On Applying Methods for Graph-TSP to Metric TSP

Nicholas Desjardins

Thesis submitted to the Faculty of Graduate and Postdoctoral Studies

in partial fulfillment of the requirements for the degree of

Master of Computer Science

Ottawa-Carleton Institute for Computer Science

University of Ottawa

Abstract

The Metric Travelling Salesman Problem, henceforth metric TSP, is a fundamental problem in combinatorial optimization which consists of finding a minimum cost Hamiltonian cycle (also called a TSP tour) in a weighted complete graph in which the costs are metric. Metric TSP is known to belong to a class of problems called NP-hard even in the special case of graph-TSP, where the metric costs are based on a given graph. Thus, it is highly unlikely that efficient methods exist for solving large instances of these problems exactly.

In this thesis, we develop a new heuristic for metric TSP based on extending ideas successfully used by Mömke and Svensson for the special case of graph-TSP to the more general case of metric TSP. We demonstrate the efficiency and usefulness of our heuristic through empirical testing.

Additionally, we turn our attention to graph-TSP. For this special case of metric TSP, there has been much recent progress with regards to improvements on the cost of the solutions. We find the exact value of the ratio between the cost of the optimal TSP tour and the cost of the optimal subtour linear programming relaxation for small instances of graph-TSP, which was previously unknown. We also provide a simplified algorithm for special graph-TSP instances based on the subtour linear programming relaxation.

Acknowledgements

First and foremost, I would like to thank my supervisor Dr. Sylvia Boyd for her continuous support and guidance throughout this thesis. Her advice, feedback and words of encouragement have been invaluable. Her confidence in me has been instrumental in my success, and has been a source of motivation in achieving my goals. I have been very fortunate to have had her as a mentor.

I would also like to thank the University of Ottawa for its financial support, whether it be in the form of teaching assistantships or scholarships.

Finally, I would like to thank my friends and family, whom have been incredibly supportive and a great source of encouragement throughout this endeavour.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

A salesperson is tasked with visiting a collection of cities, each interconnected to one another, such that each city is visited exactly once and such that the salesperson ends at the starting point. Moreover, the salesperson wishes to travel the least distance possible. More concisely, given a complete graph G on n nodes (or cities) with non-negative edge costs, we call a cycle that visits every node exactly once a TSP tour. Then, the *Travelling Salesman Problem*, henceforth TSP, is to find the cheapest TSP tour in G.

TSP is a fundamental and difficult problem in combinatorial optimization [10, 26] with applications in many areas of practical interest, including vehicle routing, scheduling and genetics [10, 31]. Unfortunately, there have yet to be any practical or efficient methods to solve large instances of TSP exactly. In fact, it is believed that no such methods exist since TSP is an NP-hard [1] problem [26]. This is not without consequence, as real-world instances are very large, and often we wish to solve such instances in a reasonable amount of time. However, if the solution need

[1] We refer the readers to Garey and Johnson [18] for details concerning complexity classes.

not be optimal, there exists methods known as heuristics which are often very fast and make it a much more manageable task to solve very large and complex instances. At times, heuristics have a performance guarantee, that is, the solutions produced have cost within a constant factor $\alpha \geq 1$ of the cost of an optimal solution. Such a heuristic is known as an α-approximation algorithm. Unfortunately, it is highly likely that no constant factor approximation exists for TSP, as this would imply that P = NP [25].

Yet, approximation algorithms are possible for the special case of metric TSP, where the cost to travel directly from any two cities is no more expensive that any other route between them. Note that TSP is also NP-hard in this metric case [25]. In 1976, Christofides developed a $\frac{3}{2}$-approximation algorithm for metric TSP i.e., one which guarantees to produce a TSP tour of cost at most $\frac{3}{2}$ the cost of an optimal TSP tour. Christofides' algorithm relies on finding the cheapest way to visit each city at least once such that there is only a single way to travel between any two cities. Such a solution forms a tree-like structure known as a minimum cost spanning tree. Using this minimum cost spanning tree, the algorithm adds a particular set of edges in a minimum cost way such that each node in this spanning tree is connected to an even number of nodes.

Despite much effort, Christofides remains the best approximation algorithm for metric TSP to this date, even after all these years. However, there is a long-standing conjecture which states that the worst-case ratio between metric TSP and its usual linear programming relaxation lower bound is $\frac{4}{3}$. This conjecture has led many to believe that a $\frac{4}{3}$-approximation for metric TSP is possible.

One approach to improve upon Christofides' result would be to look into the effect of applying Christofides' algorithm on a different spanning tree, not necessarily one of minimum cost. In fact, using a more expensive spanning tree may require Christofides' algorithm to add a less expensive set of edges to this choice of tree, resulting in a cheaper solution overall. In general, it is not yet obvious how such a tree can be found. A possible approach is to examine a collection of spanning trees, and output the tree which yields the cheapest Christofides type of solution. In fact, a derandomized version of an algorithm proposed by Gharan *et al.* [20] takes such an approach; given a collection of spanning trees based on a relaxation of TSP, apply Christofides' algorithm to each tree in the collection and output the cheapest solution. This algorithm is known as the *Best-of-Many Christofides'* algorithm. In an experimental evaluation of the Best-of-Many Chrisfofides' algorithm, Genova and Williamson [19] have shown that, over a particular set of test data, the cost of the solution produced by the column generation and splitting off variants of the Best-of-Many Christofides' algorithm is on average 4.93% and 5.25% away from the cost of an optimal solution, respectively. In contrast, their results indicate that on average, over the same test data, the the cost of the solution produced by the standard Christofides' algorithm is 9.66% away from the cost of an optimal solution.Although the Best-of-Many Christofides' algorithm has shown to be successful in practice, its performance guarantee is exactly that of Christofides' i.e., $\frac{3}{2}$. Moreover, it is much slower than Christofides' algorithm, and thus is not practical for large instances.

In this thesis, we pursue the idea of finding a better spanning tree for Christofides' algorithm by investigating a different collection of spanning trees than that used in the Best-of-Many Christofides' algorithm [19, 20]. Indeed, the collection of trees we investigate has been used successfully by Mömke and Svensson [44] to approximate a simpler case of TSP called graph-TSP. In doing so, we adapt and extend some of Mömke and Svensson's nice ideas to metric TSP, and, in the process, develop a new and innovative heuristic for metric TSP. We devise a way to efficiently find a TSP tour without having to examine each tree individually, and thus are able to easily solve much larger TSP instances than the Best-of-Many Christofides' algorithm. In fact, we are able to test our heuristic on the largest metric TSP instance in the TSPLIB library [37] on 85,900 nodes, which is far greater than the largest instance tested by Genova and Williamson [19] which has 3694 nodes. We implement this heuristic and demonstrate, experimentally, that as well as being much more efficient, it produces solutions that are almost as good as the Best-of-Many Christofides' algorithm. In fact, over the same test data as Genova and Williamson [19], we find that the cost of the solution produced by our heuristic is on average 4.78% away from the cost of an optimal solution, and 0.83% and 0.33% away from the cost of the solution produced by the column generation and splitting off variants of the Best-of-Many-Christofides' algorithm, respectively.

A second approach taken in this thesis, for trying to improve upon Christofides' $\frac{3}{2}$ result, is to look at special cases of metric TSP. Sometimes, ideas developed for special cases may be useful for more general forms of the problem. Indeed, there has been a great deal of progress for the special case of *graph-TSP*, in which the cost of edges uv is the cost of a shortest path between nodes u and v in an underlying connected graph whose edges have cost one. Similar to its counterparts, graph-TSP is known to be an NP-hard problem [22]. Yet, Mömke and Svensson [44] have developed a $\frac{4}{3}$-approximation algorithm for a special class of graphs, namely 2-edge connected graphs with degree at most three. For general instances of graph-TSP, Mömke and svensson [44] gave a 1.462-approximation algorithm, and Sebő and Vygen [42] gave a $\frac{7}{5}$-approximation algorithm, the latter being the best approximation for graph-TSP to date.

In this second part of this thesis, we take a closer look at the special case of graph-TSP, and investigate ways to improve upon the current best approximation guarantee of $\frac{7}{5}$ [42]. Indeed, the best approximation guarantee of α is often related to the worst-case ratio between the cost of an optimal solution of a problem, here graph-TSP, and the cost of an optimal solution of the natural linear programming relaxation of it. This ratio is known as the integrality gap. We successfully show that for small instances of graph-TSP, the integrality gap does not exceed $\frac{4}{3}$. In fact, we conjecture that there are at least three infinite families of graph-TSP instances for which this ratio reaches $\frac{4}{3}$ asymptotically. We also give a new $\frac{4}{3}$-approximation algorithm for specific connected graph-TSP instances that have degree at most four. This algorithm is similar to the $\frac{4}{3}$-approximation developed by Newman [34], however, it avoids having to solve a minimum cost circulation problem as part of the solution.

1.1 Thesis contributions

The main contributions of this thesis are as follows:

1. We show there is an equivalent method to that of Christofides' method of forming a spanning Eulerian multi-subgraph, and conclude that such a method yields a $\frac{3}{2}$-approximation algorithm for metric TSP when applied to a minimum cost spanning tree.

2. We adapt and extend ideas successfully used by Mömke and Svensson [44] for

approximating graph-TSP to metric TSP, and develop a novel and practical heuristic which implicitly finds a TSP tour amongst a collection of spanning trees that may be exponential in size. We show experimentally that this heuristic runs fast, even on large instances, and that the collection of spanning tree produces good solutions. Indeed, we show that the cost of the solution produced by our heuristic is on average 4.78% away from the cost of an optimal solution, over all test data used by Genova and Williamson [19]. Furthermore, over the same test data, we show that the cost of the solution produced by our heuristic is on average 0.83% and 0.33% away from the cost of the solution reported by Genova and Williamson [19] for their implementation of the column generation and splitting off variants of the Best-of-Many Christofides' algorithm, respectively.

