and (3.50).

$$\begin{aligned} \bar{y}(\vec{D}(I)) &= \sum_{(i;j)\in D(I)} \bar{y}_{ij} = \sum_{(i;j)\in D(I)} \frac{\bar{x}_{ij}}{2} = 1. \\ \bar{y}(\overleftarrow{D}(I)) &= \sum_{(i;j)\in D(I)} \bar{y}_{ij} = \sum_{(i;j)\in D(I)} \frac{\bar{x}_{ij}}{2} = 1. \end{aligned} \qquad \forall\ I \in M = \{\, 1, 2, \ldots, m \,\} \qquad (3.52)$$

Hence $\bar{y}_f$ satisfies ($\vec{P}$.2).

In $\mathcal{D}$, for every arc (i,j) incident into a node j, there is another arc (j,i) incident out of the node j. Also by (3.48), the arc values associated with these arcs are the same and are equal to $\frac{\bar{x}_{ij}}{2}$. Therefore, the substitution of arc values given by $\bar{y}_f$ in every node conservation flow constraint of ($\vec{P}$.3) yields zero. Hence, $\bar{y}_f$ satisfies ($\vec{P}$.3).

Clearly $\bar{y}_f$ satisfies (3.47) and the nonnegative constraints, $\bar{y} \geq 0$, because

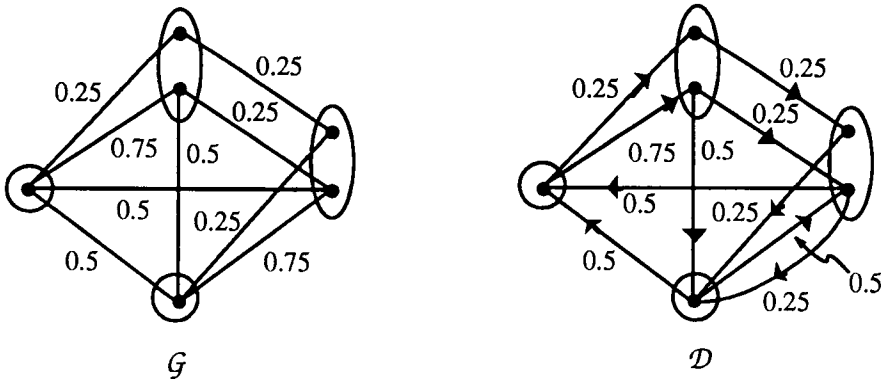
of (3.48) and the fact that $\bar{x}_f \geq 0$. Hence, we can conclude that $\bar{y}_f \in F$ and as a result F is nonempty.

We conclude that F has at least one extreme point. This follows from a fundamental theorem of linear programming that states if F is nonempty, then it has at least one extreme point (see Goldfarb and Todd [1989]). ■

The following corollary easily follows from the above lemma.

Corollary 3.2. For a given feasible solution $\bar{x}_f$ for (P^1), there exists at least one feasible solution $\bar{y}_f$ for $(\vec{P}^1)$.

Figure 3.21 shows an $\bar{x}_f$ solution on a GTSP graph $\mathcal{G}$, and a feasible $\bar{y}_f$ solution with respect to $\bar{x}_f$ on a GTSP digraph $\mathcal{D}$ equivalent to $\mathcal{G}$. Note that in $\bar{y}_f$, no arc value is equal to half of its corresponding edge value. This means that, in this example, for the $\bar{x}_f$ solution we have at least two feasible solutions for $(\vec{P}^1)$ associated with the digraph $\mathcal{D}$. One is shown in Figure 3.21-(b), and the other can be calculated in such a way that every arc value is equal to half of its corresponding edge value.



$\mathcal{G}$ $\mathcal{D}$

Figure 3.21: $\bar{x}_f$ and $\bar{y}_f$ solutions of a SGTSP

Let T be a function that maps $\Re^n$ into $\Re^{2n}$ in such a way that if $v = (v_1, v_2, ..., v_n)$, $0 \le v \le 1$, then $T(v) = (\frac{v_1}{2}, \frac{v_1}{2}, \frac{v_2}{2}, \frac{v_2}{2}, ..., \frac{v_n}{2}, \frac{v_n}{2}) = w$. We say w is the *image* of v under T, and write $w = T(v)$. Note that T is a one-to-one function.

Lemma 3.7. Given a feasible solution to (P^1), $\bar{x}_f$, over a GTSP graph $\mathcal{G}$, $\bar{y}_f = T(\bar{x}_f)$ is a feasible solution to $(\vec{P}^1)$ over a GTSP digraph $\mathcal{D}$ equivalent to $\mathcal{G}$.

Proof. This follows immediately from the proof of Lemma 3.6 and Corollary 3.2. ■

Lemma 3.8. Given $\bar{y}_f = T(\bar{x}_f)$, a feasible solution to $(\vec{P}^1)$ over the GTSP digraph $\mathcal{D} = (\mathcal{N}, \mathcal{A})$ described in Lemma 3.7,

i. $\bar{y}_f$ satisfies node flow conservation, $y(\vec{d}(i)) - y(\overleftarrow{d}(i)) = 0$, at every node $i \in \mathcal{N}$;

ii. $0 \le \bar{y}(\vec{d}(i)) \le 1$, $0 \le \bar{y}(\overleftarrow{d}(i)) \le 1$ for every node $i \in \mathcal{N}$;

iii. $\bar{y}(\vec{d}(i)) = 1$, $\bar{y}(\overleftarrow{d}(i)) = 1$, if $i \in S_I$ and $|S_I| = 1$.

Proof. i. This follows from constraints $(\vec{P}.3)$ of the problem $(\vec{P}^1)$, since $\bar{y}_f$ is a feasible solution to $(\vec{P}^1)$.

ii. This is in accordance with constraints $(\vec{P}.2)$ and $(\vec{P}.3)$ of the problem $(\vec{P}^1)$.

iii. If $i \in S_I$ and $|S_I| = 1$, $\vec{d}(i)$ is the set of arcs incident into the only node, i, in S_I, and $\overleftarrow{d}(i)$ is the set of arcs incident out of the node in S_I. Thus substitution of arc values in the constraints $(\vec{P}.2)$ yields

$$\left.\begin{aligned} \bar{y}(\vec{d}(i)) &= 1 \\ \bar{y}(\overleftarrow{d}(i)) &= 1 \end{aligned}\right\} \quad i \in S_I \text{ and } |S_I| = 1,\ I \in M;$$

hence the proof. ■

To summarize the previous results, any solution to a (P^1) formulation over a GTSP graph can be transformed, by way of function T, to a feasible solution to the ($\vec{P}^1$) formulation of a GTSP digraph equivalent to the GTSP graph.

3.4.2 From AGTSP to ATSP

In this section, we present an overview of how any canonical AGTSP can be transformed into a standard ATSP. We then show how a feasible solution to ($\vec{P}^1$) can be transformed into a feasible solution to ($\vec{P}^2$), where ($\vec{P}^2$) is a particular relaxation of an asymmetric TSP formulation ($\vec{P}^{TSP}$). Problem ($\vec{P}^{TSP}$) is formulated over digraph $\mathcal{D}^{TSP}$ which is an augmented and modified form of $\mathcal{D}$.

Transformation of Canonical form AGTSP to Standard ATSP. Noon and Bean [1989b] discussed an efficient transformation that can transform any problem formulated as AGTSP into a standard ATSP. In the following, we describe a method, adapted from a relevant part of the above citation, to transform $\mathcal{D}$ to $\mathcal{D}^{TSP}$.

For any nodeset S_I in $\mathcal{D}$ with $|S_I| = r_I > 1$, assume that its nodes have a given ordering. Let $i^1, i^2, ..., i^{r_I-1}, i^{r_I}$ be the nodes in S_I. Augment $\mathcal{D}$ by creating intraset arcs of the form $(i^1,i^2), (i^2,i^3), ...,(i^{r_I-1},i^{r_I}), (i^{r_I},i^1)$ for every nodeset S_I with $|S_I| > 1$, $I \in M$. Let this set of arcs be denoted as $\mathcal{A}^{intra}$, and let $\vec{c}^{intra}$ be the set of arc costs associated with the arcs in $\mathcal{A}^{intra}$, that is $\vec{c}^{intra} = \{ \vec{c}_{ij} \mid (i,j) \in \mathcal{A}^{intra} \}$. We define $A(S_I)$ as the set of all intraset arcs in a nodeset S_I, thus, $\mathcal{A}^{intra}$ can be written as $A(S_1) \cup A(S_2) ... \cup A(S_m)$. To modify $\mathcal{D}$, while

keeping arc costs and arc values intact, we shift backward the tail of every interset arc of the form $(i^k,j^l)\in \mathcal{A}$ by one, if $i^k\in S_I$ with $|S_I| > 1$. That is,

if $|S_I| = r_I > 1$,

- each arc $(i^k,j^l)\in \mathcal{A}$, $k > 1$, becomes arc $(i^{k-1},j^l)\in \mathcal{A}^{inter}$ with $\vec{c}_{i^{k-1}j^l} = \vec{c}_{i^kj^l}$ and $\bar{y}_{i^{k-1}j^l} = \bar{y}_{i^kj^l}$.
- each arc $(i^1,j^l)\in \mathcal{A}$, $k > 1$, becomes arc $(i^{r_I},j^l)\in \mathcal{A}^{inter}$ with $\vec{c}_{i^{r_I}j^l} = \vec{c}_{i^1j^l}$ and $\bar{y}_{i^{r_I}j^l} = \bar{y}_{i^1j^l}$.

and if $|S_I| = r_I = 1$,

- each arc $(i^1,j^l)\in \mathcal{A}$, becomes arc $(i^1,j^l)\in \mathcal{A}^{inter}$ with $\vec{c}_{i^1j^l} = \vec{c}_{i^1j^l}$ and $\bar{y}_{i^1j^l} = \bar{y}_{i^1j^l}$.

Therefore $\mathcal{D} = (\mathcal{N}, \mathcal{A})$ can be transformed into $\mathcal{D}^{TSP} = (\mathcal{N}, \mathcal{A}^{TSP})$, where $\mathcal{A}^{TSP} = \mathcal{A}^{intra}\cup\mathcal{A}^{inter}$.

Problem $(\vec{P}^{TSP})$ is an asymmetric TSP formulation over digraph $\mathcal{D}^{TSP}$.

$(\vec{P}^{TSP})$: $\vec{Z}^{TSP}(y) = \text{Minimize} \sum_{(i,j)\in \mathcal{A}^{TSP}} \vec{c}_{ij}y_{ij}$ $(\vec{P}^{TSP}.1)$

Subject to,

$\vec{y}(\vec{d}(i)) = 1$ for all $i\in \mathcal{N}$ $(\vec{P}^{TSP}.2)$

$\overleftarrow{y}(\vec{d}(i)) = 1$ for all $i\in \mathcal{N}$ $(\vec{P}^{TSP}.3)$

$0 \le y_{ij} \le 1$ for all $(i,j)\in \mathcal{A}$ $(\vec{P}^{TSP}.4)$

y_{ij} integer for all $(i,j)\in \mathcal{A}$. $(\vec{P}^{TSP}.5)$

y is a TSP tour $(\vec{P}^{TSP}.6)$

By relaxing the last two constraints of $(\vec{P}^{TSP})$, we obtain the new problem $(\vec{P}^2)$,

$$(\vec{P}2): \qquad \vec{Z}^{TSP}(y) = \text{Minimize} \sum_{(i,j)\in \mathcal{A}^{TSP}} \vec{c}_{ij} y_{ij}$$

Subject to,($\vec{P}^{TSP}$.2), ($\vec{P}^{TSP}$.3), and ($\vec{P}^{TSP}$.4).

Given $\bar{y}_f = \{ \bar{y}_{ij} \mid (i,j)\in \mathcal{A} \}$, a feasible solution to $(\vec{P}1)$, we want to construct $\bar{y}_f^{TSP} = \{ \bar{y}_{ij} \mid (i,j)\in \mathcal{A}^{TSP} \}$, a feasible solution to $(\vec{P}2)$. Arc values associated with intraset arcs in $\mathcal{D}^{TSP}$ play a key role in the construction of $\bar{y}_f^{TSP}$. In fact, the intraset arcs are the only arcs with no values assigned to them, thus assignment of arc values to the intraset arcs has to be done in such a way that $\bar{y}_f^{TSP}$ be a feasible solution to $(\vec{P}2)$.

Theorem 3.4. Given $\bar{y}_f$, a feasible solution to $(\vec{P}1)$, we can construct $\bar{y}_f^{TSP}$, a feasible solution to $(\vec{P}2)$.

Proof. Consider a nodeset S_I in $\mathcal{D}$ with $|S_I| = 1$. Transforming $\mathcal{D}$ to $\mathcal{D}^{TSP}$ has no effect on the arc structure in S_I, since it contains no intraset arcs no arcs are shifted. Therefore, by Lemma 3.8, part iii, constraints ($\vec{P}^{TSP}$.2) and ($\vec{P}^{TSP}$.3) are satisfied.

Let S_I in $\mathcal{D}$ be a nodeset with $r_I > 1$ nodes ordered as $i^1, i^2, ..., i^{r_I-1}, i^{r_I}$, and define $a^k = \bar{y}(\vec{d}(i^k))$ for all $k\in K = \{ 1, 2, ..., r_I \}$. By Lemma 3.8, parts i and ii, $a^k = \bar{y}(\overleftarrow{d}(i^k))$ and $0 \le a^k \le 1$, for any $i^k\in S_I$, $|S_I| > 1$, in $\mathcal{D}$. Due to the shifting of interset arcs, the transformation of $\mathcal{D}$ to $\mathcal{D}^{TSP}$ only affects $\bar{y}(\overleftarrow{d}(i^k))$ for any $i^k\in S_I$, $|S_I| > 1$, in $\mathcal{D}^{TSP}$. On the other hand, for any $i^k\in S_I$, $|S_I| > 1$, in $\mathcal{D}^{TSP}$ $\bar{y}(\vec{d}(i^k))$ remains unchanged. It is also assumed that $\bar{y}(\vec{d}(i^k))$ and $\bar{y}(\overleftarrow{d}(i^k))$ are excluded from any intraset arc values, this is because, no intraset arc incident into node i^k is considered to be in $\vec{d}(i^k)$ and no intraset arc incident

out of node i^k is considered to be in $\overleftarrow{d}(i^k)$. Thus, in $\mathcal{D}^{TSP}$, $\bar{y}(\overrightarrow{d}(i^k)) = a^k$, for all $k \in K$; $\bar{y}(\overleftarrow{d}(i^k)) = a^{k+1}$, for $k < r_I$, and $\bar{y}(\overleftarrow{d}(i^{r_I})) = a^1$.

To include arc values associated with intraset arcs (i^1,i^2), (i^2,i^3), $\ldots,(i^{r_I-1},i^{r_I})$, (i^{r_I},i^1) in S_I, let f^k, for $k \in K$, be the arc value associated with the intraset arc incident into the node i^k. It is clear that for any node $i^k \in S_I$, $\overrightarrow{d}(i^k)$ and (i^{k-1},i^k) are the arcs incident into the node i^k, and $\overleftarrow{d}(i^k)$ and (i^k,i^{k+1}) are the arcs incident out of the node i^k where we implicitly assume compatibility of indices. Based on the preceding, we can determine arc values associated with the intraset arcs in such a way that the constraints $(\vec{P}^{TSP}.2)$ and $(\vec{P}^{TSP}.3)$ be satisfied.

The solution of the following system of r_I linear equations in r_I unknowns, $f^1, f^2, \ldots, f^{r_I}$, yields the arc values associated with intraset arcs in a nodeset S_I with $|S_I| > 1$ over $\mathcal{D}^{TSP}$.

$$(1) \quad a^1 + f^1 = 1$$

$$(2) \quad a^2 + f^2 = 1$$

$$\vdots$$

$$(r_I) \quad a^{r_I} + f^{r_I} = 1$$

By assuming that $\bar{y}(\overrightarrow{d}(i^k))$ and $\bar{y}(\overleftarrow{d}(i^k))$ are arc values associated with the entering arcs incident into node i^k and leaving arcs incident out of node i^k, respectively, the above system can be interpreted as follows.

(1) and (2) stand for $\bar{y}(\vec{d}(i^1)) = 1$ and $\bar{y}(\overleftarrow{d}(i^1)) = 1$,

(2) and (3) stand for $\bar{y}(\vec{d}(i^2)) = 1$ and $\bar{y}(\overleftarrow{d}(i^2)) = 1$,

.

.

.

(r_I-1) and (r_I) stand for $\bar{y}(\vec{d}(i^{r_I-1})) = 1$ and $\bar{y}(\overleftarrow{d}(i^{r_I-1})) = 1$,

(r_I) and (1) stand for $\bar{y}(\vec{d}(i^{r_I})) = 1$ and $\bar{y}(\overleftarrow{d}(i^{r_I})) = 1$, respectively.

From the system, we can conclude that $f^k = 1 - a^k$ for $k \in K$, and $0 \le f^k \le 1$ since $0 \le a^k \le 1$. Hence, $\bar{y}_f^{TSP}$ is a feasible solution for $(\vec{P}2)$. ■

Lemma 3.9. Given $\bar{y}_f^{TSP}$, a feasible solution for $(\vec{P}2)$, the summation of arc values associated with intraset arcs in a nodeset S_I, $|S_I| = r_I > 1$, is equal to $|S_I| - 1$.

Proof. As in the proof of Theorem 3.4, the following system of equations is associated with the nodes in S_I.

$$a^1 + f^1 = 1,$$

$$a^2 + f^2 = 1,$$

.

.