3. We prove some results regarding T-joins, which prove to be instrumental in the development of our heuristic. These results may be useful in future algorithms developed for metric TSP.

4. We conduct a computational study to investigate the integrality gap for small instances of graph-TSP. In particular, we show that the integrality gap does not exceed $\frac{4}{3}$ for general connected graphs on $6 \leq n \leq 10$ nodes, and for cubic and subcubic connected graphs on $6 \leq n \leq 16$ nodes, improving upon $\frac{7}{5}$ [42]. For general connected graph-TSP instances, we characterize the graphs that give the integrality gap for each corresponding value of n, and propose a stronger conjecture with regards to the integrality gap of graph-TSP, which depends of the number of nodes n. Additionally, we conjecture that there are three families of infinite graph-TSP instances for each value of n modulo 3, which yield an

integrality gap of $\frac{4}{3}$ asymptotically.

5. We propose a simplified $\frac{4}{3}$-approximation algorithm for particular graph-TSP instances in which every nodes has degree at most four.

1.2 Thesis outline

We finish this chapter with a literature review on metric TSP and graph-TSP. Following this, we structure the remainder of this thesis as follows:

In Chapter 2, we introduce to the reader general definitions and notations with regards to graph theory, integer linear programming, polyhedral theory and heuristics. We also focus on TSP, its integer linear programming formulation and important approximation algorithms.

In Chapter 3, we study T-joins and special cases of T-joins, specifically odd joins and $\hat{U}$-tight odd joins. We study their polytopes and the methods with which we can find such objects. We take a closer look at $\hat{U}$-tight odd joins and highlight special vectors in its polytope which show the existence of special $\hat{U}$-tight odd joins.

In Chapter 4, we discuss the Mömke and Svensson method of forming a spanning Eulerian multi-subgraph, and show that it is equivalent to that of Christofides' method of forming a spanning Eulerian multi-subgraph. Furthermore, we show how we can adapt and extend the ideas proposed by Mömke and Svensson [44] for approximating graph-TSP to metric TSP, and how we can combine these ideas with our results for $\hat{U}$-tight odd joins to develop a new efficient heuristic for metric TSP. We take special care in discussing the results of an experimental evaluation of our heuristic, and how they compare to the results obtained from an experimental evaluation of the Best-of-Many Christofides' algorithm by Genova and Williamson [19], with a particular interest in the results they report for their implementation of the column generation and splitting off variants.

In Chapter 5, we investigate, by means of a computational study, the integrality gap for small instances of graph-TSP. We focus on three particular classes of graphs, including general connected graphs on $6 \leq n \leq 10$ nodes. We characterize the graphs that give the integrality gap for each corresponding value of n, from which we further strengthen the $\frac{4}{3}$ integrality gap conjecture.

In Chapter 6, we propose a simplified $\frac{4}{3}$-approximation algorithm for particular 2-edge connected graph-TSP instances with maximum degree four, and propose ways in which the ideas used for this one can be extended beyond these graphs.

In Chapter 7, we make some concluding remarks and suggestions for future work.

1.3 Literature review

In 1976, Christofides [9] presented a $\frac{3}{2}$-approximation for metric TSP, which remains the best approximation to date. It was later shown by Shmoys and Williamson [43], as well as Wolsey [45], that the integrality gap in the metric case is at most $\frac{3}{2}$. In fact, in a computational study of the integrality gap, Benoit and Boyd [3] have shown that the integrality gap is at most $\frac{4}{3}$ for small instances up to ten nodes and for special graphs. This is further supported by Boyd and Carr [6], which show that the integrality gap is $\frac{4}{3}$ for cost functions optimized at certain graphs.

In the special case of graph-TSP, there has been a great deal of progress in recent years. Gharan et al. [20] proposed a randomized polynomial-time algorithm with approximation ratio of $\left(\frac{3}{2} - \varepsilon\right)$ for some small value $\varepsilon > 0$, which provides a slight improvement on Christofides' $\frac{3}{2}$ approximation for metric TSP. Following suit, Mömke and Svensson [44] proposed a 1.461-approximation algorithm for graph-TSP. In fact, when applied to 2-edge connected graphs with degree at most three, the performance guarantee improves to $\frac{4}{3}$. Furthermore, they show that for those particular graphs, the integrality gap is $\frac{4}{3}$. Due to an improved analysis of the Mömke and Svensson's 1.461-approximation for graph-TSP, Mucha [33] was able to improve their performance guarantee to $\frac{13}{9}$. At present, Sebő and Vygens [42] hold the best approximation guarantee for general graph-TSP. They propose a polynomial-time $\frac{7}{5}$-approximation algorithm for general graph-TSP and show that this one yields an integrality gap of at most $\frac{7}{5}$.

Chapter 2

Preliminaries

In this chapter, we present a number of definitions and notation relevant to graph theory, integer programming and polyhedral theory, all of which will be utilized throughout this thesis. We also take special care in discussing metric TSP, particularly, its integer linear programming formulation and relaxation, as well as the integrality gap and heuristics for it.

2.1 Graph Theory

Here we present some definitions and notation relating to graph theory that will be utilized in this thesis. For a detailed background of graph theory, we refer the reader to the text by Bondy and Murty [5].

Let $G = (V, E)$ be a *graph* with disjoint finite sets V of nodes and E of edges. Where it may be ambiguous, we write $V(G)$ instead of V and, similarly, we write $E(G)$ instead of E. We use n and m to represent $|V|$ and $|E|$, respectively. For any node $u \in V$, we denote its degree by $deg(u)$ i.e., the number of edges incident u in

G. Each edge of the graph consists of a pair of nodes, which we say are its *ends*. If an edge e is undirected, we write $e = uv$ to denote its ends. Similarly, if an edge is directed, we write $e =< u, v >$ to denote its ends, and say that node u is its *tail* and v is its *head*. A graph is *subcubic* if each node has degree at most three, *cubic* if it is 3-regular and *subquartic* if each node has degree at most four.

Let $U \subseteq V$ and $F \subseteq E$ be *subsets* of node set V and edge set E, respectively. We write $V \setminus U$, or $V - U$, to denote the set of nodes in V not in U. Similarly, we write $E \setminus F$, or $E - F$, to denote the set of edges in E not in F. We use $\bar{U} := V \setminus U$ and $\bar{F} := E \setminus F$. We define a *cut* $\delta(U)$ to be the set of edges with one end in U and the other in $\bar{U}$, and write $\delta_G(U) := \delta(U) \cap E(G)$ where it may be ambiguous. We define $\gamma(U)$ as the set of edges with both ends in U, and write $\gamma_G(U) := \gamma(U) \cap E(G)$ where necessary. For $k \geq 1$, a *k-edge cut* is one for which $|\delta(U)| = k$.

A *TSP tour* of G corresponds to a Hamiltonian cycle in G. We say a graph G is *Eulerian* if it is connected and each node has even degree. Any such graph G contains an *Euler tour*, which consists of a closed walk in which each edge of G appear exactly once. An Euler tour can be found in $O(m)$ time [13]. Note that we can obtain a TSP tour by *shorcutting* an Euler tour, that is, we can obtain a TSP tour by walking along the edges of an Euler tour and skipping over any nodes previously visited in the walk, as seen Figure 2.1.1.

Let $c \in \mathbf{R}^E$ be a cost function indexed by the edges of graph G, where $\mathbf{R}$ represents the set of real numbers. We write $c_e := c(e)$ to denote the cost of an edge $e \in E$, and write $c(G)$ to mean $c(E)$. We say that c is *metric* if the costs are symmetric, non-negative and satisfy the triangle inequality i.e., $c_{uw} + c_{wv} \geq c_{uv}$ for any node $u, v, w \in V$. We say that c is *graph metric* if the cost of edges uv of G is equal to

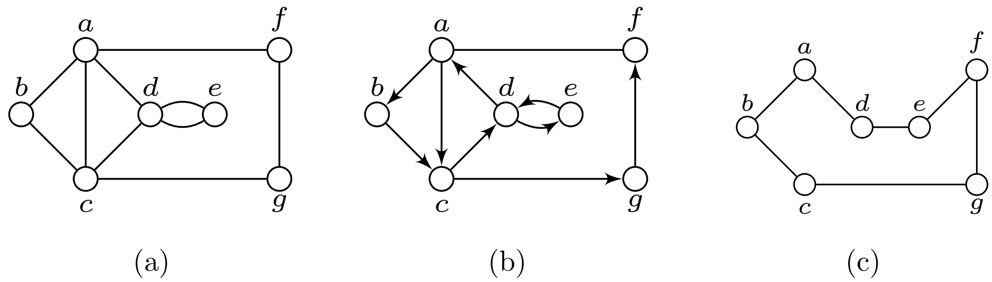


Figure 2.1.1: Example of shortcutting. (a) An Eulerian graph. (b) An Euler tour of the Eulerian graph in which the nodes are visited in the following order: $e, d, a, b, c, g, f, a, c, d, e$. (c) The TSP tour obtained by shortcutting the Euler tour.

the cost of a shortest path between u and v in an underlying undirected unweighted connected graph.

A set $M \subseteq E$ of edges is a *matching* in G if no two edges in M share the same ends. If each node of G is an end of an edge in M, then M is said to be *perfect matching.* A *minimum cost perfect matching* is the least expensive perfect matching of G, with respect to the cost of the edges. Such a perfect matching can be computed in $O(n(m+n\log n))$ time [17].