.

$$a^{r_I} + f^{r_I} = 1.$$

The sum of all the equations yields (3.53)

$$\sum_{k=1}^{r_I} a^k + \sum_{k=1}^{r_I} f^k = r_I. \qquad (3.53)$$

It follows from the definition of a^k that,

$$\sum_{k=1}^{r_I} a^k = \bar{y}(\vec{d}(i^1)) + \bar{y}(\vec{d}(i^2)) + \cdots + \bar{y}(\vec{d}(i^{r_I})).$$

Consequently,

$$\sum_{k=1}^{r_I} a^k = \bar{y}(\vec{D}(I)),$$

since $\vec{D}(I)$ is the set of all interset arcs incident into the nodes in S_I in $\mathcal{D}^{TSP}$. Furthermore, $\bar{y}(\vec{D}(I)) = 1$ by the constraints $(\vec{P}.2)$ of the problem $(\vec{P}^1)$ and the fact that arc values associated with interset arcs in $\mathcal{D}^{TSP}$ are the same as their corresponding arcs in $\mathcal{D}$. Thus,

$$\sum_{k=1}^{r_I} a^k = 1. \qquad (3.54)$$

Combining (3.53) and (3.54) leads to expression (3.55),

$$\sum_{k=1}^{r_I} f^k = r_I - 1 = |S_I| - 1, \qquad (3.55)$$

the desired result. ■

This lemma guarantees that no violated subtour elimination constraint exists over the nodes in any nodeset.

3.4.3 From ATSP Inequalities to SGTSP Inequalities

In the last two sections we showed that a feasible solution to (P^1), $\bar{y}_f$, can always be transformed to a feasible solution to $(\vec{P}^2)$, $\bar{y}_f^{TSP}$. Problem (P^1) is a linear program which can occur in a cutting plane procedure for the SGTSP, and $(\vec{P}^2)$ is an LP-based ATSP formulation which can be used to search for violated ATSP valid inequalities. We can use this

transformation from SGTSP into ATSP to identify violated TSP inequalities. The identified ATSP inequalities must then be transformed into SGTSP inequalities. In this section, we address the issue of transforming ATSP inequalities to SGTSP inequalities.

Suppose $\beta y \leq \beta_\circ$ is an identified ATSP valid inequality corresponding to the digraph $\mathcal{D}^{TSP}$. The inequality cannot be a subtour elimination constraint over nodes in any nodeset, since by Lemma 3.9, we are guaranteed the contrary. Assume that in the inequality no arc values associated with intraset arcs in $\mathcal{D}^{TSP}$ exist, or in case of existence of such arc values, their corresponding arcs are all the arcs in one or more arcsets $A(S_I)$, for $I \in M$. For the latter case, we adjust the inequality by dropping all the terms associated with intraset arcs from the left hand side, and subtract the equivalent from the right hand side.

At this point, we are left with $\beta' y \leq \beta'_\circ$, a valid inequality such that all of its left-hand-side terms correspond to interset arcs of $\mathcal{D}^{TSP}$. We then shift forward the tail index of any arc included in $\beta' y \leq \beta'_\circ$ by one. By this action we are transforming back to our original problem, the AGTSP. If all the included arcs in $\beta' y \leq \beta'_\circ$ are twin arcs, the pairs of twin arcs can be substituted by their corresponding edge. The end result is a valid inequality for the SGTSP. Let $\beta'' x \leq \beta''_\circ$ denote the final SGTSP inequality from this back transformation.

With regard to the above discussions, the two proceeding conditions are required for transforming $\beta y \leq \beta_\circ$ to $\beta'' x \leq \beta''_\circ$.

1. All terms associated with intraset arcs of any nodeset must be excluded from $\beta y \leq \beta_\circ$.

2. All terms in $\beta' y \leq \beta'_{\circ}$, the adjusted form of $\beta y \leq \beta_{\circ}$, must be associated with twin arcs.

Suppose we have an instance of $\bar{x}_f$ such that its transformation into $\bar{y}_f$ does not require introducing twin arcs as interset arc. That is, by giving direction to edges in $\mathcal{G}$ they become arcs in $\mathcal{D}$ with node flow conservation constraints being satisfied automatically. In such a case, satisfying the condition 2 for back transformation is unnecessary, since every term in $\beta' y \leq \beta'_{\circ}$ is associated with an interset arc (i,j) that is equivalent to its corresponding edge (i;j). In Appendix C, we show a complete example of this special case.

CHAPTER 4

CUTTING PLANE PROCEDURE

4.1 Introduction

A cutting plane procedure is a method for computing bounds on the optimal objective value of an integer linear program. For a given integer linear program P, the procedure progresses by successively obtaining tighter LP descriptions of the problem P. At each step, the procedure achieves a tighter LP description by adding violated valid inequalities, also known as *cutting planes*. Assuming an integer feasible solution to the problem exists, the procedure continues until either:

a: the solution to the LP description is feasible and, therefore, optimal for the integer program P, or

b: no tighter LP description can be found.

Let LP_i denote the LP description of P at iteration i. Cutting planes identified at any iteration i must cut off the optimal solution to LP_i, yet leave the feasible region of the problem P intact. The violated valid inequalities are named cutting planes for the reason that they cut off portions of the

feasible region of LP_i but leave the feasible region of P unaltered. Figure 4.1 depicts a graphical illustration of a general cutting plane procedure.

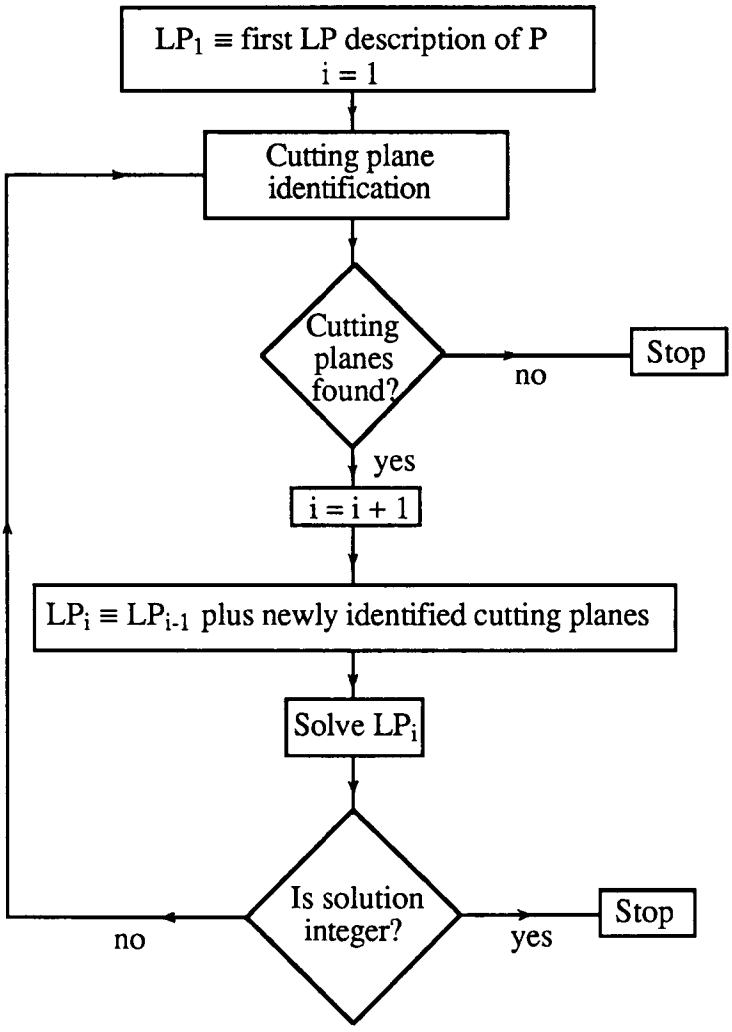


Figure 4.1: A graphical illustration of a general cutting plane procedure.

In the previous chapter we investigated several classes of valid inequalities for the symmetric GTSP. In this chapter, we aim to investigate implementation aspects of the studied valid inequalities in a cutting plane procedure framework.

Major components of a cutting plane procedure, usually, are identification routines for valid inequalities, upper bounding routines, and an LP solver. In this chapter, we present algorithms and routines for the identification of violated valid inequalities. The valid inequalities we identify are disjoint prevention constraints, generalized TSP subtour elimination constraints, and two special cases of kite constraints. In addition, four different combination heuristics to find feasible solutions of a SGTSP are presented. These combination heuristics are included in a routine called the upper bounding routine which tries to find good feasible solution to the SGTSP in order to provide an upper bound on the optimal objective value of a problem. We conclude this chapter with a discussion of our computational results.

4.2 Disjoint Prevention Contraint as a Cutting Plane

In the previous chapter we established that in a SGTSP the maximum number of DP constraints is n(m-1). Since the maximum number of these constraints is polynomial in the size of the problem one option would be to include the entire set of DP constraints in an initial LP description, and then to begin a cutting plane procedure. While this option would omit any searches for violated DP constraints, it would increase size of the LP description considerably. The other option would be not to include any DP constraints in the initial LP description and then identify and add violated DP constraints only as needed within a cutting plane procedure. An algorithm for identification of violated DP constraints is presented in the next section.

These two alternatives were compared computationally and the results are given in Appendix D. The results indicate that the alternative of appending DP constraints only as needed is computationally superior to the alternative of including the entire set of DP constraints in the initial LP description.

4.2.1 Routine to Identify Violated DP Constraints

The input of the following routine is a P_1 solution from a SGTSP with n nodes and m nodesets. The routine will find any DP constraints which are violated with respect to the input, the P_1 solution. These violated DP constraints are considered as the output of the routine.

For a P_1 solution of a problem, we refer nodes i and j as *involved nodes* if the value of variable x_{ij} is greater than zero. That is, $\overline{x}_{ij} > 0$. The set of involved nodes in a P_1 solution is denoted by INVN where $|INVN| = NINVN \leq n$. Let VDP be the set of violated DP constraints and DP_{iJ} defines the DP constraint over node i and nodeset S_J.

Step 0: Let the nodes in INVN be arbitrarily renamed and ordered, i = 1, ..., NINVN. Set $i_0 = 0$, $J_0 = 0$, VDP = ø, and k = 0 where $k = |VDP|$.

Step 1: Set $i_0 = i_0 + 1$. If $(i_0 = NINVN + 1)$, go to step 5.

Step 2: Set $J_0 = J_0 + 1$. If $(J_0 = m + 1)$, set $J_0 = 0$, go to step 1.

Step 3: If ($NC_{i_0J_0}$ exists), find $x(NC_{i_0J_0})$ and $x(NC'_{i_0J_0})$. Otherwise go to step 2

Step 4: If $(x(NC_{i_0J_0}) - x(NC'_{i_0J_0})) < 0$, add $DP_{i_0J_0}$ to VDP, set k = K + 1, go to step 2. Otherwise go to step 1.

Step 5: End.

In the above routine k gives the number of identified violated DP constraints and VDP stores their corresponding node/nodeset identifications. The routine is guaranteed to identify all DP constraints that are violated by a P_1 solution. This routine requires at most n(m-1) steps.

4.3 Generalized TSP Subtour Elimination Constraints

As was shown in Chapter 3, any constraint that is valid for symmetric TSP can be generalized to the SGTSP. In our cutting plane procedure, we search for violated subtour elimination (SE) constraints over aggregated form of a P_1 solution of a problem. Our rationale for selecting SE constraints among known valid inequalities for the TSP is that these constraints are known facets of traveling salesman polytope and can be identified by a polynomial algorithm requiring at most $O(m^4)$ steps (see Padberg and Grötschel [1985]). Note that m is the number of nodesets in a SGTSP and also the number of nodes in its aggregated form.

4.3.1 Routine to Identify Violated SE Constraints

This routine searches for violated SE constraints over the aggregated form of a P_1 solution. If there exists a violated SE constraint in the aggregated form of the P_1 solution, the routine will find it.

The core of the polynomial algorithm for identifying SE constraints is a minimum capacity cut problem routine. Let $\bar{x}$ be a P_1 solution of a problem and $\overline{X}$ be its aggregated form. Consider the graph $\mathcal{G}_a = (\mathcal{V}_a, \mathcal{E}_a)$ associated

with the aggregated form of the problem, and let the value $\overline{X}_{IJ}$, I,J$\in$ M = {1, ..., m} define a *capacity* associated with the edge (I;J) for every (I;J)$\in \mathcal{E}_a$. A subtour elimination constraint associated with a set of nodes $W_a \subseteq \mathcal{V}_a$ is violated if the cutset corresponding to W_a and $\mathcal{V}_a \backslash W_a$ has a capacity less than two.

To identify the minimum cutsets with capacity less than two, we use a polynomial algorithm described in Padberg and Rinaldi [1988]. The worst case bound of the algorithm is $O(m^4)$, however, it known to be empirically superior to the standard algorithm for finding the minimum capacity cut due to Gomory and Hu [1961]. Figure 4.2 shows the flow diagram of the routine for identifying SE constraints.

4.4 Identification of Violated Kite Constraints

In this section we present heuristic algorithms for identifying two special cases of kite constraints. The first heuristic algorithm searches for violated simple kite (SK) constraints with the following specification. The tail frame associated with each simple kite consists of all nodes of a node cone iJ with node i in the connective nodeset of the kite; that is, the algorithm searches for violated constraints associated with a special case of simple kites. The second heuristic algorithm searches for violated kite chain (KC) constraints consists of two kite frames having one or more connective nodesets. Note that, violated KC constraints identified by this heuristic algorithm are associated with a special case of kite chain.

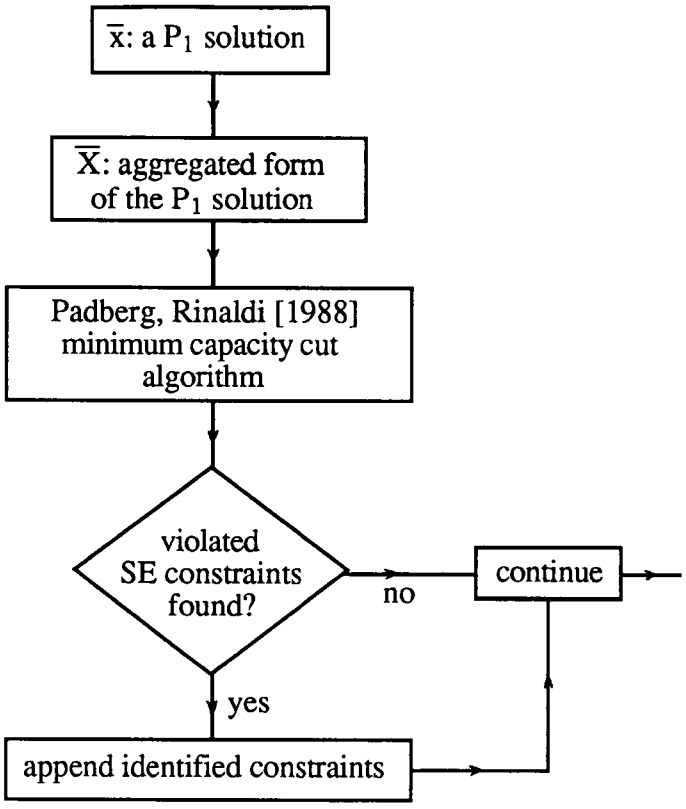


Figure 4.2: The flow diagram of the routine for identifying violated SE constraints.

4.4.1 Minimal Acceptable Covering Nodes Problem

We study the minimal acceptable covering nodes (MACN) problem because of its practical use in the development of heuristics for identification of violated kite constraints.

For a given graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, we define the minimal acceptable covering node problem as the problem of selecting the minimum cardinality subset of nodes $W \subseteq \mathcal{V}$ such that certain conditions are satisfied.

We use a heuristic algorithm to solve the MACN problem in order to find a kite frame K_i with a small number of covering nodesets and a large value corresponding to $\bar{x}(E(K_i))$. Now, we introduce the following heuristic

algorithm that finds a small cardinality subset of nodes in a graph with respect to a collection of conditions.

4.4.1.1 Minimal Acceptable Covering Nodes Algorithm

This algorithm finds, heuristically, a small cardinality subset of nodes $W \subseteq \mathcal{V}$ such that C, a collection of conditions, is satisfied. The notation C[W] = true denotes that W satisfies the C conditions. Let S be the set of scanned nodes, then $\mathcal{V} \backslash S$ is the set of unscanned nodes.

Step 1: Initialization. Set $S = \emptyset$, $k = 1$, Minimal_Exists = false.

Step 2: Selection of Starting Node. Pick a node $i_k \in \mathcal{V}$ as the first scanned node. Set $S = \{i_k\}$, $k = k + 1$.

Step 3: Choosing a Node to be Scanned. Select a node, i_k, from $\mathcal{V} \backslash S$.

Set $S = S \cup \{i_k\}$. It follows that $|S|$ is equal to k.

If (C[S] = true), set Minimal_Exists = true, go to step 4.

If ($|S| < |\mathcal{V}|$), set $k = k + 1$, go to step 3. Otherwise, go to step 4.

Step 4: Termination. If (Minimal_Exists = true), the set S contains a small cardinality subset of nodes in $\mathcal{V}$ with respect to the collection of conditions C and the starting node i_1. Stop.

This algorithm is of order n^2 steps where n is the cardinality of set $\mathcal{V}$. In order to find the MACN of a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, $|\mathcal{V}| = n$ with respect to the conditions C, we need to apply the algorithm n times, once for every node as the starting node. The small cardinality subset of nodes is selected from the collection of at most n identified subsets.

4.4.2 Identifying Violated Simple Kite Constraints

We apply the heuristic MACN algorithm to identify violated kite constraints. For a simple kite, suppose that the tail frame, K_t, and the connective nodeset, S_{C_1}, of a simple kite is given. Using a modified version of MACN, we can identify a kite frame K_1 with a small number of covering nodesets, if the simple kite constraint associated with the kite frames K_1, K_t, and the connective nodeset S_{C_1} is violated.

For a P_1 solution of a SGTSP with m nodesets, assume we are given a kite frame K_t with two covering nodesets S_I and S_J, with S_I specified as the connective nodeset and $\bar{x}(E(K_t)) = w_t > 0$. We want to identify another kite frame, K_1, with $I \in S(K_1)$ such that the simple kite constraint associated with these kite frames is violated. That is, we search for a kite frame K_1 such that $x(E(K_t)) + x(E(K_1)) > |S(K_1)| - 1$.

In order to find K_1, we construct a graph with m-1 nodes. Each node of the graph corresponds to a nodeset of the SGTSP with the exception of nodeset S_J. In fact, we make an *adjusted aggregated form* of the GTSP graph. The new form is called adjusted because we do not include the following.

a. Nodeset S_J and any edges incident to the nodes in S_J.

b. Any edges that are incident to the nodes in $C_1(K_t) = S_{C_1} \cap K_t$.

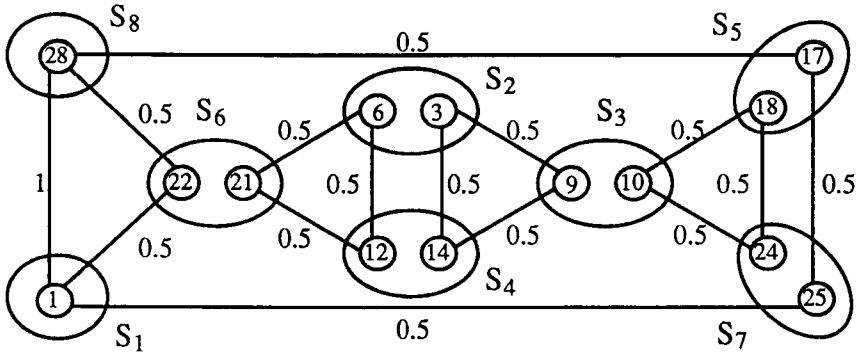
We construct the adjusted aggregated form graph which is similar to the aggregated form graph with the exclusion of the nodeset and the edges described in parts (a) and (b).

The above restrictions follow from our original definition of kite chain and the expression of kite chain inequality. We impose part (a) because

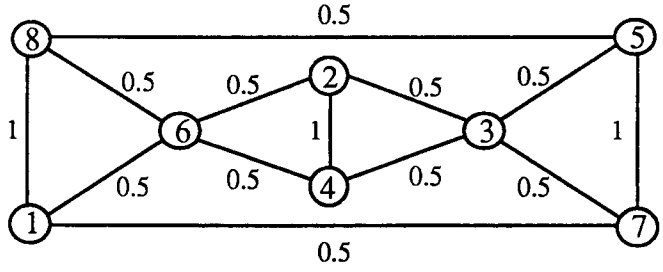
nodeset S_I must not be one of the covering nodesets of K_1. Without (a),K_t could have two connective nodesets out of two covering nodesets which is not allowed by part (b) of the properties of kite chain. Exclusion of edges that are incident to the nodes in $C_1(K_t)$, is enforced by the expression of the simple kite inequality. This enforcement comes from the fact that not-in-kite edges are excluded from participating in the left hand side of any kite inequality (see section 3.3.4.2 for discussion about not-in-kite edges). In this case the not-in-kite edges are edges in $E(K_t \cup K_1) \setminus \{E(K_t) \cup E(K_t)\}$.

Figure 4.3 presents an example of the above discussion. Figure 4.3-(a) shows a P_1 solution taken from a SGTSP with 30 nodes 8 nodesets. Note that the figure only shows the involved nodes of the P_1 solution. In this problem $S_1 = \{1\}$, $S_2 = \{2, 3, 4, 5, 6\}$, $S_3 = \{7, 8, 9, 10\}$, $S_4 = \{11, 12, 13, 14\}$, $S_5 = \{15, 16, 17, 18\}$, $S_6 = \{19, 20, 21, 22\}$, $S_7 = \{23, 24, 25, 26\}$, and $S_8 = \{27, 28, 29, 30\}$. Depicted in Figures 4.3-(b) and 4.3-(c), respectively, are the aggregated form of the P_1 solution and the adjusted aggregated form with respect to a kite(tail) frame $K_t = \{2, 3, 4, 5, 6, 21\}$, where $S(K_t) = \{\mathbf{2}, \mathbf{6}\}$ and $S_6 \equiv S_{C_1}$ as the connective nodeset. Note that, in the adjusted form graph, no aggregated edge exists between nodesets S_4 and S_6; this is because the edge (12;21) is considered as a not-in-kite edge and therefore is excluded. This is because the edge is incident to the node 21, is not in $E(K_t)$, and can not be in any $E(K_1)$.

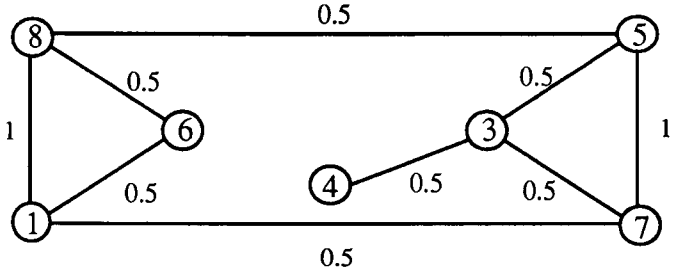
Consider a P_1 solution of a GTSP graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. Let K_t be a tail frame in the solution and $\mathcal{G}_{ad} = (\mathcal{V}_{ad}, \mathcal{E}_{ad})$ be the adjusted aggregated form graph



(a) a P_1 solution



(b) aggregated form



(c) adjusted aggregated form with respect to K_t

Figure 4.3: A P_1 solution, its aggregated form and its adjusted aggregated form with respect to prespecified K_t with $S(K_t) = \{\mathbf{2}, \mathbf{6}\}$ and connective nodeset S_6.

with respect to a kite frame K_t. We use a modified MACN algorithm to find K_1 such that

$$\bar{x}(E(K_t)) + \bar{x}(E(K_1)) > |S(K_1)| - 1, \text{ or equivalently}$$

$$w_t + \bar{x}(E_{ad}(S)) > |S| - 1,$$

where $S \subseteq \mathcal{V}_{ad}$ is the set of scanned nodesets.

In the following modified MACN algorithm let $\mathcal{V} \equiv \mathcal{V}_{ad}$, and set C[S] = true if $\frac{w_t + \bar{x}(E_{ad}(S))}{|S| - 1} > 1$. The algorithm might find a violated simple kite constraint, if one exists, with the following specifications.

1. K_t is the tail frame and K_1 is the other kite frame.
2. $S(K_t) = \{I, J\}$.
3. S_J is the connective nodeset, i.e. $S_{C_1} = S_J$ and $J \in S(K_1)$.

Algorithm 4.1:

Step 1: Initialization. Set S = {J} as the starting nodeset, M = {1, ..., m}, SimpleKiteExists = false and $K_1 = \emptyset$.

Step 2: Choosing a Nodeset to be Scanned. From $M \setminus (S \cup \{I\})$, select a nodeset R whose inclusion in S results in the maximum increase in $\bar{x}(E_{ad}(S))$. Set $S = S \cup \{R\}$.

If (C[S] = true), set SimpleKiteExists = true, $K_1 = \bigcup_{L \in S} S_L \setminus (S_J \cap K_t)$, go to step 3.

If $(S = M \setminus \{I\})$, go to step 3, otherwise repeat step 2.

Step 3: Termination. If (SimpleKiteExists = true), K_1 is the identified kite frame. Stop.

Given a kite frame K_t with $S(K_t) = \{I, J\}$ for a P_1 solution, we can apply the algorithm 4.1 for every nodeset in $M \setminus \{I\}$ as the starting nodeset in S. If f distinct kite frames were identified, we could construct f distinct simple kites, $[K_1, K_t]$, ..., $[K_f, K_t]$, that their associated constraints are violated by the P_1 solution.

Applying algorithm 4.1 to the example depicted in Figure 4.3 yields $S = \{6, 8, 1\}$, $K_1 = S_1 \cup S_8 \cup S_6 \setminus \{21\}$, and

$$\bar{x}(E(K_t)) + \bar{x}(E(K_1)) = 0.5 + 2 = 2.5 \not\leq 2.$$

Note that the value of the right hand side of the above expression is 2, since $|S(K_1)| - 1 = 3 - 1 = 2$. The computational complexity of the algorithm is $O(m^2)$.

4.4.3 A Routine to Identify Violated Simple Kite Constraints

We use the modified MACN algorithm to identify violated simple kite constraints. The tail frame associated with each simple kite consists of all nodes participating in a node cone iJ and node i belongs to the connective nodeset of the simple kite. $E(i \cup S_J)$ gives the set of all edges in a node cone iJ. For a given P_1 solution, the nodes of any node cone iJ, $i \in S_I$ (with $\bar{x}(E(i \cup S_J)) > 0$) can constitute the tail frame of a potential violated simple kite constraint, if S_I (the designated connective nodeset) contains at least two involved nodes.

Let PTF be the set of all possible tail frames arbitrarily ordered from 1 to p. The following routine outlines the steps required for identification of violated simple kite constraints.

Step 1: Find the set PTF from a P_1 solution. Set $p = |PTF|$.

If (p = 0), stop.

Set k = 1.

Step 2: Pick the kth PTF.

Let S_I be the connective nodeset associated with the kth PTF with node cone iJ, $i \in S_I$. Set $K_t = \{i \cup S_J\}$.

Call algorithm 4.1 with I as the starting nodeset.

If (SimpleKiteExists = true), construct the constraint associated with K_t, K_1, and $S_{C_1} = S_J$. Append the identified constraint.

Set k = k + 1.

If ($k \leq p$), go to step 2, otherwise stop.

This procedure is of order m^2p steps where m is the number of nodesets.

4.4.4 A Routine to Identify Violated Kite Chain Constraints

The previous routine searches for violated kite constraints associated with a special case of simple kite. The objective of this routine is to search for violated kite constraints associated with a special case of kite chain. The kite chain associated with this special case consists of two kite frames (K_1 and K_2) and one or more connective nodesets.

In a given P_1 solution, consider a nodeset S_I with $k \geq 2$ involved nodes arbitrarily ordered from i_1 to i_k. The routine assigns one of these nodes to a kite frame, say K_1, with the remaining nodes of S_I assigned to the other kite frame, K_2. An algorithm similar to the MACN algorithm is used to identify K_1 with a predefined number of covering nodesets, τ, $\tau = |S(K_1)|$. The algorithm uses an adjusted aggregated form that excludes all edges incident to the nodes in S_I that are assigned to K_2. In other words, the

adjusted aggregated form excludes edges incident to nodes in $S_I \cap K_2$. In the algorithm, the starting nodeset is S_I and the stopping criterion is occurred when $|S| = \tau$.

The next step, after identifying K_1, is to determine a kite frame, say K_2, with respect to the kite frame K_1 such that the associated kite constraint is violated. In this segment, the routine employes the algorithm 4.1 over the adjusted aggregated form of the solution with respect to the nodes in K_1. The starting nodeset for the algorithm is S_I. If a violated kite chain constraint consisting of K_1 and K_2 is identified, the connective nodesets are easily identified. These connective nodesets are the nodesets that are in common in both covering nodesets of K_1 and K_2. Note that the nodeset S_I is necessarily one of the connective nodesets in the identified kite chain.

The routine assigns the value 3 to τ as the starting value of τ and begins to search for a violated kite chain constraint. If no violated kite chain constraint is found, τ is increased by one and another search begins. This step is repeated until a violated kite chain constraint is identified or τ reaches m-1. For the latter case, the routine concludes that is unable to identify a violated kite chain constraint. Figure 4.4 shows a flow diagram of the routine for a given nodeset S_I with $k \geq 2$ involved nodes $i_1, i_2, ..., i_k$. In order to identify as many violated kite chain constraints as possible in a P_1 solution, the routine is called for every nodeset with more than one involved node. The routine is of order m^3m' steps where m' is the maximum number of nodes in a nodeset.

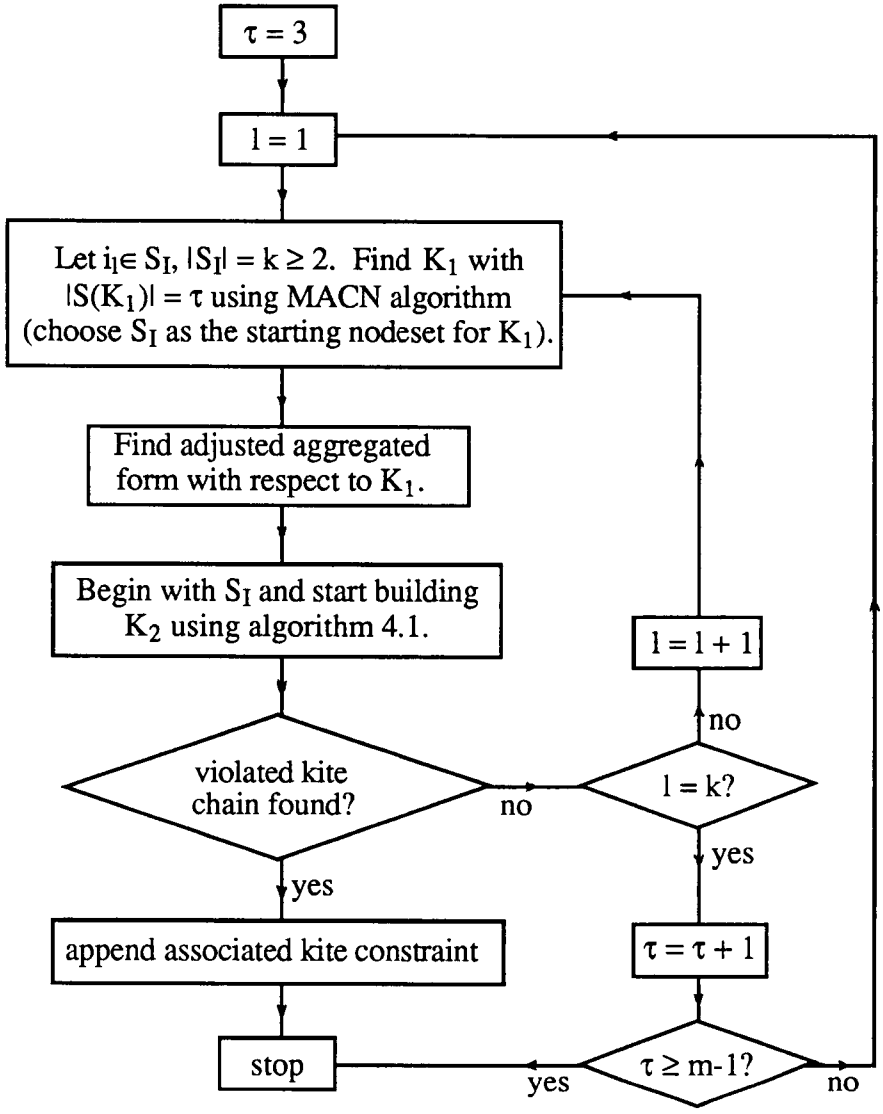


Figure 4.4: The routine to identify a violated kite chain constraint.

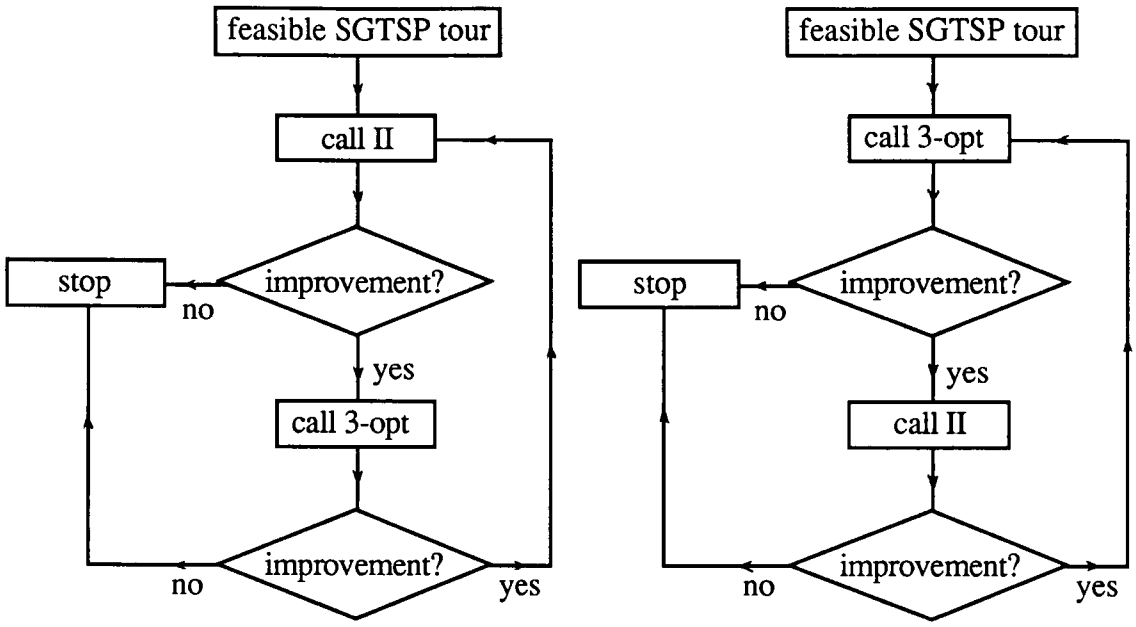
4.5 Heuristic Approaches for the SGTSP

In this section we present four combination heuristics for the SGTSPs. These heuristics will be utilized in an upper bounding routine for a cutting plane procedure. For a given problem, each combination heuristic first generates an initial solution then improves the solution. We discuss two heuristic tour construction algorithms, nearest neighbor and nearest insertion. The role of each tour construction algorithm is generating an initial solution for a given SGTSP. Each of these heuristics is combined with two improvement procedures. The function of each improvement procedure is to improve the initial solution. The difference between these improvement procedures is the order of calling two local improvement techniques.