2.2 Polyhedral theory

Here, we present some basic definitions and results related to polyhedral theory and its role for TSP. We refer the reader to Schrjiver's book [40] for further details.

A subset H of $\mathbf{R}^n$ is said to be an *affine halfspace* if for some vector $a \in \mathbf{R}^n$ and constant $b \in \mathbf{R}$ it is of the form $H = \{x \in \mathbf{R}^n : a^\top x \leq b\}$. A *polyhedron* $P \subset \mathbf{R}^n$ is defined as the intersection of finitely many halfspaces i.e., $P = \{x \in \mathbf{R}^n : Ax \leq b\}$ for some matrix $A \in \mathbf{R}^{m\times n}$ and vector $b \in \mathbf{R}^n$. Given a polyhedron P, we say that the system $Ax \leq b$ *determines* P. If $x \in P$, then we say that x is *feasible* for P, otherwise we say that it is *infeasible*. A polyhedron P need not be bounded, however, if it is, then P is said to be a *polytope*. A non-empty subset F of a polyhedron P is a *face* of P if it is of the form $F = \{x \in P : A'x = b'\}$ where $A'x \leq b'$ is a subsystem of $Ax \leq b$. Note that $F = P$ is a face of P. A *vertex* of P is the unique solution that corresponds to the intersection of finitely many linearly independent equations from the system $Ax = b$. An *integer polyhedron* is a polyhedron in which all of its vertices are integer valued. The *incidence vector* χ^F of a subset set $F \subseteq E$, is defined by:

$$\chi_e^F = \begin{cases} 1 & \text{if } e \in F, \\ \\ 0 & \text{if } e \notin F. \end{cases}$$

Let $x_1, x_2, \ldots, x_k \in \mathbf{R}^n$. We say that $x \in \mathbf{R}^n$ is a *convex combination* of the points $x_1, x_2, \ldots, x_k$ if

$$x = \sum_{i=1}^{k} \lambda_i x_i,$$

for real numbers λ_i such that $0 \leq \lambda_i \leq 1$ for $i = 1, 2, \ldots, k$ and $\lambda_1 + \lambda_2 + \cdots + \lambda_k = 1$. The following result will be useful in the chapters to come.

Theorem 2.2.1. *Let $P \subset \mathbf{R}^n$ be a polytope, and let $c \in \mathbf{R}^n$. If $x \in \mathbf{R}^n$ can be written as a convex combination of points $X \subset P$, then there is at least one point $x' \in X$ such that $c^\top x' \leq c^\top x$.*

Proof. Suppose no such point exists i.e., $c^\top x' > c^\top x$ for all $x' \in X$. As x can be written as a convex combination of points $x_1, x_2, \ldots, x_k \in X$, we have

$$x = \sum_{i=1}^{k} \lambda_i x_i,$$

for real numbers λ_i such that $0 \leq \lambda_i \leq 1$ for $i = 1, 2, \ldots, k$ and $\lambda_1 + \lambda_2 + \cdots + \lambda_k = 1$. Multiplying both sides of the equation with $c^\top$, yields

$$c^\top x = \sum_{i=1}^{k} \lambda_i c^\top x_i > \sum_{i=1}^{k} \lambda_i c^\top x = c^\top x.$$

Thus, we have reached a contradiction. □

The set of all points $x \in \mathbf{R}^n$ that can be expressed as a convex combination of points in $X \subset \mathbf{R}^n$ is the *convex hull* of X. Note that a polytope is the convex hull of its vertices and can be defined as such. Furthermore, the convex hull characterizes the vertices of the polytope it defines. Generally, we know either the vertices or the halfspaces for a polytope. However, in some cases we know both, which may be extremely useful. For example, let $G = (V, E)$ be a graph, and for any $F \subseteq E$, let $x(F) = \sum_{e \in F} x_e$. Then, the *spanning tree polytope*, which is the convex hull of incidence vectors of spanning trees of G, is determined by the following system for $x \in \mathbf{R}^E$ [41]:

$$\begin{array}{llll}
(2.2.1) \qquad & \text{(i)} & x(E) = |V| - 1, & \\
& \text{(ii)} & x(\gamma(U)) \leq |U| - 1 & \text{for all non-empty } U \subseteq V, \\
& \text{(iii)} & x_e \geq 0 & \text{for all } e \in E.
\end{array}$$

Thus, if $x \in \mathbf{R}^E$ satisfies the constraints of (2.2.1), then it can be written as a convex combination of incidence vectors of spanning trees.

Finally, let $P \subset \mathbf{R}^E$ be a polyhedron and let $G = (V, E)$ be a graph, then the *support graph* of a point $x \in P$ is the graph $G_x = (V, E')$, where $E' \subseteq E$ corresponds to the edges e in E such that $x_e > 0$.

2.2.1 The subtour polytope

Let $K_n = (V, E)$ be a complete graph on n nodes, then the *subtour polytope* is determined by the following system for $x \in \mathbf{R}^E$:

$$\text{(2.2.2)} \qquad \begin{array}{rll} \text{(i)} & x(\delta(u)) = 2 & \text{for all } u \in V, \\ \text{(ii)} & x(\delta(U)) \geq 2 & \text{for all } U \subset V,\ 3 \leq |U| \leq n-3, \\ \text{(iii)} & 0 \leq x_e \leq 1 & \text{for all } e \in E. \end{array}$$

The subtour polytope is of interest, as the *travelling salesman polytope* that is the convex hull of all incidence vectors to TSP tours, is contained inside it. Upon closer examination of the subtour polytope, we obtain the following known results, two of which we prove for completeness.

Theorem 2.2.2. *[8] Let G_{x^*} be the support graph of a vertex x^* of the subtour polytope (2.2.2), then $|E(G_{x^*})| \leq 2n - 3$.* □

Theorem 2.2.3. *Let G_x be the support graph of a point x in the subtour polytope (2.2.2), then G_x is 2-node connected.*

Proof. Assume this is not the case, that G_x is 1-node connected. Therefore, G_x contains at least one cut node u. Consider node set U of a connected component of $G_x \setminus \{u\}$. Then, as $\delta(U) \subset \delta(u)$, it follows that $x(\delta(U)) < x(\delta(u)) = 2$ contradicting the feasibility of x for the subtour polytope (2.2.2). □

Theorem 2.2.4. *Let x be a point in the subtour polytope (2.2.2), then $\left(\frac{n-1}{n}\right) x$ is in the spanning tree polytope (2.2.1).*

Proof. Observe that $2\left(\frac{n-1}{n}\right)x(E) = \left(\frac{n-1}{n}\right)\sum_{u\in V} x(\delta(u))$, and recall that $x(\delta(u)) = 2$ for all $u \in V$. Thus, $\left(\frac{n-1}{n}\right)x(E) = \left(\frac{n-1}{n}\right)\frac{1}{2}\sum_{u\in V} x(\delta(u)) = n-1$. Now, consider $\emptyset \neq U \subseteq V$. Note that $\sum_{u\in U} x(\delta(u)) = x(\delta(U)) + 2x(\gamma(U))$, and that $x(\delta(U)) = 0$ if $U = V$. Therefore, $x(\delta(U)) \geq 0$ and thus, $\left(\frac{n-1}{n}\right)\gamma(U) = \left(\frac{n-1}{n}\right)\left(\frac{1}{2}\sum_{u\in U} x(\delta(u)) - \frac{1}{2}x(\delta(U))\right) \leq \left(\frac{n-1}{n}\right)|U| \leq |U| - 1$. Finally, given that $x_e \geq 0$ for all edges $e \in E$, it follows that $\left(\frac{n-1}{n}\right)x_e \geq 0$ for all edges $e \in E$. Thus, $\left(\frac{n-1}{n}\right)x$ satisfies the constraints of the spanning tree polytope (2.2.1). □

2.3 Integer and linear programming

We refer the readers to Schrjiver's book [40] for further details on the topic of integer and linear programming.

Given a vector $x = (x_1, x_2, \ldots, x_n) \in \mathbf{R}^n$ of *decision variables* and a vector $c \in \mathbf{R}^n$, the problem of maximizing, or minimizing a linear *objective function* $c^\top x$, over a polyhedron $P \subset \mathbf{R}^n$ determined by a set of *constraints*, is known as a *linear programming* problem, or LP.

Let $A \in \mathbf{R}^{m\times n}$ and $b \in \mathbf{R}^n$, then, in its standard form, an LP problem is formulated as follows:

$$\max\{c^\top x : Ax \leq b, x \geq 0\} \text{or} \min\{c^\top x : Ax \leq b, x \geq 0\}.$$

A vector $x' \in \mathbf{R}^n$ is said to be a *feasible solution* for the LP problem if it satisfies all of the constraints, and *infeasible* otherwise.Additionally, x' is said to be *optimal* if it maximizes $c^\top x$ for a maximization problem, or if it minimizes $c^\top x$ for a minimization problem. It is known that we can solve LP problems in a polynomial number of steps

[27]. If we restrict the decision variables to be integer, an LP problem is said to be an *integer linear programming*, or ILP, problem.

Unfortunately, much like TSP and metric TSP in general, solving an ILP problem is NP-hard [4]. However, if we remove the integer constraints we obtain a *linear programming relaxation* of the original ILP problem, which we know we can solve in polynomial time. Furthermore, the solution to this relaxation provides a lower bound for minimization ILP problems and an upper bound for maximization ILP problems.