The local improvement techniques were developed to improve the initial solutions. These techniques are improvement insertion (II) and an edge-exchange procedure by Lin and Kernigham [1973] known as 3-opt. The improvement techniques, II and 3-opt, are called together in the improvement procedures named Imp-Opt and Opt-Imp. In Imp-Opt, a given solution is improved first by II and then by 3-opt. This order of calling II and 3-opt is repeated until no improvement is achieved. The only difference between Opt-Imp and Imp-Opt is the order of calling II and 3-opt. Figure 4.5 shows the flow diagrams of Imp-Opt and Opt-Imp.

4.5.1 Nearest Neighbor

One of the tour construction heuristics we use is nearest neighbor. This



(a) Imp-Opt (b) Opt-Imp

Figure 4.5: Imp-Opt and Opt-Imp procedures.

heuristic is a slight modification of classic Nearest Neighbor for TSP. The heuristic builds a path from a starting node by successively adding the least cost edge incident to the last selected node in the path. The algorithm stops when one node from every nodeset is in the path. The path becomes a tour by adding the edge incident to the last selected node and the starting node. This routine is started from every one of the n nodes in a problem. The least cost tour among the resulting n tours is the output of the algorithm.

Let S(i) be the index of a nodeset S_I such that $i \in S_I$, that is, S(i) = I if $i \in S_I$. Define B as the set of all visited nodesets. Assume that all n nodes of a problem are arbitrarily ordered from 1 to n and let M = {1, ..., m} be the set of indices of all m nodesets. The set T of edges gives a partial or feasible tour, and the set T^* of edges represents the identified least cost tour with cost TourCost. The algorithm is as follows and for all edges of the form (i;j) the condition i < j is dropped.

Step 1: Set i = 0, TourCost = ∞, and $T^* = \emptyset$.

Step 2: Set $B = \emptyset$, $T = \emptyset$, cost = o, and i = i + 1.

If (i > n) stop.

Set $B = B \cup \{S(i)\}$.

Find the least cost edge (i;j) from the collection of all edges incident to the node i such that $j \notin S_J$, if $J \in B$.

Add (i;j) to T, set $B = B \cup \{S(j)\}$, and cost = cost + c_{ij}.

Step 3: Select the least cost edge (j;l) from the collection of all edges incident to the node j such that $l \notin S_J$, if $J \in B$.

Add (j;l) to T, set $B = B \cup \{S(l)\}$, and cost = cost + c_{jl}.

If (cost $\geq$ TourCost) go to step 2.

Set j = l. If ($|B| < m$) go to step 3.

Add (j;i) to T, and set cost = cost+c_{ji}.

if (cost < TourCost), set $T^* = T$, TourCost = cost. Go to step 2.

The output of the algorithm is the tour T^* with cost TourCost. In the worst case the algorithm requires $O(n^2m)$ steps.

4.5.2 Nearest Insertion

The second tour construction heuristic we use is nearest insertion. This heuristic begins with the least cost edge (i;j) as a partial tour [(i;j), (j;i)]. A node then selected from the set of unvisited nodesets in such a way that its insertion into the current partial tour increases the cost of the incomplete tour the least. This enlargement of the partial tour is continued until a complete tour with m edges is found.

The following outlines the required steps for the algorithm. The set B gives the indices of visited nodesets, and the set T contains the edges of a partial tour. The output of the algorithm is a feasible GTSP tour with the cost of TourCost. The algorithm requires on the order of n^2 computations.

Step 1: Initialization. Set $T = \emptyset$, $B = \emptyset$.

Find the least cost edge (i;j). Set $B = \{S(i), S(j)\}$.

Add (i;j) to T, and set TourCost = c_{ij}.

Step 2: Selection. Find a node k from $\bigcup_{I \in M \setminus B} S_I$ in such a way that if it is inserted between any two consecutive nodes p and q in T, $c_{pk} + c_{kq} - c_{pq}$ is minimized.

Step 3: Insertion. Set TourCost = TourCost + $c_{pk} + c_{kq} - c_{pq}$. Drop (p;q) from T and add (p,k) and (k;q) to T.

Set $B = B \cup \{S(k)\}$.

If ($|B| = m$), stop, otherwise go to step 2.

4.5.3 Improvement Insertion

Improvement insertion (II) is a tour improvement procedure that attempts to find a better tour given an initial tour. The goal of this procedure is to improve a given feasible tour by using one step removal and then one step insertion. Suppose T^* is a feasible initial tour with cost TourCost and edges (p;k) and (k;q) incident to node k are in the tour. Let T' be defined as the partial tour with node k removed, that is, $T' = (p;q) \cup T^* \setminus \{(p;k),(k;q)\}$, and let the cost of T' be cost = TourCost - c_{pk} - c_{kq} + c_{pq}. If $k \in S_k$ and k is removed from T^*, then S_k becomes an unvisited

nodeset. To make the partial tour T′ a feasible tour, one of the nodes in S_k must be inserted somewhere in T′ in such a way that the resulting cost of the newly formed tour is minimized. The procedure repeats removal and insertion steps for every node in T^* and then picks the least cost tour named T with cost LeastCost. If improvement occurs, that is LeastCost is less than TourCost, the procedure replace T^* with T and TourCost with LeastCost. The procedure continues until no improvement occurs.

The improvement insertion procedure is as follows. In the procedure $T^* = \{(i_1;i_2), (i_2;i_3), ..., (i_{m-1};i_m)\}$ is a given starting feasible tour with cost TourCost. Note that in step 3, if k = 1, index k-1 is equivalent to m, and if k = m, index k+1 is equivalent to 1.

Step 1: Set LeastCost = TourCost.

Step 2: Set k = 1, T′ = ø, cost = 0.

Step 3: Set $T' = (i_{k-1};i_{k+1}) \cup T^* \setminus \{(i_{k-1};i_k), (i_k;i_{k+1})\}$.

Set cost = TourCost + $c_{i_{k-1}i_{k+1}} - c_{i_{k-1}i_k} - c_{i_ki_{k+1}}$.

Choose the node l from $S_{S(i_k)}$ such that if l is inserted between any two consecutive nodes p and q in T′, the resulting tour cost is minimized.

Step 4: If (cost + $c_{pl} + c_{lq} - c_{pq}$ < TourCost), set $T = T' \cup \{(p;l), (l;q)\} \setminus \{(p;q)\}$, LeastCost = cost + $c_{pl} + c_{lq} - c_{pq}$, and go to step 5.

Set k = k + 1, T′= ø, cost = 0, go to step 3.

Step 5: If (k < m), set T′= ø, cost = 0, and go to step 3.

Step 6: If (LeastCost < TourCost), T^* = T, TourCost = LeastCost, and go to step 2, otherwise, stop.

The output of the procedure is a tour T^* with cost TourCost. This procedure is of order m^2m' steps where m' is the maximum number of nodes in a nodeset.

4.5.4 3-opt Improvement

This procedure utilizes the well known 3-opt heuristic by Lin and Kernigham [1973] for the TSP. A given SGTSP tour $T = \{(i_1;i_2), (i_2;i_3), ..., (i_{m-1};i_m)\}$ is improved over an m x m cost matrix associated with nodes in T in both ends.

Note that the order of nodesets in an improved tour is changed if the 3-opt procedure is used, while the nodes selected within each nodeset remain unchanged. This is notably different than when the improvement insertion routine is applied for tour improvement. In the case of II, the order of some nodesets and/or the selected nodes of nodesets of the starting tour might be changed.

4.5.5 The Upper Bounding Routine

The objective of this routine is to provide a good upper bound for a given SGTSP. The upper bound is later used in the cutting plane procedure which will be discussed in the next section. The routine is composed of the two heuristics, nearest neighbor and nearest insertion, for obtaining initial tours and the two improvement procedures, Imp-Opt and Opt-Imp, for improving the initial tours. The routine determines the least cost identified tour out of four improved tours. Figure 4.6 depicts the flow diagram of this

routine. The cost associated with the output tour T^* is an upper bound for the given SGTSP.

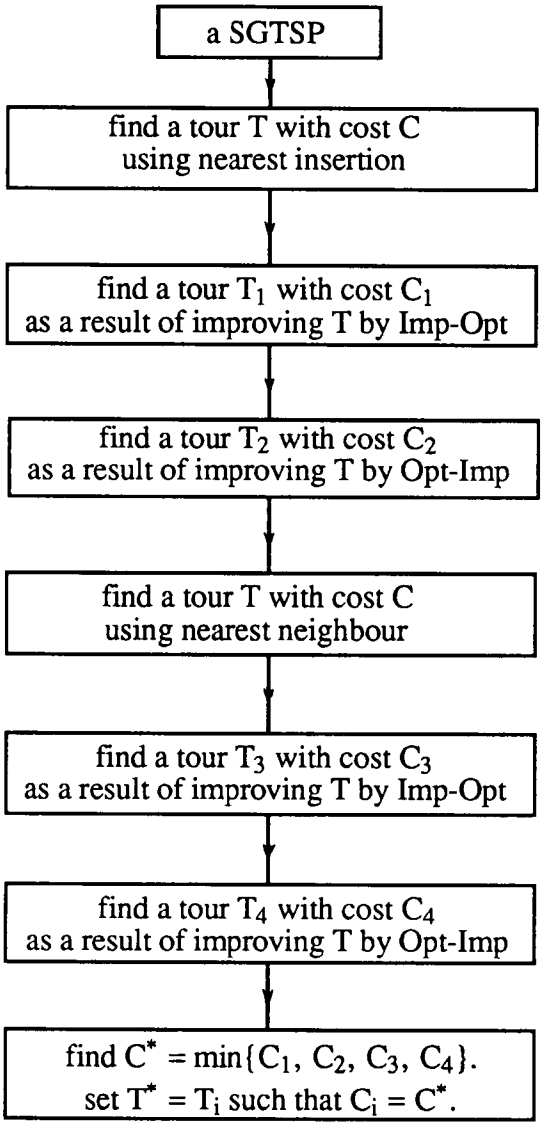


Figure 4.6: The procedure for finding a feasible upper bound tour.

A small computational study was performed which compared the four combination heuristics on the basis of solution cost. The details and results of the study are given in Appendix D. The results suggest that, on average the nearest neighbor/Imp-Opt combination performs better than the others.

To obtain the best possible result in the cutting plane procedure, however, we selected the least cost tour from the four available.

4.6 Cutting Plane Procedure

Two of the key elements of our cutting plane procedure (CPP) are the identification routines for finding violated constraints and the heuristic procedures for finding feasible solutions. These elements work together to tightly bound the optimal solution of a problem. The third key element of the CPP is an LP solver. The cutting plane procedure employs the LP solver for solving LP descriptions obtained during the process of the procedure.

The input to the CPP is a SGTSP with possibly some additional side constraints. The input allows for the construction of the first LP description known as the *initial description*. The initial description consists of the objective function, degree constraints, and [0,1] bounds on variables plus any additional constraints included in the original problem input. The output of the procedure is either the optimal solution of the input problem or the final lower bound identified by the CPP for it. Figure 4.7 shows the flow diagram for the CPP. The diagram shows the order in which the various identification routines are called. The sequence of calling identification routines for violated DP, SE, SK, and KC constraints outperformed other tested sequences in a preliminary computational study.

A preprocessing routine is used to reduce the size of the given problem. The routine employs an edge removal subroutine that attempts to eliminate edges that cannot be in an optimal solution of the problem. The edge removal subroutine utilizes a procedure called the arc/node elimination

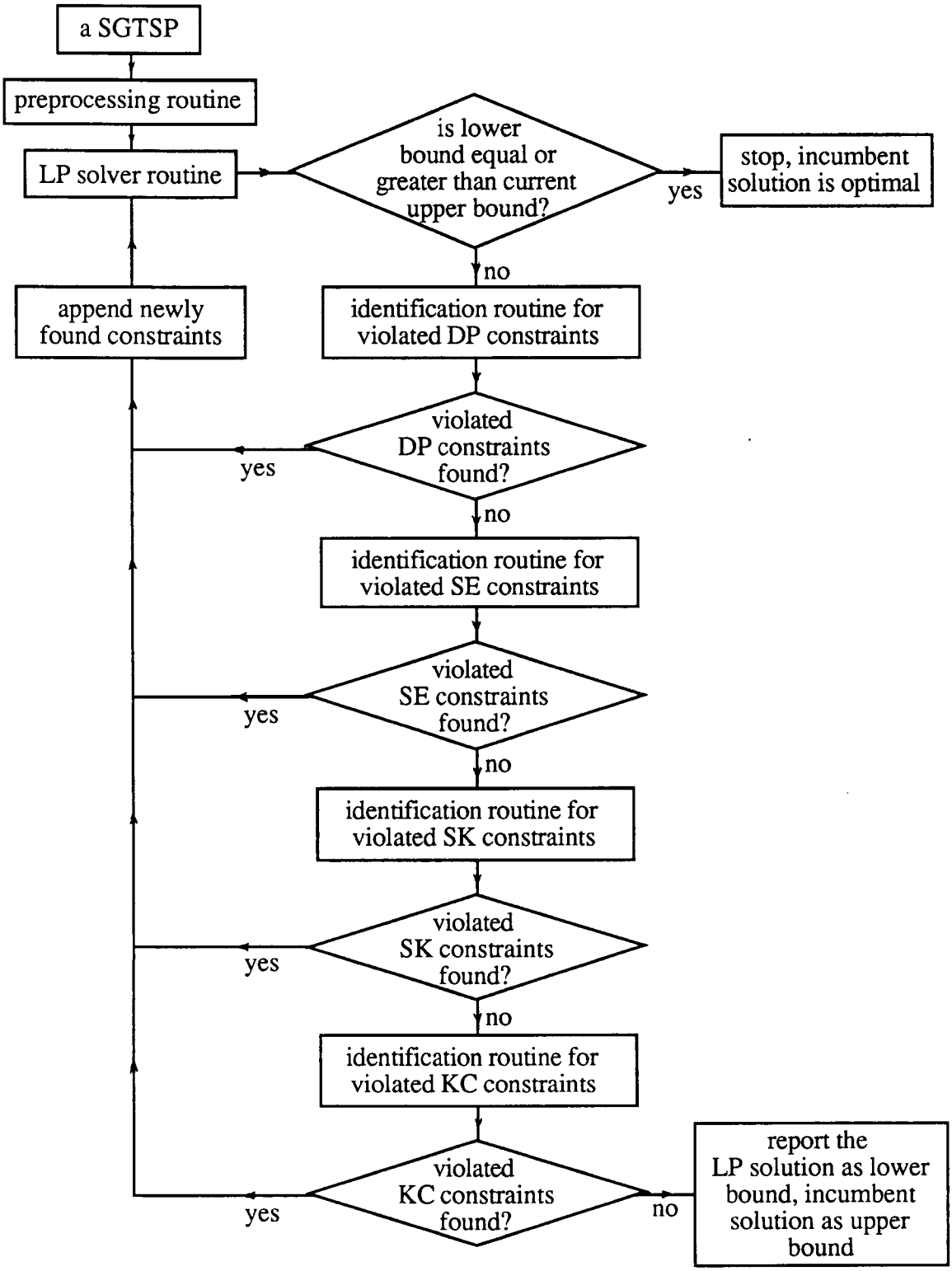


Figure 4.7: Flow diagram of the cutting plane procedure.

procedure. This procedure discussed in Noon [1988] and Noon and Bean [1989a], and was originally developed to reduce the size of asymmetric GTSPs. The output of the edge removal subroutine is a problem that has been reduced in size by fixing some of the variables to zero, where these variables are removed from the problem. The second objective of the preprocessing routine is to obtain an upper bound for the given problem. The upper bound is obtained by calling the upper bounding routine. Figures 4.8 and 4.9 provide flow diagrams of the preprocessing routine and the edge removal subroutine, respectively. In Figure 4.8, the value of the upper bound is denoted by UB, and in Figure 4.9, the abbreviation NE gives the number of edges left in the problem.

The primary purpose of the LP solver routine is to solve any LP description of the SGTSP. A secondary function of the LP solver routine is to heuristically search for new upper bounds, and, when possible, fix variables to zero or one. The assignment of zero or one to a variable is based on two values, the reduced (or relative) cost of the variable and the current *duality gap*. The duality gap is the difference between the current LP objective function value and the upper bound.

The ability to fix variables depends on the characteristics of the reduced costs and the fact that all the variables must take an integer values of zero or one. Let $\bar{c}_{ij}$ be the reduced cost associated with a variable x_{ij}. Indeed, a unit change in a nonbasic variable x_{ij} of value zero or one, while keeping feasibility, causes at least a change of $\bar{c}_{ij}$ in the objective value.

Figure 4.10 shows the flow diagram of the LP solver routine. The flow written of the variable fixing segment is depicted in Figure 4.11.

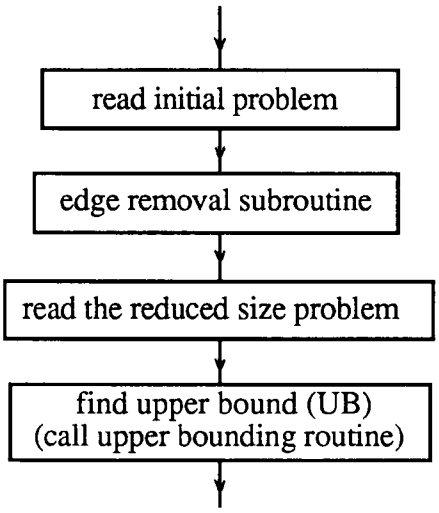


Figure 4.8: Flow diagram of the preprocessing routine.

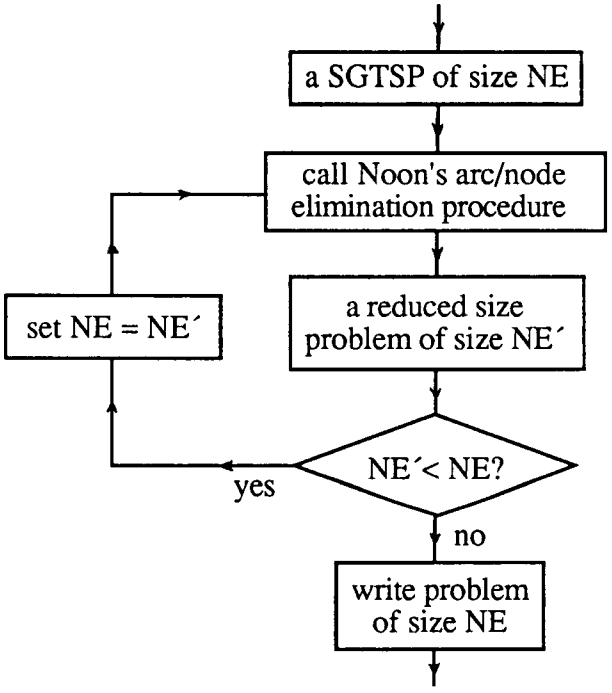


Figure 4.9: Flow diagram of the edge removal subroutine.

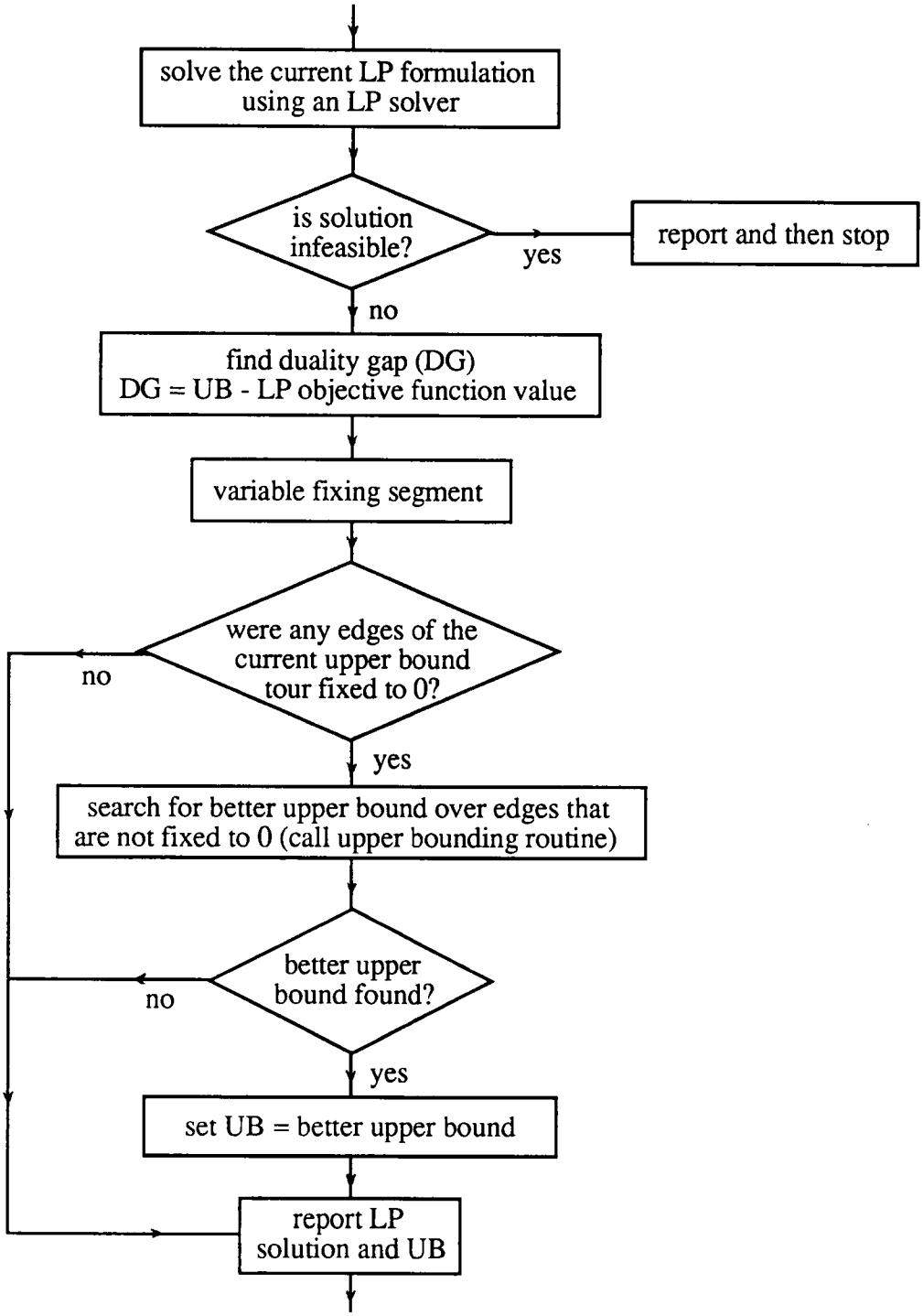


Figure 4.10: Flow diagram of the LP solver routine.

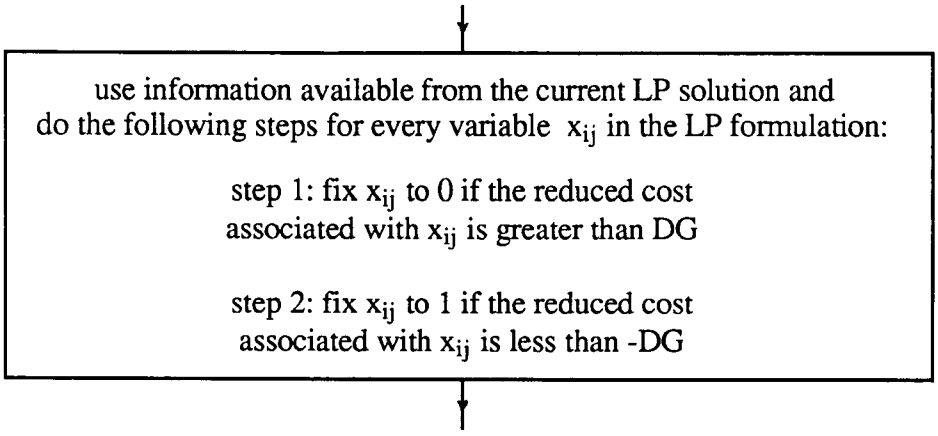


Figure 4.11: Flow written of the variable fixing segment.

4.7 Computer Implementation

This section provides details of the computer implementation of the CPP. We will discuss the CPP code, input file format, test problem configurations and the computational results.

4.7.1 Computer Code

The cutting plane procedure was coded and tested on a DEC VAX 6000-440 at the University of Tennessee Computing Center. All procedures except a timer function and an LP solver were coded in FORTRAN. A large version of the commercial LP solver LINDO (known as LINDO-BIG), was used for solving the linear programs. This version of LINDO can handle linear programming problems with up to 44999 columns and 14999 rows. Two object codes associated with LINDO-BIG, LINDO1L.OBJ and LINDO2L.OBJ (released on 15 January 1990), were linked to the FORTRAN

code. The linkage allowed access to the LINDO subroutines necessarily called within the CPP.

To obtain CPU times for certain portions of the procedure, the function CTIME is called from the IMSL library via linking. The subtour elimination constraint identification routine written by G. Rinaldi and the arc/node elimination routine written by C. E. Noon are the only "borrowed" source codes that are used in the CPP code. All other codes in the CPP were written by the author.

4.7.2 Input File Format

In this section we describe the format of an input file for the CPP. The format was designed so that any SGTSP with or without side constraints can be accommodated. The primary problem size limitations are those imposed by LINDO. The number of nodesets is unrestricted and any nodeset can have any number of nodes from one to m´ (a parameter for the maximum number of nodes) in each nodeset. For any edge (i;j), i, j, and c_{ij} are the required data for the edge. This data is used to construct the objective function of the problem.

Side constraints are introduced in the form of inequality (4.1), where $K \geq 0$ is the number of side constraints.

$$\sum_{(i;\, j)\in E_k} a_{ij}^k x_{ij} \leq b^k \text{ for } k = 1, \ldots, K. \qquad (4.1)$$

The notation E_k represents the set of $|E_k|$ edges that participate in the kth side constraint. If an edge (i; j) has a non-zero left hand side coefficient in some side constraint, it must have been previously defined in the objective

function even if it has an edge cost of zero. It is assumed that the operator of each side constraint is less-than-or-equal, ≤. Thus, any binding constraint of the form ax = b must be introduced in the form of

$$ax \leq b \text{ and } -ax \leq -b.$$

The input file has five main parts that are separated by character strings that serve as the title of each part. The parts are, 1) the number of sets, 2) the vector of the number of nodes in each nodeset, 3) the number of side constraints, 4) the k-vector of right hand sides values of the side constraints, and 5) the coefficients of the rows in a given problem. Specifically, 5) gives the coefficient of the objective function and the nonzero left hand side coefficients of the side constraints, if any. Each of the five parts are separated by a "0" delimiter and the file ends with a "0". The general format for the input file is given in Figure 4.12. It should be noted that, when the number of side constraints is zero, (i.e. K = 0), no lines associated with the values of right hand sides exist, however, the title string of this part, "CSRHS", must still be included.

4.7.3 Test Problems

In this section we describe the test problems that were used for computational study of the CPP. First, we describe an input file generator program. Then, the configurations of the test problems are described.

4.7.3.1 Input File Generator Program

A FORTRAN program was created that allowed a user to generate test problems of various configurations. For the test problems, the input files

```
NSETS          [Stands for number of sets]
      m                                              [format: I7]
SMEMB          [Stands for set members]
      |S1|     |S2|     |S3|     |S4|     |S5|
      |S6|     . . .    |S7|                         [format: 5I7]
NCNST          [Stands for number of constraints]
      K                                              [format: I7]
CSRHS          [Stand for constraints' right hand sides]
      b1       b2       b3       b4       b5
      b6       . . .    bK                          [format: 5I7]
COEFF,CONST    [Stands for coefficients of objf.1 and constraints]
      .        .        .
      i        j        cij      } [objective function cx]
      .        .        .                           [format: 3I7]
      0                                              [format: I7]
      .        .        .
      i        j        a1ij     } [1st side constraint a1.x]
      .        .        .
      0
      .
      .
      .
      0
      .        .        .
      i        j        aKij     } [Kth side constraint aK.x]
      .        .        .
      0        [end of file]
1: objective function.
```

Figure 4.12: Format of the input file.

were written in the previously defined format with a zero number of side constraints. The input generator program allows the use of a predefined configuration. This configuration is described by the total number of nodes and the number of nodes within each nodeset. The program, also provides another configuration that can be generated by the program. The latter configuration is based on the configuration described in Laporte and Nobert [1983].

The Laporte and Nobert configuration method is as follows. For a problem prespecified with n nodes and m nodesets, we assign $|S_1| = 1$ and

$$|S_k| = \begin{cases} \left\lfloor \frac{n-1}{m-1} \right\rfloor + 1 & \text{for } k = 2, \ldots, (n-1) \bmod (m-1) + 1, \\ \left\lfloor \frac{n-1}{m-1} \right\rfloor & \text{for } k = (n-1) \bmod (m-1) + 2, \ldots, m. \end{cases}$$

Note that if $a \bmod b = c$, then $a = ib + c$ where i is a nonnegative integer value.

The input generator permits a user to generate up to ten different problems for any selected configuration. In addition to the edge costs, a problem can include node costs assigned to each node. A node cost is denoted by F_i for every node i in a problem. So as to not introduce a node variable for every node to accommodate the node costs, we distribute the node costs by adding to each edge one half the sum of the node costs corresponding to the edge's end nodes. Node costs are drawn from a uniform [0, F] distribution. The user must specify the F value as either F = 0, 50, 100, 150, or 200.

The program also provides the option of generating Euclidean or non-Euclidean problems. For the Euclidean problems, the x and y coordinates of a node are each drawn from a uniform [0, 100] distribution. An edge cost c_{ij} associated with the edge (i; j) is calculated by the expression (4.3).

$$c_{ij} = INT\{ [(x_i - x_j)^2 + (y_i - y_j)^2]^{0.5} + 0.5(F_i + F_j) + 0.5\}, \qquad (4.3)$$

where x_i and y_i are the x and y coordinates of node i respectively. The function INT is a function that truncates the decimal part of a real number. This function together with adding the fixed value 0.5 to the argument value returns the nearest integer of the argument value.

The edge cost associated with an edge (i; j) in a non-Euclidean problem, denoted by $\bar{c}_{ij}$, is drawn from a uniform [0, 100] distribution. The final cost corresponding to an edge (i; j) is determined by the expression (4.4).

$$c_{ij} = INT[\bar{c}_{ij} + 0.5(F_i + F_j) + 0.5]. \qquad (4.4)$$

Figure 4.13 shows a schematic summary of the input file generator program.

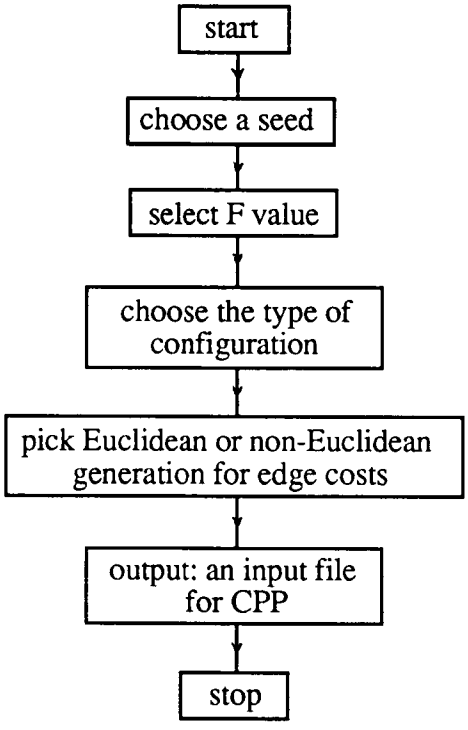


Figure 4.13: A flow diagram of the input generator program.

4.7.3.2 Test Problems Configurations

Randomly generated problems are rarely representative of real world problems; however, algorithms that are successful on random problems often perform well on real world problems. The CPP was tested over problems that are comparable to or are more difficult than previous problems reported in the literature. When F is set to zero, Euclidean test problems appear to be the most difficult. We shall refer to problems with F = 0 as *hard problems*. Problems with F = 100 are considerably less difficult and are referred to as *easy problems* (see Laporte and Nobert [1983], and Noon [1988]). For both types of problems, hard and easy, configurations equivalent to those considered in Laporte and Nobert [1983] are investigated. It should be noted that our preliminary results reemphasized the difficulty of Euclidean problems with F equal to zero as compared to the problems with not equal to zero.

4.7.4 Computational Results

The objective of this section is, by using the CPP, to investigate final bound quality on the test problems. To pursue this objective, 120 different problems were generated and attempted. Each of the 120 problems was defined on a complete GTSP graph which implies that a feasible solution existed for each. The problems consisted of 60 hard problems and 60 easy problems. Within each group, 12 distinct configurations were tested and for each configuration 5 problems were generated using different random seeds. In each group, problems were partitioned into three subgroups, where the number of nodes of the problems in each subgroup is the same.

Tables 4.1, 4.2, 4.3, and 4.4 contain collected data from the solutions of the test problems. The top and bottom portions of each table contain the results for the easy and hard problems, respectively. In addition, the easy problems are numbered in parentheses, and the hard problems are numbered in square brackets. Unless otherwise specified, values given in each column are averaged over five problems of the same configuration.

Table 4.1 provides information about the number of edges of the problems. In this table, the performance of the preprocessing edge removal routine is of interest. This is measured by the ratio of the number of edges left in a problem after applying edge removal routine over the initial number of edges in the complete GTSP graph of the problem. These ratios are given in the last column of Table 4.1. For problems with the same number of nodes but different numbers of nodesets, the effect of the edge removal routine weakens as the number of nodesets increases. Note that when the number of nodes is fixed, an increase in the number of nodesets decreases the number of nodes in each nodeset. The results indicates that, for a problem with few nodes in each nodeset, few edges can be removed. When the nodesets contain a relatively large number of nodes, the case is reversed and many edges are candidates for removal.

Comparing the ratios of both types of problems shows that the edge removal routine for hard problems is less effective. In fact, since no node costs are included in the edge costs of the hard problems, their edge costs are relatively close to one another.

The variable fixing segment inside the LP solver routine has affected the solutions of the problems. In fact, inclusion of the segment in the code allowed the problems to be solved in less time than when no variable fixing

Table 4.1: Data pertaining to edge removal and variable fixing.

Problem	SIZE (n, m)	Initial number of edges †	Number of edges after calling the edge removal routine	Number of edges whose values are set to 0	Number of edges whose values are set to 1	Number of edges left at the end of CPP	Number of edges left/initial number of edges
(1)	40-9	704.00	218.40	140.40	0.20	77.80	0.31
(2)	40-11	723.00	302.60	145.60	0.20	156.80	0.42
(3)	40-16	747.00	476.40	405.20	0.20	71.00	0.64
(4) ††	40-21	761.00	761.00	626.60	1.00	133.40	1.00
(5)	50-9	1099.00	325.00	197.00	0.20	127.80	0.30
(6)	50-11	1129.00	433.80	257.40	0.40	176.00	0.38
(7)	50-16	1168.00	724.60	495.00	0.20	229.40	0.62
(8)	50-21	1187.00	576.40	424.00	1.20	151.20	0.49
(9)	60-11	1625.00	592.80	400.40	0.00	192.40	0.36
(10)	60-13	1654.00	483.60	375.80	0.00	107.80	0.29
(11)	60-16	1683.00	840.20	615.40	0.00	224.80	0.50
(12)	60-21	1712.00	897.40	529.40	0.40	367.60	0.52
[1]	30-7	379.00	113.80	22.00	0.20	91.60	0.30
[2]	30-9	396.00	228.40	117.40	0.20	110.80	0.58
[3]	30-11	407.00	290.00	96.40	0.00	193.60	0.71
[4]	30-16	421.00	410.00	241.00	0.00	169.00	0.97
[5]	40-9	704.00	313.40	108.20	0.00	205.20	0.45
[6]	40-11	723.00	411.80	153.80	0.20	257.80	0.57
[7]	40-16	747.00	607.00	322.40	0.00	284.60	0.81
[8]	40-21	761.00	736.60	395.20	0.00	341.40	0.97
[9]	50-9	1099.00	306.40	63.20	0.00	243.20	0.28
[10]	50 -11	1129.00	461.40	163.00	0.00	298.40	0.41
[11]	50-16	1168.00	882.80	234.40	0.40	648.00	0.76
[12]	50-21	1187.00	1134.20	454.40	0.00	679.80	0.96

† The initial number of edges are the same for all five problems of the same size.
†† The edge removal routine was not called for solving these problems.