2.3.1 Integer linear programming formulation of TSP

Let $K_n = (V, E)$ be a complete graph on n nodes and m edges, and let $c \in \mathbf{R}^E$ be a non-negative cost function indexed by the edges of G. Observe that any TSP tour can be represented by its incidence vector. Therefore, we have the following ILP formulation of TSP where $x \in \mathbf{R}^E$:

$$\text{minimize} \sum (c_e x_e : e \in E) \tag{2.3.1}$$

$$\text{subject to} \quad x(\delta(v)) = 2 \quad \text{for all } v \in V, \tag{2.3.2}$$

$$x(\delta(S)) \geq 2 \quad \text{for all } S \subset V,\ 3 \leq |S| \leq n-3, \tag{2.3.3}$$

$$0 \leq x_e \leq 1 \quad \text{for all } e \in E. \tag{2.3.4}$$

$$x_e \text{ integer} \quad \text{for all } e \in E. \tag{2.3.5}$$

The constraints defined by (2.3.2) are called the *degree constraints* and the constraints defined by (2.3.3) are called the *subtour elimination constraints.* Constraints

(2.3.4) and (2.3.5) are known as the bound constraints and integer constraints, respectively. Note that there are n degree constraints, an exponential number of subtour elimination constraints with respect to n, and m bound and integer constraints.

Note that by relaxing the integer constraint, that is removing constraint 2.3.5, we obtain the LP relaxation for TSP which is known as the *Subtour Elimination Problem*, or *SEP*. This LP relaxation can be solved in polynomial-time using the ellipsoid method [23], however, a practical polynomial-time algorithm is not yet known. Observe that the constraints of this relaxation are precisely those that determine the subtour polytope (2.2.2).

2.4 Integrality gap

Let $G = (V, E)$ be a graph with cost function $c \in \mathbf{R}^E$, and let $OPT(G)$ and $OPT_{LP}(G)$ be optimal values for the ILP problem and LP relaxation, respectively. We note that in the case of maximization, $OPT_{LP}(G)$ provides an upper bound for $OPT(G)$, and similarly, in the case of minimization, $OPT_{LP}(G)$ provides a lower bound for $OPT(G)$. We examine the quality of the the lower bound, or upper bound, by evaluating the ratio between $OPT(G)$ and $OPT_{LP}(G)$. The closer this ratio is to one, the better the bound. However, it is possible that we might be unlucky, and that the lower bound, or upper bound, isn't close to the optimal value. In fact, by evaluating the worst-case ratio between $OPT(G)$ and $OPT_{LP}(G)$ over all graphs G and all possible cost functions $c \in \mathbf{R}^E$, we can identify just how unlucky we can get. This ratio is known as the *integrality gap*.

2.4.1 Integrality gap and TSP

For metric TSP, there is a long-standing conjecture concerning the integrality gap for SEP, which is as follows:

Conjecture 2.4.1. *The integrality gap for SEP for metric TSP is* $\frac{4}{3}$.

It is known that the integrality gap is at least $\frac{4}{3}$, for there is a well-known infinite family of graphs with particular edge costs that reaches an integrality gap of $\frac{4}{3}$ asymptotically. Indeed, consider graph G in Figure 2.4.1(a) which consists of two triangles joined together by three disjoint paths of length k.

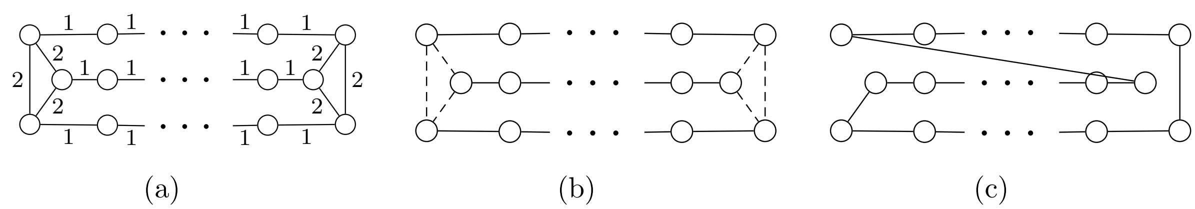


Figure 2.4.1: Infinite family of metric TSP instances for which the integrality gap reaches $\frac{4}{3}$ asymptotically.

Let the cost of the edges uv not shown in Figure 2.4.1(a) be the cost of a shortest path between nodes u and v in the graph, and consider the corresponding optimal subtour LP solution x^*, where $x^*_e = 1$ for the solid edges and $x^*_e = 1/2$ for the dashed edges, as seen in Figure 2.4.1(b). Then, with respect to k, the cost of the optimal LP solution is $OPT_{LP}(G) = 3k + 6$, and the cost of the correspond optimal TSP tour, or ILP solution, is $OPT(G) = 4k + 6$. Then, the ratio between $OPT(G)$ and $OPT_{LP}(G)$, is

$$\frac{OPT(G)}{OPT_{LP}(G)} = \frac{4k+6}{3k+6}.$$

This ratio reaches $\frac{4}{3}$ as $k \to \infty$.

In light of this example, if Conjecture 2.4.1 were true, then $\frac{4}{3}$ would be the best integrality gap possible. Note that the same conjecture applies in the special case of *graph-TSP*, in which the cost of edges uv is the shortest path between nodes u and v in an underlying connected graph whose edges have cost one.

2.5 Heuristics

Given the absence of efficient methods to solve difficult problems to optimality, it is necessary to seek other methods which can produce a feasible solution for the problem at hand. *Heuristics* are examples of such methods. Although usually quite fast, such methods do not always guarantee that the solution it produces is optimal. However, there are heuristics which have a performance guarantee, that is, for some positive constant $\alpha \geq 1$, the cost of the solution produced is at most α times the cost of an optimal solution, assuming we are minimizing. Such heuristics are known as *α-approximation algorithms.*

2.5.1 Christofides' algorithm

Christofides [9] developed a $\frac{3}{2}$-approximation algorithm for metric TSP. Given a connected graph $G = (V, E)$, *Christofides' algorithm* begins by computing a minimum cost spanning tree $T^* = (V, F^*)$ of G, which can be done in $O(m + n \log n)$ time [41]. It then finds a minimum cost perfect matching M^* of the subgraph of G induced by the odd degree nodes in T^*. The algorithm then combines the edge sets M^* and F^* to form a spanning Eulerian multi-subgraph $H^* = (V, E^*)$ of G. As H^* is Eulerian, it must have an Euler tour. The algorithm shortcuts this Euler tour to obtain the final TSP tour.

We now show that Christofides' algorithm yields a $\frac{3}{2}$-approximation algorithm for metric TSP. However, we first show that the cost of the TSP tour obtained through shortcutting is no more expensive than the cost of the spanning Eulerian multi-subgraph H^*.

Lemma 2.5.1. *Let $K_n = (V, E)$ be the complete graph on n nodes with metric edge costs $c \in \mathbf{R}^E$, and let $H^* = (V, E^*)$ be a spanning Eulerian multi-subgraph of K_n. Then, the TSP tour obtained by shortcutting H^* has cost at most $c(E')$.*

Proof. Consider edge uv in the TSP Tour. If edge uv is the result of skipping over some node w in H^*, then $c_{uw} + c_{wv} \geq c_{uv}$ as c is metric. If edge uv is an edge in H^*, then its cost in the TSP tour is exactly its cost in H^* i.e., c_{uv}. Thus, the cost of each edge in the final TSP tour is at most the cost of its corresponding edge, or path, in H^*. Therefore, the cost of the final TSP tour is at most the cost of H^*. $\square$

Theorem 2.5.2. *Let $K_n = (V, E)$ be the complete graph on n nodes with metric edge costs $c \in \mathbf{R}^E$. Then, the Christofides' algorithm yields a $\frac{3}{2}$-approximation for metric TSP.*

Proof. Let Q^* be an optimal TSP tour for K_n. Then, removing an edge from Q^* forms a spanning tree of cost $c(T) \leq c(Q^*)$, and thus for a minimum cost spanning tree T^* of K_n, we have that $c(T^*) \leq c(Q^*)$. Let W be the set of odd degree nodes in T^*, and let Q' be a cycle through the nodes of W, visited in the same order as in Q^*. As $|W|$ is even, there are an even number of edges in Q', and therefore we can partition the edges of Q' into two perfect matchings. Since the costs satisfy the triangle inequality, it must be that the cost of Q' is at most the cost of Q^*. Then, for any minimum cost perfect matching M^* of G, we have that $c(M^*) \leq \frac{1}{2}c(Q') \leq \frac{1}{2}c(Q^*)$. The cost of the

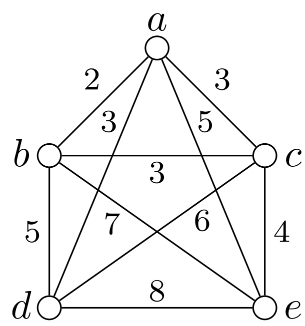
Eulerian graph H^* is

$$c(H^*) = c(T^*) + c(M^*) \leq c(Q^*) + \frac{1}{2}c(Q^*) = \frac{3}{2}c(Q^*).$$

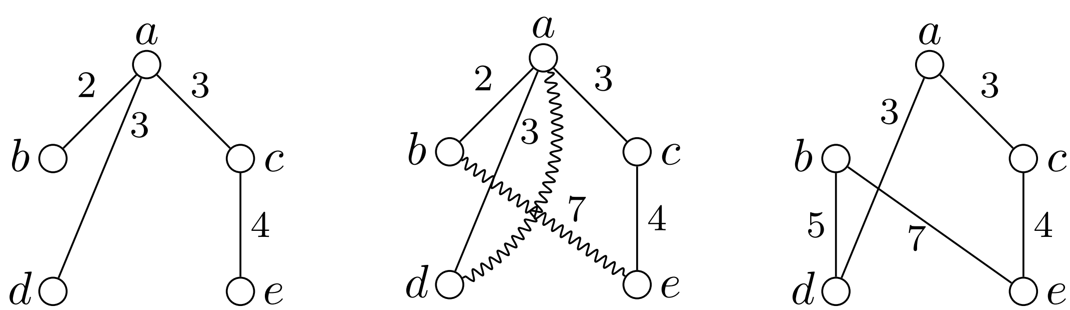
It follows from Lemma 2.5.1 that the final TSP tour obtained through shortcutting cannot be more expensive than H^*, and thus it has cost at most $\frac{3}{2}c(Q^*)$. $\square$

2.5.2 Best-of-Many Christofides' algorithm

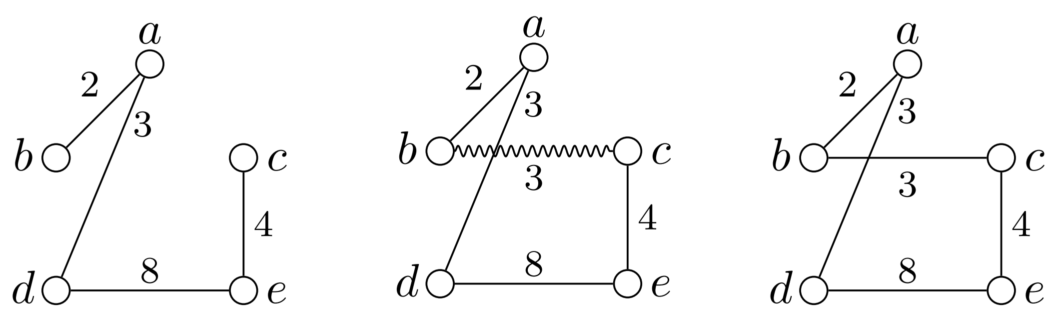
There may be a tree other than the minimum cost spanning tree which yields a better TSP tour than that produced by Christfodes, as seen in Figure 2.5.1. In an effort to find such a tree, one approach is to evaluate a collection of trees. This is the approach taken by Gharan *et al.* [20] in their *Best-of-Many Christofides'* algorithm. Let x^* be the optimal subtour LP solution for the complete weighted graph K_n on n nodes, then we know by Theorem 2.2.4 that we can write $\left(\frac{n-1}{n}\right) x^*$ as a convex combination of incident vectors of spanning trees. In a derandomized version of their algorithm, Christofides' algorithm is applied to every spanning tree in this convex combination, and the cheapest TSP tour produced is returned. It is not shown here, but the cost of the TSP tour produced will be at most $\frac{3}{2}$ the cost of the optimal TSP tour.



(a)



(b)



(c)

Figure 2.5.1: Example which illustrates how applying Christofides' algorithm on more expensive spanning tree can yield a cheaper TSP tour. (a) Graph on five nodes and five edges with specified metric edge costs. (b) Christofides' algorithm applied to the minimum cost spanning tree, yielding a TSP tour of cost 22. (c) Christofides' algorithm applied to a more expensive spanning tree, yielding a TSP tour of cost 20. Note that the undulated edges in both (b) and (c) represent the edges of the corresponding minimum cost perfect matchings.

Chapter 3

T-joins and special cases

In this section, we discuss T-joins and special cases of T-joins, namely odd joins and $\hat{U}$-tight odd joins. In Section 3.1, we introduce the concept of a T-join, define the associated polytope and discuss various methods to solve the problem of finding a minimum cost T-join. In Section 3.2, we focus on a special case of T-joins, that of odd joins. We define the polytope that defines odd joins and discuss methods to solve the minimum cost odd join problem. In Section 3.3, we further restrict ourselves to a special case of odd joins, namely $\hat{U}$-tight odd joins. We define the polytope for this case as well as discuss methods of finding a minimum cost tight odd join. Additionally, we investigate vectors feasible for the $\hat{U}$-tight odd join polytope, as they prove to be useful in tackling approximations for graph-TSP and metric TSP.

3.1 T-joins

Let $G = (V, E)$ be a graph and let $T \subseteq V$. A subset $J \subseteq E$ is a *T-join* for G if the odd degree nodes in the subgraph (V, J) are exactly the nodes in T (see Figure

3.1.1). Observe that if a T-join exists, it must be that $|T|$ is even as there are an even number of odd degree nodes in any graph.

Consider a subset $S \subseteq V$ such that $|S \cap T|$ is odd. Such a subset S is said to be T-*odd*, and its corresponding cut, $\delta(S)$, is said to be a T-*cut*. Given there are an odd number of T nodes in any T-odd set S, and that no T-join is possible on an odd number of nodes, it must be that there is at least one edge in $\delta(S) \cap J$ for any T-join J. In fact, given a graph $G = (V, E)$ and non-negative edge cost function $c \in \mathbf{R}^E$, we can formulate the minimum cost T-join problem as the following LP [11]:

$$\text{(3.1.1)} \qquad \begin{aligned} &\text{minimize } \sum (c_e x_e : e \in E) \\ &\text{subject to} \\ &x(D) \geq 1 \quad \text{for all } T-\text{cuts } D, \\ &x_e \geq 0 \quad \text{for all } e \in E. \end{aligned}$$

It was shown by Edmonds and Johnson [12] that for non-negative edge costs the lower bound provided by LP (3.1.1) is exact. In fact, the following set of linear inequalities for $x \in \mathbf{R}^E$ determines the *up-hull* of the T-*join polytope* [41]:

$$\text{(3.1.2)} \qquad \begin{array}{rll} \text{(i)} & x(D) \geq 1 & \text{for all } T-\text{cuts } D, \\ \text{(ii)} & x_e \geq 0 & \text{for all } e \in E. \end{array}$$

Unfortunately, LP (3.1.1) may not always be bounded if the edge costs are allowed to be negative [11]. However, for for general cost functions $c \in \mathbf{R}^E$, the optimal value

(a) (b)

Figure 3.1.1: Examples of possible T-joins of a particular graph. The solid nodes are those that belong to node set T, and the undulant edges are those in the T-join. (a) A T-join where T consists of a mixture of odd and even degree nodes in the graph. (b) A T-join where T is the set of all odd degree nodes in the graph.

of the following LP provides a minimum cost T-join [41]:

$$\text{(3.1.3)} \qquad \text{minimize } \sum(c_e x_e : e \in E)$$
$$\text{subject to}$$
$$\begin{aligned} x(\delta(U) \setminus F) + |F| - x(F) \geq 1 \quad & \text{for all } U \subseteq V,\ F \subseteq \delta(U), \\ & \text{such that } |F| + |U \cap T| \text{ is odd}, \\ 0 \leq x_e \leq 1 \quad & \text{for all } e \in E. \end{aligned}$$

It is known that the lower bound provided by LP (3.1.3) is exact [11]. In fact, the T-join polytope, which is known to be an integer polytope, is determined by the following set of linear inequalities for $x \in \mathbf{R}^E$ [41]:

$$\text{(3.1.4)} \qquad \begin{aligned} &\text{(i)} \quad x(\delta(U) \setminus F) + |F| - x(F) \geq 1 && \text{for all } U \subseteq V,\ F \subseteq \delta(U), \\ & && \text{and } |U \cap T| + |F| \text{ odd}, \\ &\text{(ii)} \quad 0 \leq x_e \leq 1 && \text{for all } e \in E. \end{aligned}$$

We now show that the system that determines the up-hull polytope (3.1.2) does not determine the T-join polytope (3.1.4). Consider the graph $G = (V, E)$ in Figure 3.1.2(a), and let $T = \{a, c\}$. Define the vector $x^* \in \mathbf{R}^E$ as follows $x^*_{ab} = 1$, $x^*_{ac} = 1$ and $x^*_{bc} = 0$, as shown in Figure 3.1.2(a). For any non-empty subset $S \subset V$ in G, it must be that $x^*(\delta(S)) \geq 1$, and therefore for any T-cut D, we have $x^*(D) \geq 1$. By definition of x^*, it is clear that $x^*_e \geq 0$ for all edges e in E, and thus x^* satisfies both constraints of the up-hull polytope (3.1.2). Now, consider the sets $U = \{a, c\}$ and $F = \{ac\}$, then $|U \cap T| + |F|$ is odd and we have that:

$$x^*(\delta(U) \setminus F) + |F| - x^*(F) = 0 + 1 - 1 = 0 < 1.$$

Consequently, x^* violates constraint (3.1.4)(i), and thus is not feasible for the T-join polytope (3.1.4).

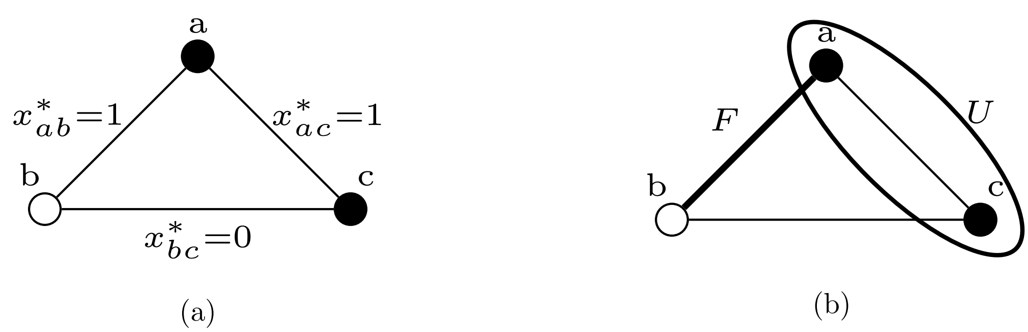


Figure 3.1.2: Example of a graph $G = (V, E)$ and a vector $x^* \in \mathbf{R^E}$ that shows not all feasible vectors in the up-hull polytope (3.1.2) are feasible for the T-join polytope (3.1.4). The solid nodes are those in the set T. (a) The graph G and the corresponding x^*-value of each edge. (b) The sets $U = \{a, c\}$ and the edge $F = \{ab\}$ which show x^* is not feasible for the T-join polytope (3.1.4).