Table 4.2: Number of cuts identified in the problems.

Problem	No of DP cuts (NDP)	No. of SE cuts (NSE)	No. of SK cuts (NSK)	No. of GK cuts (NGK)	Total number of cuts (T)	NDP/T	NSE/T	NSK/T	NGK/T
(1)	33.20	3.80	22.00	59.40	118.40	28.04%	3.21%	18.58%	50.17%
(2)	39.60	4.60	31.80	73.00	149.00	26.58%	3.09%	21.34%	48.99%
(3)	41.60	9.40	45.40	89.00	185.40	22.44%	5.07%	24.49%	48.00%
(4)	42.80	11.80	44.40	82.00	181.00	23.65%	6.52%	24.53%	45.30%
(5)	47.00	3.20	20.80	151.40	222.40	21.13%	1.44%	9.35%	68.08%
(6)	56.80	6.20	29.40	148.60	241.00	23.57%	2.57%	12.20%	61.66%
(7)	59.40	8.40	45.00	151.20	264.00	22.50%	3.18%	17.05%	57.27%
(8)	45.00	12.80	42.60	95.80	196.20	22.94%	6.52%	21.71%	48.83%
(9)	78.60	5.40	36.80	292.20	413.00	19.03%	1.31%	8.91%	70.75%
(10)	53.40	6.20	33.20	130.60	223.40	23.90%	2.78%	14.86%	58.46%
(11)	71.80	7.40	38.00	219.80	337.00	21.31%	2.20%	11.28%	65.22%
(12)	68.20	12.40	56.20	200.20	337.00	20.24%	3.68%	16.68%	59.41%
[1]	23.20	1.00	10.80	53.00	88.00	26.36%	1.14%	12.27%	60.23%
[2]	46.60	4.00	21.60	122.40	194.60	23.95%	2.06%	11.10%	62.90%
[3]	45.00	0.80	23.40	126.80	196.00	22.96%	0.41%	11.94%	64.69%
[4]	49.80	7.00	50.60	151.60	259.00	19.23%	2.70%	19.54%	58.53%
[5]	58.40	4.00	25.20	171.40	259.00	22.55%	1.54%	9.73%	66.18%
[6]	57.00	3.20	17.80	155.80	233.80	24.38%	1.37%	7.61%	66.64%
[7]	63.60	5.40	36.80	223.00	328.80	19.34%	1.64%	11.19%	67.82%
[8]	62.00	7.40	54.00	176.60	300.00	20.67%	2.47%	18.00%	58.87%
[9]	57.80	3.60	28.60	172.20	262.20	22.04%	1.37%	10.91%	65.68%
[10]	67.00	2.80	35.00	217.80	322.60	20.77%	0.87%	10.85%	67.51%
[11]	82.20	6.20	36.40	304.60	429.40	19.14%	1.44%	8.48%	70.94%
[12]	88.00	8.20	40.80	306.40	443.40	19.85%	1.85%	9.20%	69.10%

Table 4.3: Ratio of different level of solution over the best known solution.

Problem	Initial LP Solution/BKS	Solution at the Stage 1/BKS	Solution at the Stage 2/BKS	Solution at the Stage 3/BKS	LP Final/BKS
(1)	0.88	0.93	0.94	0.97	1.00
(2)	0.89	0.94	0.96	0.98	1.00
(3)	0.92	0.95	0.97	0.98	1.00
(4)	0.94	0.96	0.97	0.98	1.00
(5)	0.79	0.84	0.89	0.92	0.98
(6)	0.85	0.92	0.93	0.95	1.00
(7)	0.89	0.94	0.96	0.97	1.00
(8)	0.93	0.95	0.96	0.97	0.99
(9)	0.83	0.89	0.91	0.93	0.98
(10)	0.87	0.92	0.94	0.96	1.00
(11)	0.90	0.93	0.96	0.97	1.00
(12)	0.92	0.95	0.96	0.98	1.00
[1]	0.67	0.78	0.80	0.84	0.93
[2]	0.50	0.71	0.73	0.77	0.93
[3]	0.51	0.57	0.70	0.71	0.86
[4]	0.59	0.71	0.75	0.79	0.96
[5]	0.50	0.64	0.68	0.72	0.89
[6]	0.54	0.71	0.72	0.75	0.88
[7]	0.56	0.74	0.76	0.78	0.92
[8]	0.60	0.75	0.77	0.80	0.92
[9]	0.44	0.59	0.62	0.66	0.84
[10]	0.47	0.65	0.67	0.70	0.86
[11]	0.51	0.64	0.67	0.70	0.83
[12]	0.53	0.70	0.71	0.73	0.86

Table 4.4: Summary reports for the CPP.

Problem	Upper bound	Initial LP	Final LP	BKS	Total CPU time "sec." (TT)	CPU time for the LP solver	CPU time for cut identifications (CIT)	CIT/TT	Number of optimal solutions †
(1)	440.40	383.20	433.81	435.60	63.97	58.34	5.63	0.09	4
(2)	575.00	499.80	562.94	563.80	94.11	84.74	9.37	0.10	4
(3)	863.20	790.00	859.85	861.00	106.64	77.26	29.38	0.28	4
(4)	1188.80	1112.70	1182.12	1183.00	153.08	111.07	42.01	0.27	4
(5)	407.60	320.90	398.33	407.00	409.98	391.99	17.99	0.04	1
(6)	494.20	410.20	483.29	483.60	692.86	661.42	31.44	0.05	3
(7)	749.80	658.40	737.08	737.20	887.89	835.12	52.76	0.06	4
(8)	946.60	875.60	938.22	945.40	317.43	264.80	52.62	0.17	3
(9)	484.80	398.20	473.98	481.60	3037.91	2964.41	73.50	0.02	2
(10)	510.40	437.60	502.83	504.60	353.11	325.61	27.50	0.08	3
(11)	707.60	628.00	696.67	697.80	3019.31	2919.47	99.85	0.03	4
(12)	916.40	826.40	896.00	899.40	1379.49	1225.97	153.52	0.11	2
[1]	121.40	75.20	109.54	120.40	34.48	32.16	2.32	0.07	2
[2]	159.20	78.00	147.70	158.80	154.11	143.04	11.07	0.07	2
[3]	188.20	94.60	160.00	186.00	141.22	124.43	16.78	0.12	0
[4]	272.40	156.70	258.76	267.80	305.75	242.42	63.34	0.21	1
[5]	136.00	66.10	118.69	134.20	606.12	586.95	19.17	0.03	0
[6]	166.80	86.50	146.18	166.80	572.16	541.74	30.42	0.05	1
[7]	228.40	126.80	209.37	228.40	690.36	606.21	84.15	0.12	1
[8]	297.40	177.00	273.21	297.40	576.90	422.63	154.27	0.27	0
[9]	123.00	51.20	101.09	122.60	557.77	536.12	21.65	0.04	0
[10]	141.80	65.00	121.10	141.80	1534.30	1482.12	52.18	0.03	0
[11]	203.20	100.00	165.54	203.20	4068.02	3892.86	175.16	0.04	1
[12]	265.60	140.20	227.48	265.60	2492.93	2193.05	299.88	0.12	0

† The values of this column are not averaged over 5 problems.

was used. Problems in which some of the edge variables were fixed to one were often solved very quickly and obtained optimal solutions. When the value of an edge (i;j), $i \in S_I$ and $j \in S_J$, is fixed to be value of one, it means the edge must necessarily be in the optimal tour of the problem. Fixing the value of edge (i;j) to one also means that the values of all edges in $E(S_I \cup S_J) \setminus (i;j)$ can be automatically fixed to zero.

Table 4.2 provides information concerning the various classes of cutting planes identified and added during the CPP. Kite chain constraints (excluding simple kite constraints) constitute approximately 45% of the total number of cuts in the easy problems and approximately 58% of the total in the hard problems. The percentages associated with each of the other cuts descend in the order DP, SK, and SE. The number of SE cuts in the easy problems are relatively higher than the number in the hard problems

In problems with the same number of nodesets and different number of nodes, the total number of cuts increases when the total number of nodes increases. This behavior is observed in both types of problems. In hard problems, when the number of nodes is fixed and the number of nodesets increase from small to large, the total number of cuts increases. However, this is not the case for the hard problems with 40 nodes.

Table 4.3 focuses on the performance of the CPP under various levels of cut identification. The information given in the table not only support the calling order of the cutting plane identification routines, but also gives the effect of the cutting planes towards bounding if used cumulatively.

In Table 4.3, the first column labeled "Initial Solution/BKS" gives the ratio of the initial LP optimum objective value over the best known solution (BKS) of the problem. The initial LP is described only by the objective

function and the degree and variable bound constraints. This would be the lower bound if no violated inequalities could be identified and added. The best known solution for a problem is the objective value of either a confirmed optimal solution or the best heuristically obtained solution available to us.

Three distinct LP objective values for every problem were obtained corresponding to three stages of the cutting plane process. These three stages are defined at the point where a new class of violated valid inequalities were required to continue the CPP. That is, no further improvement of the current solution in a stage is possible unless a new class of cuts are introduced.

The column corresponding to stage 1 reflects the ratio of bounds if only degree constraints and DP cuts were used. The next stage reflects the ratio if we had access only to degree constraints, DP and SE cuts, and the last stage assumes that degree constraints, and DP, SE, and SK cuts were the only accessible constraints.

The best known solutions for the easy problems are, in fact, the confirmed optimal solutions of the problems. The best known solutions of the hard problems, however, are mostly obtained from the best heuristically obtained solutions. These facts must be taken into consideration when analyzing the ratios of the LP objective values over the BKSs.

In the 40, 50, and 60 node easy problems, and the 40 and 50 node hard problems, as the number of nodesets increases, the quality of lower bounds obtained at the initial solution improves. Similar patterns appear for lower bounds obtained at the stage 1 and the stage 2 of problems of the same subgroups.

Figures 4.14 and 4.15 provide a graphical representation of the data given in Table 4.3. Figure 4.14, is associated with the easy problems and Figure 4.15 is associated with the hard problems. In the figures, the points shown by the symbol "I" give the ratio of the initial LP objective values over the BKSs of the problems. The symbol "D" shows points associated with the ratio of objective values at this stage 1 over the BKSs. The symbol "S" and "K" show the points associated with the ratio of objective values obtainable at the second and third stages respectively over the BKSs. The points shown by the symbol "C" are the ratio of final LP objective values, obtained by using degree constraints, and DP, SE, SK, and KC cuts, over the BKSs of the problems.

Table 4.4 provides a summary report for the CPP. The total CPU times for the CPP do not appear to follow a specific pattern, however, the CPU times spent to identify violated inequalities are a function of the size of the problems. In problems with the same number of nodesets but different number of nodes per nodeset, the identification times increase when the number of nodes increases. Within each subgroup, a same pattern appears. This suggests that problems with fewer nodes in each nodeset require relatively more time for cutting plane identification routines. It should be noted that identification time for valid inequalities represents a small portion of the total time required to solve any problem.

The number of problems that were solved optimally are given in the last column of Table 4.4. This information shows that 38 out of the 60 easy problems and 8 out of the 60 hard problems were solved to optimality.

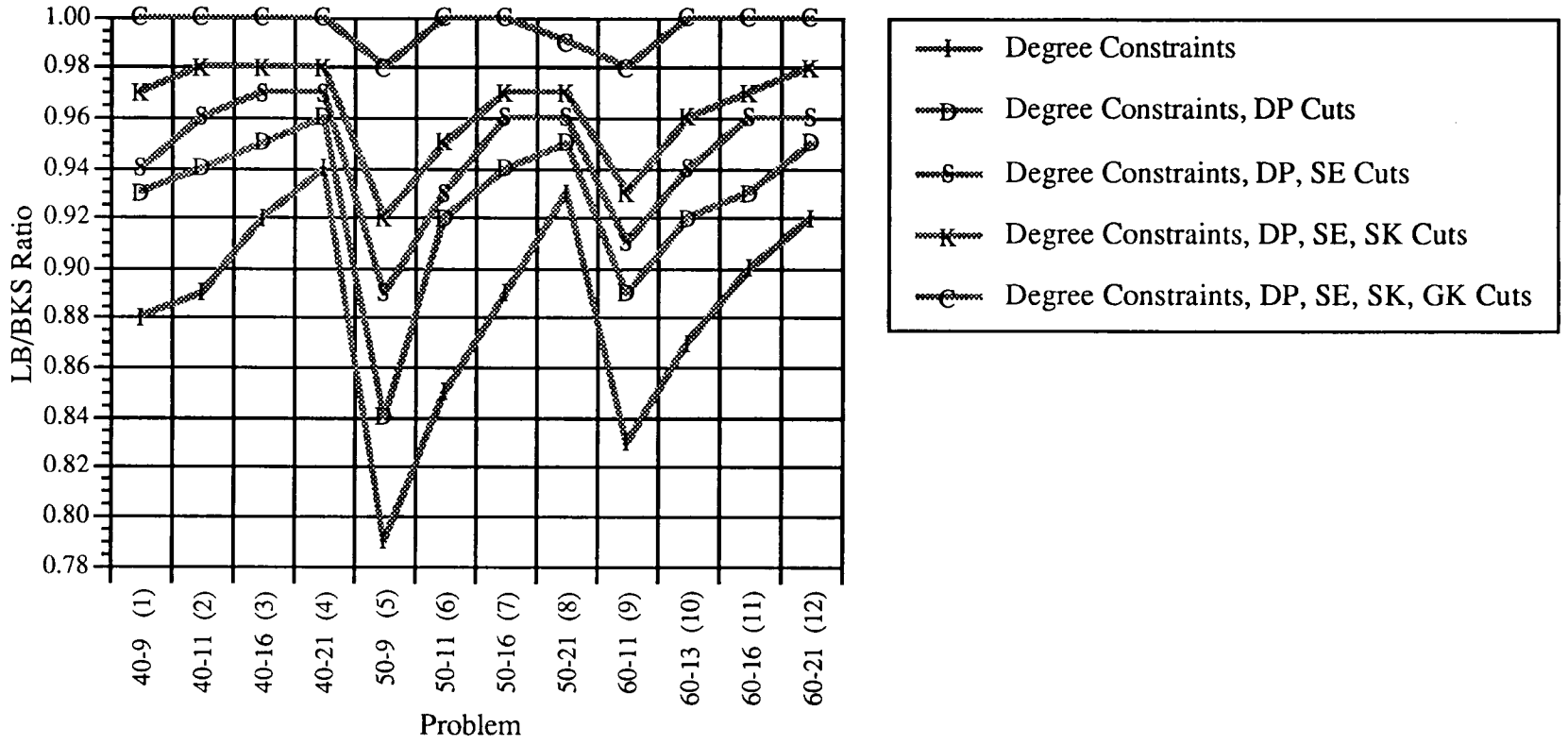


Figure 4.14: The performance of the CPP under various levels of cut identifications (easy problems).

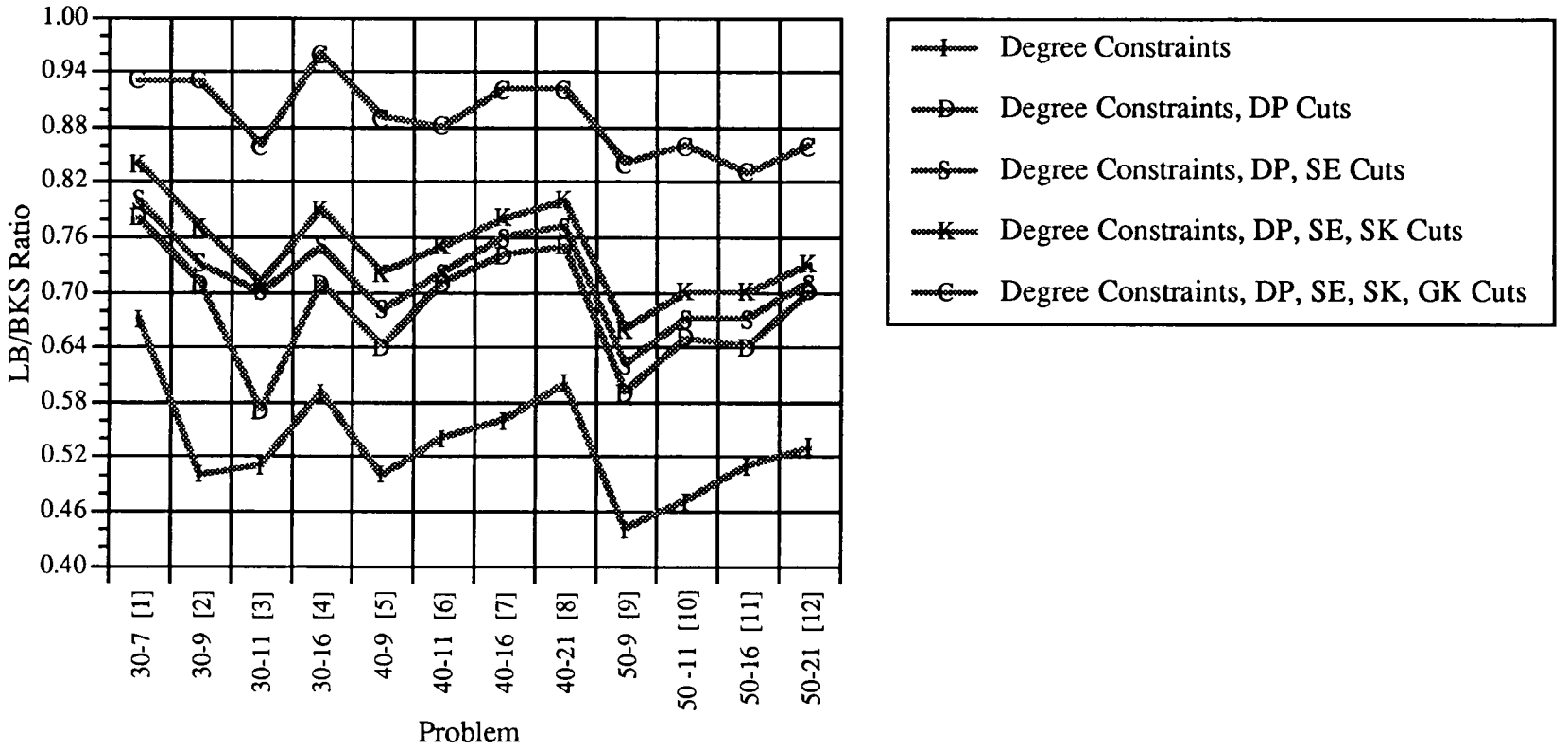


Figure 4.15: The performance of the CPP under various levels of cut identifications (hard problems).