3.1.1 Solving the minimum cost T-join problem

Given a graph $G = (V, E)$ with edge cost function $c \in \mathbf{R}^E$ and $T \subseteq V$ such that $|T|$ is even, the *minimum cost T-join problem* consists of finding a minimum cost T-join for G. If the cost of each edge is non-negative, then we may find such a T-join by solving linear programming formulation (3.1.1) directly. Additionally, Edmonds and Jackson [12] propose solving the minimum cost T-join problem by reducing this problem to a minimum cost perfect matching problem. Indeed, first form the complete graph on node-set T, and let the cost of each edge uv in this graph be the cost of a shortest path between u and v in G. Then, compute the minimum cost perfect matching M^* in this complete graph. Recall that a minimum cost perfect matching can be computed in $O(n(m + n \log n))$ time [17]. This perfect matching determines a pairing of the nodes in T. The minimum cost T-join can be obtained by taking the symmetric difference of the edge set of the shortest paths that correspond to an edge in the minimum cost perfect matching [11, 15].

For general cost functions, it is possible to find the minimum cost T-join by solving the linear programming formulation (3.1.3). Lovász and Plummer [36] proposed an algorithm which reduces the minimum cost T-join problem for general cost functions to the minimum cost T-join problem for non-negative cost functions. Using such an algorithm, the minimum cost T-join problem for general costs can be solved in time $O(n^4)$ [11]. Additionally, Schrijver [39] and Grötschel *et al.* [23] propose a reduction to the minimum cost perfect matching by expanding each node of the input graph into a clique of size equal its degree, or equal its degree with an additional node, such that a clique has odd size if its corresponding node is in the set T, and even size if its corresponding node is not in T, respectively. Finally, Edmonds and Jackson [12] also propose reducing the minimum cost T-join for general cost functions to a single capacitated b-matching problem, which can be solved in $O(n^2 m \log(n^2/m))$ time [30].

3.2 Odd joins

Let $G = (V, E)$ be a graph and let T denote the set of all of all odd degree nodes in G. Then, a T-join for G is said to be an *odd join*, see Figure 3.1.1(b). Sebő and Vygen [42] state that the following set of linear inequalities for $x \in \mathbf{R}^E$ determines the *odd join polytope*:

$$
\text{(3.2.1)} \qquad
\begin{array}{lll}
\text{(i)} & x(\delta(U) \setminus F) + |F| - x(F) \geq 1 & \text{for all } U \subseteq V,\ F \subseteq \delta(U) \\
& & \text{and } |\delta(U) \setminus F| \text{ odd}, \\
\text{(ii)} & 0 \leq x_e \leq 1 & \text{for all } e \in E.
\end{array}
$$

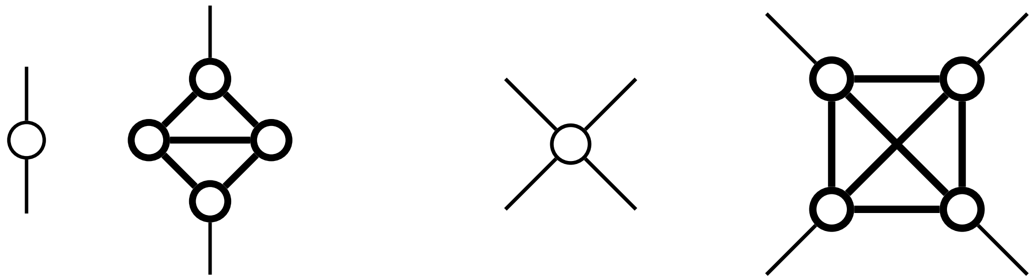
This implies the following theorem (for details of a proof see Yao [16]).

Theorem 3.2.1. *Let $G = (V, E)$ be a graph and T denote the set of all odd degree nodes in G. Then, the T-join polytope (3.1.4) and the odd join polytope (3.2.1) are equivalent.* □

Observe that for any graph G, its edge-set E is an odd join for G, and therefore any graph G has an odd-join.

3.2.1 Solving the minimum cost odd-join problem

Let $G = (V, E)$ be a connected graph with general edge costs. We can solve the minimum cost odd join problem using any of the methods proposed in Section 3.1.1.



(a) (b)

Figure 3.2.1: Examples of node expansion operations: (a) Expansion of a degree two node into a diamond. (b) Expansion of a degree four node into a clique of size four.

In fact, we focus our attention on that of the reduction of the minimum cost T-join problem to the minimum cost perfect matching problem outlined by Schrijver [39] and Grotschell et al. [23], and apply this one to the minimum cost odd join problem. For completeness, we provide a proof of the reduction for the special case of odd joins.

Before proceeding to the reduction, we first describe the expansion procedure used in the reduction. For any node u of degree-two, we introduce four new nodes, and connect them such that we form a *diamond* which consists of a cycle of length four with a chord, and such that the edges incident to u in the original graph are incident to the two nodes not connected by the chord. Otherwise, if node u has degree at least three, we introduce a number of new nodes corresponding to the degree of u in the graph, and we form a *clique* of size equal its degree by adding an edge between each pair of new nodes. Note that in this case, the edges incident to node u in the original graph are each incident to a different node in the clique. We refer to these diamonds and cliques as *gadgets*, which consist of *gadget edges* and *gadget nodes*. Figure 3.2.1 illustrates both expansion procedures, where the thicker nodes and thicker edges represent gadget nodes and gadget edges, respectively.

Theorem 3.2.2. *Let $G = (V, E)$ be a connected graph with edge cost function $c \in \mathbf{R}^E$,* *and let T denote the set of all odd degree nodes in G. Consider the graph $G' = (V', E')$, which consists of replacing every degree-two node in G with a diamond, and every other node with a clique of size equal its degree. Assign a cost of zero to each edge in E' that is not in E, and assign to each edge in E' that is in E its corresponding cost in G. Then, the problem of finding a minimum cost odd join in G can be reduced to finding a minimum cost perfect matching in G'.*

Proof. We prove the result by showing that for every odd join J for G of cost K, there exists a perfect matching of the same cost for G', and vice versa. Observe that the set of edges E of G is an odd join for G, and therefore, there must exist a minimum cost odd join. Let J be an odd join in G. Once all the nodes in G have been expanded to their respective gadgets, the edges in J form a pairwise disjoint edge set in G'. In fact, we form a matching of the subset of nodes that span the ends of the edges of the odd join in G'. In order to form a perfect matching in G', we need to match the remaining nodes. For each of the diamonds, there is an edge incident to any two unmatched nodes, of which there can be either two or four. Now, given that J is an odd join, it must be that for each clique C in G', J intersects and even (possibly zero) number of edges at every even degree node, and it must be that J intersects and odd number of edges at every odd degree node. Consequently, there is an even number of nodes in $V(C)$ that remain unmatched. As each node in $V(C)$ is adjacent to every other node in this one, we can form disjoint pairs of edges by selecting any two unmatched nodes. Add these edges to an edge set M. As such, every node in G' will be matched, and thus we have formed a perfect matching in G'. As for the cost of this perfect matching, observe that the edges in M that contribute to its cost are precisely those of J, as the edges in M that are not in J have cost zero. Therefore, the cost of the perfect matching M corresponds to the cost of the odd join J.

We now show that if there exists a perfect matching in G' then there exists an odd join in G of the same cost. Observe that the existence of a perfect matching M for G' follows from the previous argument. Let C be a clique, or a diamond, in G' with node set $V(C)$. Then, if $|V(C)|$ is even (odd), it must be that M intersects an even (odd) number of nodes in $\delta(V(C))$, otherwise there would be unmatched nodes in G'. Shrink C to its corresponding node $u \in V$ in G. Then, if the degree of u is even (odd), it must be that M intersects an even (odd) number of edges incident to u. Let J be the set of edges of G that belong to M, then it follows that $|\delta(v) \cap J|$ is even for all nodes u of even degree, and odd for all nodes u of odd degree in G. As for the cost of J, observe that each of its edges correspond to an edge in M of the same cost. Given that all other edges in $M \setminus J$ are gadget edges of cost zero, it must be that the cost of J is the same as the cost of M. □

An example of the proposed reduction is presented in Figure 3.2.2 where the solid nodes are the odd nodes of the graph, the black edges are edges of cost one, the thick nodes are the gadget nodes and the thick edges are gadget edges of cost zero. We now show that we can provide an upper bound on the number of nodes and the number of edges in the the auxiliary graph G'.

Theorem 3.2.3. *Let $G = (V, E)$ be a connected graph where, $|V| = n$ and $|E| = m$. Then, the auxiliary graph $G' = (V', E')$ has $O(m)$ nodes and $O(mn)$ edges.*

Proof. Let K represent the set of degree-two nodes in $V(G)$, and let $\bar{K} = V(G) \setminus K$. Now, in G, we expand every node in K into a diamond and all other nodes in $\bar{K}$ into a clique of size equal to its degree in G. Therefore, the total number of nodes in G'

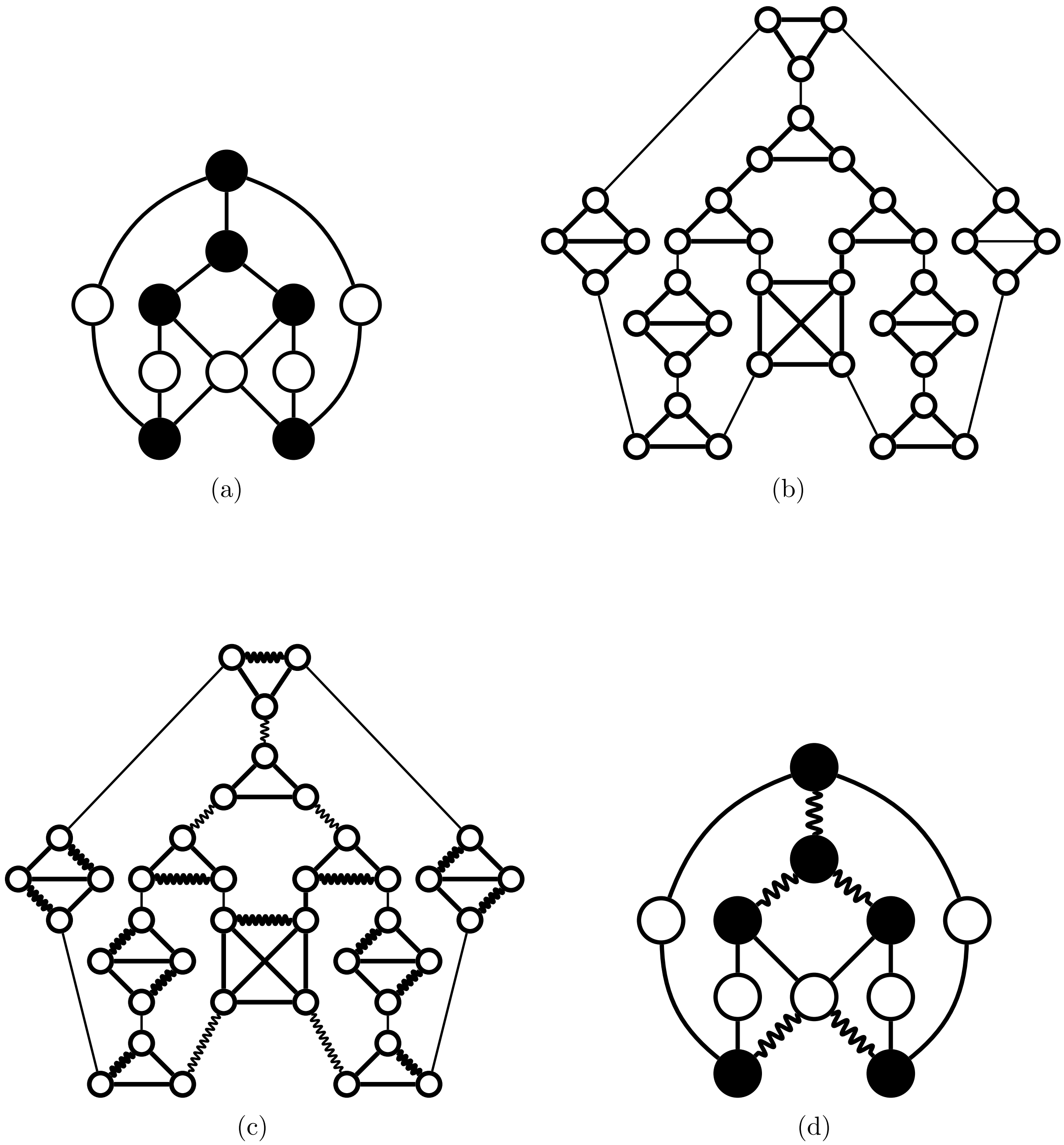


Figure 3.2.2: Example of our proposed reduction of the minimum cost odd join problem to the minimum cost perfect matching problem. (a) Input graph $G = (V, E)$. (b) Expansion G into its corresponding clique graph $G' = (V', E')$. (c) Minimum cost perfect matching of G', represented by the undulated edges. (d) Corresponding minimum cost odd join of G, represented by the undulated edges, obtained by shrinking the gagdets.

is $O(m)$. Indeed, we have that:

$$|V(G')| = 4|K| + \sum_{u \in \bar{K}} deg(u) \leq 4n + 2m \leq 6m,$$

as required.

Now, let C_i represent a clique of size $i \geq 3$. Recall that nodes of degree-two are expanded into a diamond, which consists of five gadget edges. Then, the total number of edges in G' is as follows:

$$\begin{aligned}
|E(G')| = |E(G)| + \sum_{u \in K} 5 + \sum_{u \in \bar{K}} |E(C_{deg(u)})| \\
&\leq m + 5|K| + \sum_{u \in \bar{K}} (deg(u))^2 \\
&\leq m + 5n + (n-1) \sum_{u \in \bar{K}} deg(u) \\
&\leq m + 5n + (n-1)(2m) \\
&\leq m + 5n + 2mn. \\
&\leq 8mn.
\end{aligned}$$

Consequently, there are $O(mn)$ edges in G', as required. $\square$

We make one final remark. Unfortunately, this particular reduction fails for general node-sets T. Indeed, consider a node u in T with even degree in G, then the minimum cost odd join intersects an odd number of edges incident to u. Consequently, when u is expanded into its corresponding clique C, there will be an odd number of unmatched node in $V(C)$, and therefore it is not possible to form a perfect matching. A similar argument can be made for odd degree nodes in G that are not in T.

3.3 $\hat{U}$-tight odd joins

Given a graph $G = (V, E)$ with odd degree nodes T, we are interested in particular odd joins of G, namely ones that intersect exactly one edge incident to each node in a subset $\hat{U}$ of T. We call such an odd join a *$\hat{U}$-tight odd join*. In fact, such odd joins are those for which we set constraint (i) in the odd join polytope (3.2.1) to equality for $U = \{u\}$ and $F = \emptyset$ for all $u \in \hat{U}$, or equivalently $x(\delta(u)) = 1$ for all $u \in \hat{U}$. Note, that the incidence vectors of such $\hat{U}$-tight odd joins are precisely those that lie on the face of the odd join polytope (3.2.1) where $x(\delta(u)) = 1$ for each u in $\hat{U}$ [42]. This face is itself a polytope, which we call the *$\hat{U}$-tight odd join polytope*, and is determined by the following set of linear inequalities for $x \in \mathbf{R}^E$:

(3.3.1)

(i)	$x(\delta(u)) = 1$	for all $u \in \hat{U}$,
(ii)	$x(\delta(U) \setminus F) + \lvert F\rvert - x(F) \geq 1$	for all $U \subseteq V,\ F \subseteq \delta(U)$ and $\lvert\delta(U) \setminus F\rvert$ odd,
(iii)	$0 \leq x_e \leq 1$	for all $e \in E$.

3.3.1 Special vectors for the $\hat{U}$-tight odd join polytope

In this section we investigate several vectors that satisfy all of the constraints for the $\hat{U}$-tight odd join polytope (3.3.1) and are thus in the polytope. In particular, this shows that there exists $\hat{U}$-tight odd joins with useful properties.

3.3.1.1 One-third vector

Let $G = (V, E)$ be a 2-edge connected graph, and let the vector $x^* \in \mathbf{R}^E$ be defined as $x^*_e = \frac{1}{3}$ for all edges e in E. Then we say that x^* is the *one-third vector* for G.

Theorem 3.3.1. *Let $G = (V, E)$ be a 2-edge connected graph, let T denote the set of all odd degree nodes in G and let $\hat{U}$ be a subset of all degree-three nodes in T. Then, the one-third vector x^* is in the $\hat{U}$-tight odd join polytope (3.3.1).*

Proof. It suffices to show that the one-third vector x^* satifies the constraints of the $\hat{U}$-tight odd join polytope (3.3.1). Consider $U \subseteq V$ and $F \subseteq \delta(U)$ such that $|\delta(U) \setminus F|$ is odd. Then, we have that:

$$\begin{aligned} x^*(\delta(U) \setminus F) + |F| - x^*(F) &= \frac{1}{3}|\delta(U) \setminus F| + |F| - \frac{1}{3}|F| \\ &= \frac{1}{3}|\delta(U) \setminus F| + \frac{2}{3}|F| \\ &= \frac{1}{3}\left(|\delta(U)| + |F|\right). \end{aligned} \tag{3.3.2}$$

We show that $|\delta(u)| + |F| = 3$ for any node $u \in \hat{U}$. Indeed, for any such node $u \in \hat{U}$, we have that $|\delta(u)| = 3$ and $F = \emptyset$. Therefore, it follows from equation (3.3.2) that:

$$x^*(\delta(u)) = \frac{1}{3}|\delta(u)| = 1.$$

Therefore, constraint (3.3.1) (i) is satisfied.

We now show that $|\delta(U)| + |F| \geq 3$. As $|\delta(U) \setminus F|$ is odd, we distinguish between two cases:

Case 1 $|F|$ is even. Then, $|F| \geq 0$ and $|\delta(U)|$ is odd. As G is 2-edge connected we have that $|\delta(U)| \geq 3$ and therefore $|\delta(U)| + |F| \geq 3$.

Case 2. $|F|$ is odd. Then, $|F| \geq 1$ and $|\delta(U)|$ is even. As G is 2-edge connected, we have that $|\delta(U)| \geq 2$, and therefore $|\delta(U)| + |F| \geq 3$.

It follows that $|\delta(U)| + |F| \geq 3$, and therefore by (3.3.2) x^* satisfies constraint (3.3.1)(ii).

Finally, it is clear that x^* satisfies the constraint (3.3.1)(iii) as $x^*_e = \frac{1}{3}$ for all edges e in E, and therefore $0 \leq x^*_e \leq 1$ holds. $\square$

Letting $\hat{U} = \emptyset$ in Theorem 3.3.1 we obtain the following corollary, which can also be found in [16, 42].