CHAPTER 5

CONCLUSIONS AND FUTURE RESEARCH

5.1 Conclusions

The Symmetric Generalized Traveling Salesman Problem is an important yet difficult problem in combinatorial optimization. The problem plays host to a number of applications that includes operational problems such as distribution and scheduling, and planning problems such as facility location. The modeling flexibility of the GTSP can be attributed to its ability to simultaneously model the decisions of sequence and selection. In recent years, the problem has attracted the attention of more researchers and practitioners.

The literature contains little work on either the structural or computational aspects of the symmetric GTSP. In this study, we have analytically investigated four new classes of valid inequalities for the SGTSP. Although the disjoint prevention constraints and the generalized subtour prevention constraints are rather straightforward, we have shown that they contribute to the lower bounding process noticeably. Perhaps the

most important contribution of our study has been the development of a general definition for SGTSP kite constraints and the proof of their validity.

Another aspect of this research has been the development of a linear programming based cutting plane procedure for the SGTSP. The procedure was developed to study the computational effectiveness of the valid inequalities on a collection of test problems. This work required the development of algorithms for identifying violated constraints and heuristics for constructing and improving SGTSP tours.

5.1 Future Research

Real world problems involving sequence and selection decisions can be readily formulated using the GTSP model. Modeling and solving problems of this type is the most immediate area of future research.

Explicit restrictions in a problem such as visiting a specific node or passing a particular edge decrease the size of the problem noticeably. Such restrictions are more likely in real world applications. Through the addition of side constraints and/or additional variables, most real-world restrictions or complexities can be modeled easily on the GTSP structure. Our cutting plane procedure (CPP) can still be applied to produce lower bounds on these problems. One of the most important real-world problems involving sequence and selection decisions is the vehicle routing problem (VRP). As was shown in chapter 2, the VRP can be modeled as a SGTSP with side constraints. Our plan is to test the effectiveness of the CPP on the VRP. Future research in the area of variable and node elimination for the SGTSP and, in particular, for the SGTSP with side constraints is needed.

The computational results on the hard problems suggest that additional work is needed in the area of identifying valid inequalities for the SGTSP. Work is needed in the areas of kite identification and defining new inequalities. In our cutting plane procedure, we searched only for special cases of violated kite constraints. Better kite identification algorithms would likely find more general kites and result in a tighter lower bound. The classes of valid inequalities we have defined contain a huge number of constraints and have proved to be computationally effective. It is likely there are other yet-to-be-defined classes of inequalities which will lead to computational advances. We plan to continue the search.

As is the case for the symmetric TSP, *branch-and-cut* would likely be the best method for solving SGTSPs to optimality. A branch-and-cut algorithm is different from the standard branch-and-bound methods. Whenever a cutting plane procedure terminates with a nonoptimal solution, the branch-and-cut algorithm uses a tree-search strategy that continues to identify and append valid inequalities (cuts) after branching (see Padberg and Rinaldi [1987] and [1989]). Our cutting plane procedure can be incorporated into a branching scheme for solving SGTSPs under a variety of strategies. One strategy would be to first branch on the nodes of a nodeset (fix a node to be in a tour) and then proceed with the cutting plane procedure. A second strategy would be to use the cutting plane procedure first and then begin branching.

In a preliminary study, we tested the second strategy using the CPP as the cutting routine on two different hard problems which the CPP alone was unable to solve to optimality. The problems were configured as 31 nodes over 16 nodesets and 30 nodes over 7 nodesets. We used Noon's

approach for solving the AGTSP (see Noon and Bean [1989a]) to obtain the optimal solution of the first problem. This approach took approximately 6500 seconds CPU time. Using the described branch-and-cut strategy, we were able to solve the first problem to optimality in 1000 seconds (approximately) with 5 levels of branching and 12 subproblems evaluated. The second problem was solved to optimality in approximately 5 seconds of CPU time with only one level of branching and evaluating only 4 subproblems. The continued study of branch-and-cut schemes for solving the SGTSP will likely lead to the solution of markedly larger problems.

BIBLIOGRAPHY

BIBLIOGRAPHY

Bartholdi, III, J. J. and L. K. Platzman [1986], "Retrieval Strategies for a Carousel Conveyor," **IIE Transactions**, June 1896.

Bodin, L., B. Golden, A. Assad, and M. ball [1983], "Routing and Scheduling of Vehicles and Crews: The State of the Art," **Computers & Operations Research**, Vol. 10.

Bovet, J. [1983], "The Selective Travelling Salesman Problem," paper presented at EURO VI Conference, Vienna.

Carpaneto G. and P. Toth, "Some New Branching and Bounding Criteria for the Asymmetric Traveling Salesman Problem," **Management Science**, 26.

Christofides, N., A. Mingozzi, and P. Toth [1981], "State-Space Relaxation Procedures for the Computation of Bounds to Routing Problems," **Networks**, Vol. 11.

Current, J. and D. Schilling [1989], "The Covering Salesman Problem," **Transportation Science**, Vol. 23, No. 3.

Dantzig, G. B. , D. R. Fulkerson and S. Johnson, "Solution of a Large Scale Traveling Salesman Problem," **Operations Research**, Vol. 2.

Dreyfus, S. E. and A. M. Law [1977], **The Art and Theory of Dynamic Programming**, Academic Press, New York.

Fisher, M. L. and R. Jaikumar [1981], "A Generalized Assignment Heuristic for Vehicle Routing Problem," **Networks**, Vol. 11.

Garfinkel R. S. [1985], "Motivation and Modeling," in **The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization**, E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, eds., John Wiley & Sons, New York.

Goldfarb, D. and M. J. Todd [1989], "Linear Programming," in **Handbooks in Operations Research and Management Science, Volume 1, Optimization**, G. L. Nemhauser, A. H. G. Rinnooy Kan, and M. J. Todd, eds., North-Holland, Amsterdam.

Gomory, R. E. and T. C. Hu [1961], "Multi-Terminal Network Flows," **SIAM Journal of Applied Mathematics**, Vol. 9.

Gonzalez, R. H. [1962], "Solutions of Traveling Salesman Problem by Dynamic Programming on the Hypercube," Interim Technical Report No. 18, Massachusetts Institute of Technology.

Grötschel, M. and M. Padberg [1985], "Polyhedral Computations," in **The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization**, E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, eds., John Wiley & Sons, New York.

Henry-Labordere, A. L. [1969], "The Road Balancing Problem: A Dynamic Programming Solution of a Generalized Travelling Salesman Problem," **RIRO**, Vol. B-2.

Hoffman, A. J. and P. Wolfe [1985], "History," in **The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization**, E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, eds., John Wiley & Sons, New York.

Kershenbaum, A., Hsieh, W. and Golden, B. L. [1976], "Constrained Routing in Large Sparse Networks," Conference Record of the IEEE International Conference on Communications, Philadelphia.

Land, A. and A. Doig [1960], "An Automatic Method for Solving Discrete Programming Problems," **Econometrica**, Vol. 28, No. 3.

Laporte, G. and Y. Nobert [1983], "Generalized Travelling Salesman Problem Through n Sets of Nodes: An Integer Programming Approach," **INFOR**, Vol. 21, No. 1.

Laporte, G., H. Mercure and Y. Nobert [1985], "Finding the Shortest Hamiltonian Circuit Through n Clusters: A Lagrangean Approach," **Congressus Numerantium**, Vol. 48.

Laporte, G., H. Mercure and Y. Nobert [1987], "Generalized Travelling Salesman Problem Through n Sets of Nodes: The Asymmetrical Case," **Discrete Applied Mathematics**, Vol. 18.

Lin, S. and B. W. Kernighan [1973], "An Effective Heuristic Algorithm for the Traveling Salesman Problem," **Operation Research**, Vol. 21.

Magee, J. and W. Copacino and D. Rosenfield [1985], **Modern Logistics Management**, John Wiley & Sons, New York.

Nemhauser, G. L. and L. A. Wolsey [1988], **Integer and Combinatorial Optimization**, John Wiley & Sons, New York.

Noon, C. E. [1988], "The Generalized Traveling Salesman Problem," unpublished dissertation, Department of Industrial and Operations Engineering, The University of Michigan, Ann Arbor.

Noon, C. E. and J. C. Bean [1989a], "A Lagrangian Based Approach to the Asymmetric Generalized Traveling Salesman Problem," working paper, Department of Management, The University of Tennessee, Knoxville.

Noon, C. E. and J. C. Bean [1989b], "An Efficient Transformation of the Generalized Traveling Salesman Problem," working paper, Department of Management, The University of Tennessee, Knoxville.

Padberg, M. and G. Rinaldi [1987], "Optimization of a 532-City Symmetric Traveling Salesman Program by Branch-and-Cut," **Operations Research Letters** 6.

Padberg, M. and G. Rinaldi [1988], "An Efficient Algorithm for the Minimum Capacity Cut Problem," Consiglio Nazionale Delle Ricerche and Istituto di Analisi dei Sistemi ed Informatica, R. 222, Luglio.

Padberg, M. and G. Rinaldi [1989], "An Branch-and-Cut Algorithm for the Resolution of Large-Scale Symmetric Traveling Salesman Problem," Report 89-76, New York University, Leonard N. Stern School of Business.

Rousseau, J. [1988], "Customization Versus a General Purpose Code for Routing and Scheduling Problems: A Point of View," in **Vehicle Routing: Methods and Studies**, B. Golden and A. Assad, eds., Elsevier Science Pub., Amsterdam.

Saksena, J. P. [1970], "Mathematical Model of Scheduling Clients Through Welfare Agencies," **CORS Journal**, Vol. 8.

Srivastava, S. S., S. Kumar, R. C. Garg and P. Sen [1969], "Generalized Travelling Salesman Problem Through n Sets of Nodes," **CORS Journal**.

APPENDICES

APPENDIX A

Glossary of Mathematical Symbols

Symbol	Meaning
A, B, ..., Z	sets/ matrices.
a, b, ..., z	elements of sets/ vectors.
$\cup$	union of two sets.
$\cap$	intersection of two sets.
$\subseteq$	subset.
$\subset$	proper subset.
$\emptyset$	null set or empty set.
$\forall$	for all, for each.
$\exists$	there exist(s), there is (are).
$\mid$	such that.
$\in$	belongs to.
"$\mid$" or "/" through a symbol	opposite or negative meaning of the symbol.
Σ	summation.
$\equiv$	is equivalent to, is identical to.
A\B	A-B or A- (A$\cap$B).

A'	the complement of set A.
■	Q.E.D. : quod erat demonestrandum (which was to be demonstrated).
$\lvert A \rvert$	cardinality of set A, i.e. number of elements in the set A.
(i,j)	an arc from i to j/ ordered pair.
(i;j)	an edge between i and j/ unordered pair.
$\Re^n$	the set of n-dimensional real vectors
Z_+^n	the set of n-dimentional nonnegative integer vector.

APPENDIX B

Example Using the Iterative Process of Kite Chain Growth

Consider a SGTSP with 20 nodes and 8 nodesets, where $S_1 = \{1\}$, $S_2 = \{2, 3, 4\}$, $S_3 = \{5, 6, 7\}$, $S_4 = \{8, 9, 10\}$, $S_5 = \{11, 12, 13\}$, $S_6 = \{14, 15, 16\}$, $S_7 = \{17, 18\}$, and $S_8 = \{19, 20\}$. A P_1 solution of the problem is given in Figure B.1.

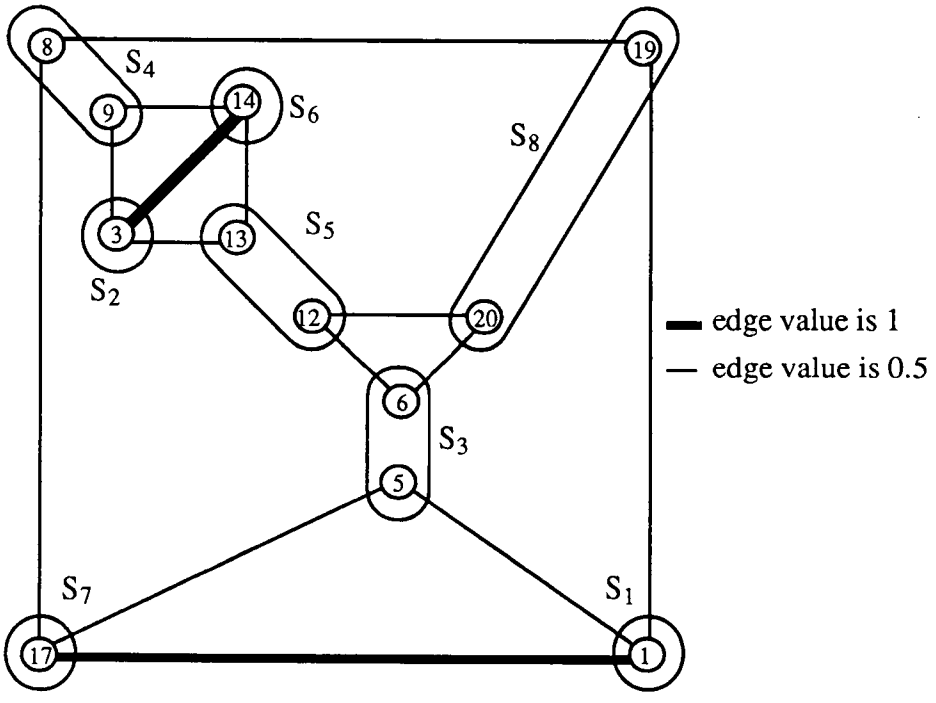


Figure B.1: A P_1 solution of a problem.

Consider a kite chain with 3 kite frames, {3, 9, 13, 14}, {6, 12, 20}, and {1, 5, 17}. Let K be the set of all nodes in the kite frames, K = {1, 3, 5, 6, 9, 12, 13, 14, 17, 20}.

Step 1. (**Initialization**). Set i = 1, K = {1, 3, 5, 6, 9, 12, 13, 14, 17, 20}, k = 3, K_1 = {3, 9, 13, 14},
$\overline{K}^1 = \{K_1\}$, $S^1 = \emptyset$, $\overline{S}^1 = \emptyset$, $L^1 = \emptyset$, and $\overline{L}^1 = \emptyset$.

Step 2. (**Linkable kite frame selection**). Select a kite frame from $K \backslash \overline{K}^1$ = {1, 5, 6, 12, 17, 20} such that it is linkable to the kite frame in $\overline{K}^1$. Let {6, 12, 20} be such a kite frame and name it K_2, that is K_2 = {6, 12, 20}.

Step 3. (**Connective nodeset scanning**). Set $S^2 = \{S_5\}$, since S_5 chains K_2 to $\overline{K}^1$. There is $\overline{n} = 1$ newly scanned connective nodeset. Index $\overline{n}$ connective nodesets from $|\overline{S}^1|+1$ to $|\overline{S}^1|+\overline{n}$. Since $|\overline{S}^1| = 0$, we index the connective nodeset 1. So $S_{C_1} \equiv S_5$, and $S_2 = \{S_{C_1}\}$.
$\overline{S}^2 = S^2 \cup \overline{S}^1$, $\Rightarrow \overline{S}^2 = \{S_{C_1}\}$. $L^2 = \{1\}$, $\overline{L}^2 = \{1\}$.

Set $RS^2 = \overline{L}^1 \cap L^2 = \emptyset$, $NS^2 = L^2 \backslash (\overline{L}^1 \cap L^2) = \{1\}$, and $PS^2 = \overline{L}^2 \backslash L^2 = \emptyset$.

Step 4. (**Covered kite frames calculation**).
$|C_1^2| = 2$, since $1 \in NS^2$.

Step 5. (**Inequalities development**). The value of i is 2. The following inequality is valid.

$$\sum_{j=1}^{2} x(E(K_j)) \le \sum_{j=1}^{2} (|S(K_j|-1) - \sum_{l \in \overline{L}^2} (|C_l^2|-1).$$

Set i = 3. (go to step 2).

Step 2. (**Linkable kite frame selection**). Select a kite frame from $K \backslash \overline{K}^2$ = {1, 5, 17} that is linkable to the kite frames in $\overline{K}^2$. Let {1, 5, 17} be such a kite frame and name it K_3, that is K_3 = {1, 5, 17}.

Step 3. (**Connective nodeset scanning**). Set $S^3 = \{S_3\}$, since S_3 chains K_3 to $\overline{K}^2$. There is $\overline{n} = 1$ newly scanned connective nodeset. Since $|\overline{S}^2| = 1$, we index S_3 to $|\overline{S}^2|+1 = 2$. So $S_{C_2} \equiv S_3$, and $S_3 = \{S_{C_2}\}$. $\overline{S}^3 = S^3 \cup \overline{S}^2$, $\Rightarrow \overline{S}^3 = \{S_{C_1}, S_{C_2}\}$. $L^3 = \{2\}$, $\overline{L}^3 = \{1, 2\}$.

Set $RS^3 = \overline{L}^2 \cap L^3 = \emptyset$, $NS^3 = L^3 \setminus (\overline{L}^2 \cap L^3) = \{2\}$, and $PS^3 = \overline{L}^3 \setminus L^3 = \{1\}$.

Step 4. (**Covered kite frames calculation**).

$|C_2^3| = 2$, since $2 \in NS^3$.

$|C_1^3| = |C_1^2| = 2$, since $1 \in PS^3$.

Step 5. (**Inequalities development**). The value of i is 3. The following inequality is valid.

$$\sum_{j=1}^{3} x(E(K_j)) \leq \sum_{j=1}^{3} (|S(K_j|-1) - \sum_{l \in \overline{L}^3} (|C_l^3|-1).$$

Since i = 3 = k, go to step 6.

Step 6. (**Growth termination**). Since i = k, $|C_l^3| \equiv |C_1|$. In addition $\overline{L}^3 = \{1, 2\}$. Therefore the above inequality can be written as

$$\sum_{j=1}^{3} x(E(K_j)) \leq \sum_{j=1}^{3} (|S(K_j|-1) - \sum_{l=1}^{2} (|C_l|-1).$$

APPENDIX C

Example Using the Inverse Transformation of ATSP Constraints

This example is based on a P_1 solution taken from a SGTSP with 30 nodes and 8 nodesets. In this problem, $S_1 = \{1\}$, $S_2 = \{2, 3, 4, 5, 6\}$, $S_3 = \{7, 8, 9, 10\}$, $S_4 = \{11, 12, 13, 14\}$, $S_5 = \{15, 16, 17, 18\}$, $S_6 = \{19, 20, 21, 22\}$, $S_7 = \{23, 24, 25, 26\}$, and $S_8 = \{27, 28, 29, 30\}$. Figure C.1 show the solution denoted by $\bar{x}_f$. The GTSP graph associated with the problem is denoted by $\mathcal{G}$.

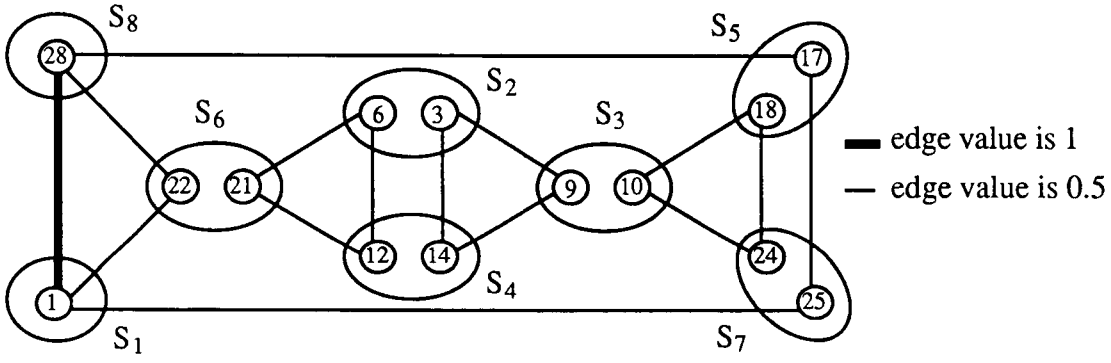


Figure C.1: $\bar{x}_f$, a feasible solution for (P^1).

From SGTSP to AGTSP

Given $\bar{x}_f$ on $\mathcal{G}$, we obtained $\bar{y}_f$, a feasible solution for $(\vec{P}1)$ on a digraph $\mathcal{D}$ equivalent to $\mathcal{G}$.

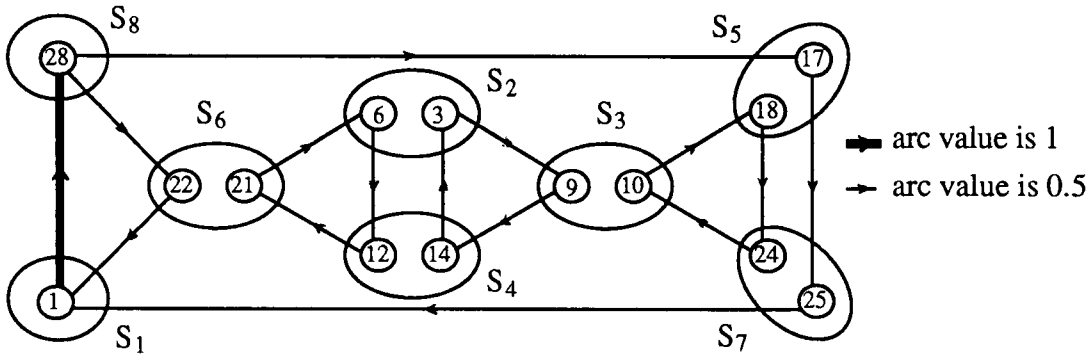


Figure C.2: $\bar{y}_f$, a feasible solution for $(\vec{P}1)$ given $\bar{x}_f$.

From AGTSP to ATSP

The solution $\bar{y}_f^{TSP}$ is a feasible solution to $(\vec{P}2)$ on digraph $\mathcal{D}^{TSP}$, where $\mathcal{D}^{TSP}$ is a digraph based on the transformation of the GTSP digraph $\mathcal{D}$ with respect to the solution $\bar{y}_f$.

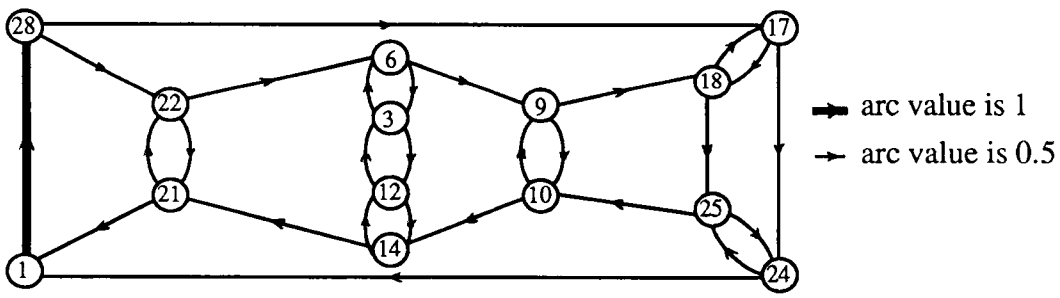


Figure C.3: $\bar{y}_f^{TSP}$, a feasible solution to $(\vec{P}2)$ on $\mathcal{D}^{TSP}$.

From ATSP inequalities to SGTSP inequalities

Identify ATSP inequalities:

Given $\bar{y}_f^{TSP}$,we have identified a comb inequality with the corresponding digraph shown in Figure C.4. The inequality (C.1) gives the expression of the identified comb inequality.

$$y(A(H)) + \sum_{i=1}^{3} y(A(T_i)) \leq |H| + \sum_{i=1}^{3} (|T_i| - 1) - \frac{3+1}{2} \equiv \beta y \leq \beta_\circ. \qquad (C.1)$$

The inequality (C.1) is violated since the value associated with its left hand side is 9.5 and the value associated with its right hand side is 9.

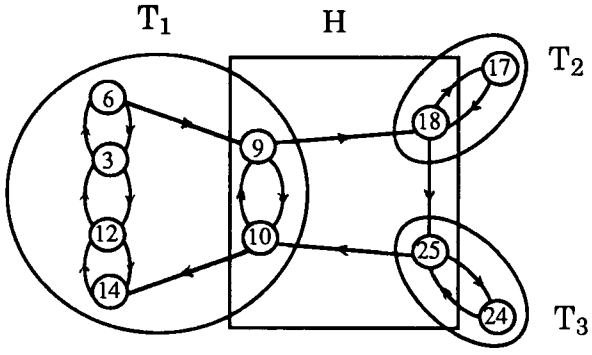


Figure C.4: A comb associated with $\mathcal{D}^{TSP}$.

Figure C.5 shows the remaining arcs after intraset arcs are dropped from the comb. These intraset arcs are, (12,14), (14,12), (3,6), (6,3), (10,9), (9,10), (24,25), (25,24), (17,18), (18,17). The value 6 is the value associated with these intraset arcs. Subtracting this value from the right hand side of (C.1) yields the value 3 which is equivalent to $\beta'_\circ$. Note that in the inequality

(C.1), the coefficients associated with the variable of arcs (9,10) and (10,9) are 2, since these two arcs are in $A(H \cap T_1)$.

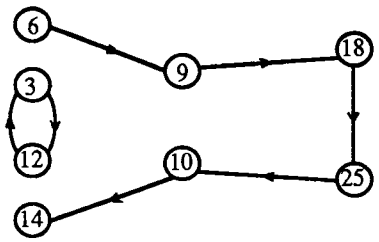


Figure C.5: The comb after the intraset arcs are dropped.

The valid inequality corresponding to interset arcs in $\mathcal{D}^{TSP}$ is as follows.

$$y_{3,12} + y_{12,3} + y_{69} + y_{9,18} + y_{18,25} + y_{25,10} + y_{10,14} \leq 3 \equiv \beta' y \leq \beta'_0. \qquad (C.2)$$

Back transforming the inequality (C.2) yields,

$$x_{6,12} + x_{39} + x_{3,14} + x_{9,14} + x_{10,18} + x_{10,24} + x_{17,23} \leq 3 \equiv \beta'' x \leq \beta''_0. \qquad (C.3)$$

The inequality (C.3) is a valid inequality for our SGTSP described at the beginning. Figure C.6 show the edges that are the result of back transformation of arcs in Figure C.5.

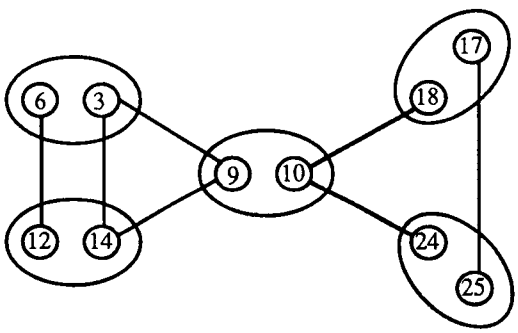


Figure C.6: Edges that are resulted form back transformation of arcs in Figure C.5.

Note that we can identify a kite chain inequality associated with the collection of nodes in the inequality (C.3). Let K_1 = {3, 6, 9, 12, 14} and K_2 = {10, 17, 18, 24, 25}. The kite frames associated with the kite chain are shown in Figure C.7. The thick edges are those edges that their corresponding variables appear in the left hand side of the kite chain inequality but do not appear in (C.3). Let **E** be the set of edges shown with thick lines in Figure C.7, then the kite chain inequality is as follows.

$$x_{6,12} + x_{39} + x_{3,14} + x_{9,14} + x_{10,18} + x_{10,24} + x_{17,23} + \sum_{(i,j)\in \mathbf{E}} x_{ij} \leq 3. \qquad (C.4)$$

The inequality (C.4) is stronger than the inequality (C.3).

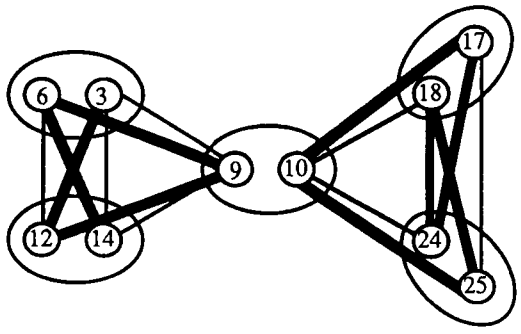


Figure C.7: A kite chain derived from the solution given in Figure C.1.

APPENDIX D

Results of Two Studies

In a study we compared the effects of two alternative concerning DP constraints, adding DP constraints as cutting planes, and using full set of DP constraints included in the initial LP description. A group of 15 hard problems of different sizes were tested. The problems have the same configuration described in Laporte and Nobert [1983]. The data given in table D.1 indicate the advantage of appending DP constraints as cutting planes over the option of having all possible DP constraints as part of the initial formulation of a problem. That is, the alternative of appending DP constraints only as needed is computationally superior to the alternative of including the entire set of DP constraints in the initial LP description. Note that, based on the applied configuration, the first nodeset of every problem contains exactly one node.

In another study, the performance of the four combination heuristics were tested. These heuristics are nearest neighbor/Imp-Opt (NNIO), nearest neighbor/Opt-Imp (NNOI), nearest insertion/Imp-Opt (NIIO), and nearest insertion/Opt-Imp (NIOI). The results showed that, on average the NNIO heuristic performs better than the others, however, in the cutting plane procedure we selected the least cost tour from the four available. The

Table D.1: DP constraints as cutting planes vs full set of DP constraints included in initial formulation.

Problem	n: No. of nodes	m: No. of nodesets	Problem configuration	No. of edges	Initial LP solution †	Final LP solution ††	Maximum no. of DP constraints	CPU time ††† (all DP const. in formulation)	No. of DP constraints appended	CPU time ††† (for DP const. as cutting plane)
1	21	11	1 and 10 X 2	200	151.00	182.67	210	1.64	15	0.62
2	31	11	1 and 10 X 3	435	97.00	120.00	310	25.72	31	1.58
3	41	21	1 and 20 X 2	800	156.00	212.00	820	340.94	39	3.08
4	61	21	1 and 20 X 3	1770	115.00	151.00	1220	2360.50	59	9.14
5	61	31	1 and 30 X 2	1800	184.00	230.00	1830	2103.90	58	9.64
6	91	31	1 and 30 X 3	4005	120.50	161.50	2730	∞	96	49.27
7	81	41	1 and 40 X 2	3200	190.00	238.00	3240	∞	83	28.94
8	121	41	1 and 40 X 3	7140	137.00	174.00	4840	∞	104	81.50
9	101	51	1 and 50 X 2	5000	197.00	238.75	5050	∞	83	34.09
10	151	51	1 and 50 X 3	11175	148.50	212.75	7550	∞	140	222.15
11	121	21	1 and 20 X 6	6960	51.00	79.96	2420	∞	91	45.04
12	121	25	1 and 24 X 5	7020	62.00	94.36	2904	∞	102	62.84
13	121	31	1 and 30 X 4	7080	89.00	121.91	3630	∞	116	98.20
14	121	41	1 and 40 X 3	7140	137.00	174.00	4840	∞	104	82.40
15	121	61	1 and 60 X 2	7200	248.50	281.00	7260	∞	104	69.06

∞: couldn't solve (out of space)

† for formulations with no DP constraints.
†† when DP constraints are added as part of formulation or as cutting planes.
††† in seconds on a DEC VAX 8800

Table D.2: The performance of upper bounding procedures.

Problem †	Nearest neighbor (NN)	NN/OPT-IMP (NNOI)	NN/IMP_OPT (NNIO)	Nearest insertion (NI)	NI/OPT-IMP (NIOI)	NI/IMP-OPT (NIIO)	Best of four
1	120.80%	120.80%	120.80%	156.00%	121.60%	121.60%	120.80%
2	106.43%	100.00%	100.00%	113.57%	100.00%	100.00%	100.00%
3	109.77%	100.00%	100.00%	156.39%	114.29%	115.04%	100.00%
4	112.00%	100.00%	100.00%	110.00%	100.00%	100.00%	100.00%
5	102.41%	102.41%	102.41%	125.30%	113.86%	113.86%	102.41%
6	104.84%	100.00%	104.84%	105.65%	100.81%	102.42%	100.00%
7	109.03%	109.03%	103.23%	129.03%	107.10%	106.45%	103.23%
8	108.39%	108.39%	104.90%	106.29%	104.20%	106.29%	104.20%
9	109.85%	109.85%	100.00%	129.55%	124.24%	110.61%	100.00%
10	103.70%	103.70%	100.93%	122.22%	102.78%	100.00%	100.00%
Average	108.72%	105.42%	103.71%	125.40%	108.89%	107.63%	103.06%

† the configuration of all the problems is 30 nodes - 8 nodesets.

least cost tour is named the best-of-four. Table D.2 gives the performance of the four combination heuristics on ten problems of size 30 node-8 nodeset with the same configuration as was used for the above study.

VITA

A Crooked Fucked Up Man

Mohammad Mehdi Sepehri was born in Hamadan, Iran on February 16, 1955. He attended Elmi Elementary School and Dr. Shariátee High School for his elementary and high school education. He obtained a Bachelor of Arts degree in Business from Tehran Business College (currently, Allame Tabatabaei University) in February 1977. After graduation, he worked as instructor teaching the senior level courses "Quantitative Analysis in Business" and "Cost Accounting" at vocational high schools in Hamadan.

In January 1986, he entered The University of Tennessee, Knoxville, to pursue a Master of Science degree in Management Science. This degree was awarded in August 1987. He subsequently accepted a graduate assistantship at The University of Tennessee, Knoxville and began work towards a Doctor of Philosophy degree in Management Science. This degree will be awarded in August 1991.

The author is a member of the Operations Research Society of America (ORSA), The Institute of Management Sciences (TIMS), and the Decision Sciences Institute (DSI). He is married and has three children.