Corollary 3.3.2. *Let $G = (V, E)$ be a 2-edge connected graph, then the one-third vector x^* is in the odd-join polytope (3.2.1).* $\square$

3.3.1.2 One-third, two-third vector

We now investigate another vector that will be of use in a later chapter. Let A and B be two disjoint edge-sets of G, such that $A \cup B = E$, and let the vector $x^* \in \mathbf{R}^E$ be defined as follows:

$$x^*_e = \begin{cases} \dfrac{1}{3} & \text{if } e \in A, \\[2ex] \dfrac{2}{3} & \text{if } e \in B, \end{cases} \tag{3.3.3}$$

for all edges e in E. We now prove a analogous result to that proved by Yao in [16], using a similar argument.

Theorem 3.3.3. *Let $G = (V, E)$ be a 2-edge connected graph, let T denote the set of all odd degree nodes in G and let $\hat{U}$ a subset of degree-three nodes in T. Let set A*

consist of all edges in a 2-edge cut in G and all edges incident to a node in $\hat{U}$, let set B consist of all other edges in G and let $x^* \in \mathbf{R}^E$ be as defined in (3.3.3). Then, x^* is in the $\hat{U}$-tight odd join polytope (3.3.1).

Proof. Let $U \subseteq V$ and $F \subseteq \delta(U)$ be such that $|\delta(U) \setminus F|$ is odd. Partition the cut $\delta(U)$ into two sets $\delta_A(U)$ and $\delta_B(U)$, where $\delta_A(U)$ denotes the set of edges in $\delta(U)$ that belong to edge-set A, and $\delta_B(U)$ denotes the set of edges in $\delta(U)$ that belong to edge-set B. Similarly, partition the set F into two sets F_A and F_B, where F_A represents the set of edges in F that belong to edge-set A and F_B represents the set of edges in F that belong to the edge-set B. Then, we can derive the following:

$$\begin{aligned}
(3.3.4) \qquad & x^*(\delta(U) \setminus F) + |F| - x^*(F) \\
&= \frac{1}{3}|\delta_A(U) \setminus F_A| + \frac{2}{3}|\delta_B(U) \setminus F_B| + |F| - \left(\frac{1}{3}|F_A| + \frac{2}{3}|F_B|\right) \\
&= \frac{1}{3}|\delta_A(U) \setminus F_A| + \frac{2}{3}|\delta_B(U) \setminus F_B| + \frac{2}{3}|F_A| + \frac{1}{3}|F_B| \\
&= \frac{1}{3}|\delta_A(U)| + \frac{2}{3}|\delta_B(U)| + \frac{1}{3}|F_A| - \frac{1}{3}|F_B| \\
&= \frac{1}{3}|\delta(U)| + \frac{1}{3}|\delta_B(U) \setminus F_B| + \frac{1}{3}|F_A|.
\end{aligned}$$

We show that x^* satisfies constraint (3.3.1)(i). Consider node u in $\hat{U}$. Then, since $|F| = 0$, we have that $|\delta_A(u)| = 3$, $|F_A| = 0$, $|\delta_B(u)| = 0$ and $|F_B| = 0$. Consequently, $x^*(\delta(u)) = 1$, as required. We now show that x^* satisfies constraint (3.3.1)(ii). Indeed, we distinguish between two cases:

Case 1 $|\delta(U)| = 2$ (i.e, a 2-edge cut). Then, we have $|\delta_A(U)| = 2$ and thus $|F_A| = 1$. Additionally, it must be that $|\delta_B(U)| = 0$ and $|F_B| = 0$, or rather

$|\delta_B(U) \setminus F_B| = 0$. Therefore, by equation (3.3.4), we have that:

$$x^*(\delta(U) \setminus F) + |F| - x^*(F) = \left(\frac{1}{3} \times 2\right) + \left(\frac{1}{3} \times 1\right) = 1.$$

Case 2. $|\delta(U)| \geq 3$. It follows immediately from equation (3.3.4) that:

$$x^*(\delta(U) \setminus F) + |F| - x^*(F) \geq \left(\frac{1}{3} \times 3\right) + \frac{1}{3}|\delta_B(U) \setminus F_B| + \frac{1}{3}|F_A| \geq 1.$$

It is clear by definition of x^* that $0 \leq x^*_e \leq 1$ holds for all edges e in E, and therefore satisfies constraint (3.3.1)(iii). □

In fact, letting $\hat{U} = \emptyset$ in Theorem 3.3.3 we obtain the following corollary, which can also be found in [16].

Corollary 3.3.4. *Let $G = (V, E)$ be a 2-edge connected graph, then the vector x^* as defined in (3.3.3) is in the odd-join polytope (3.2.1), where set A corresponds to the set of edges 2-edge cuts in G.* □

3.3.1.3 Vector based on feasible point for the subtour polytope

Let $G_{x^*} = (V, E)$ be the support graph of a vector $x^* \in \mathbf{R}^E$ in the subtour polytope (2.2.2). Now, let A and B be two disjoint edge-sets of G, such that $A \cup B = E$, and let vector $x' \in \mathbf{R}^E$ be defined as follows:

$$x'_e = \begin{cases} \dfrac{x^*_e}{2} & \text{if } e \in A, \\[2ex] 1 - \dfrac{x^*_e}{2} & \text{if } e \in B, \end{cases} \tag{3.3.5}$$

for all edges e in E.

Theorem 3.3.5. *Let $G_{x^*} = (V, E)$ be the support graph of a feasible point x^* in the subtour polytope (2.2.2). Let T denote the set of all odd degree nodes in G and let $\hat{U}$ be a subset of nodes in T. Consider the vector $x' \in \mathbf{R}^E$ as defined in (3.3.5), and let all edges incident to a node in $\hat{U}$ be in set A. Then, vector x' is in the $\hat{U}$-tight odd join polytope (3.3.1).*

Proof. We begin as in the previous theorem. Let $U \subseteq V$ and $F \subseteq \delta(U)$ be such that $|\delta(U) \setminus F|$ is odd. Partition the cut $\delta(U)$ into two sets $\delta_A(U)$ and $\delta_B(U)$, where $\delta_A(U)$ denotes the set of edges in $\delta(U)$ that belong to edge-set A, and $\delta_B(U)$ denotes the set of edges in $\delta(U)$ that belong to edge-set B. Similarly, partition the set F into two sets F_A and F_B, where F_A represents the set of edges in F that belong to edge-set A and F_B represents the set of edges in F that belong to the edge-set B. Then, we can derive the following

(3.3.6)

$$\begin{aligned}
&x'(\delta(U) \setminus F) + |F| - x'(F) \\
&= x'(\delta_A(U) \setminus F_A) + x'(\delta_B(U) \setminus F_B) + |F| - (x'(F_A) + x'(F_B)) \\
&= x'(\delta_A(U)) + x'(\delta_B(U)) + |F| - 2x'(F_A) - 2x'(F_B) \\
&= \frac{x^*}{2}(\delta_A(U)) + \left(\mathbf{1} - \frac{x^*}{2}\right)(\delta_B(U)) + |F| - x^*(F_A) - 2\left(\mathbf{1} - \frac{x^*}{2}\right)(F_B) \\
&= \frac{x^*}{2}(\delta_A(U)) + \left(\mathbf{1} - \frac{x^*}{2}\right)(\delta_B(U)) + |F_A| + |F_B| - x^*(F_A) - 2\left(\mathbf{1} - \frac{x^*}{2}\right)(F_B) \\
&= \frac{x^*}{2}(\delta(U)) + (\mathbf{1} - x^*)(\delta_B(U)) + |F_A| - x^*(F_A) - |F_B| + x^*(F_B) \\
&= \frac{x^*}{2}(\delta(U)) + (\mathbf{1} - x^*)(\delta_B(U)) - (\mathbf{1} - x^*)(F_B) + (\mathbf{1} - x^*)(F_A)
\end{aligned}$$

$$= \frac{x^*}{2}(\delta(U)) + (\mathbf{1} - x^*)(\delta_B(U) \setminus F_B) + (\mathbf{1} - x^*)(F_A),$$

where $\mathbf{1}$ represents the all-one-vector induced by E. It follows from (3.3.6) that for all u in $\hat{U}$ we have $x'(\delta(u)) = 1$. Indeed, if $F = \emptyset$, we have that:

$$x'(\delta(u)) = \frac{x^*(\delta(u))}{2} = 1,$$

as $x^*(\delta(u)) = 2$.

Given that $x^*(\delta(U)) \geq 2$ for all $\emptyset \neq U \subset V$ and that $(\mathbf{1} - x^*)$ is a non-negative vector, it follows from Equation (3.3.6) that:

$$x'(\delta(U) \setminus F) + |F| - x'(F) \geq 1.$$

Finally, by definition of x', it is clear that $0 \leq x'_e \leq 1$ for all edges in e in E, as $0 \leq x^*_e \leq 1$ for all edges e in E. Consequently, vector x' satisfies all of the constraints of the $\hat{U}$-tight odd join polytope (3.2.1). □

Let $\hat{U} = \emptyset$ in Theorem 3.3.5, then we obtain the following corollary:

Corollary 3.3.6. *Vector x' defined in (3.3.5) is in the odd join polytope (3.2.1).* □

3.3.2 Solving $\hat{U}$-tight odd joins

Solving the minimum cost $\hat{U}$-tight odd joins can be achieved by modifying and solving the LP (3.1.3) for general T-joins, such that we specify a set of odd degree nodes for which we want to be tight. Sebő and Vygen [42] propose adding a large weight to all edges incident to a node in $\hat{U}$ and use any minimum T-join algorithm to find the minimum cost $\hat{U}$-tight odd join. The weights added are such that the cost of any $\hat{U}$-tight odd join that intersects more than one edge incident to nodes in $\hat{U}$ is more expensive than any $\hat{U}$-tight odd join that intersects exactly one edge incident to that node. Finally, we propose a generalization of the reduction presented in Section 3.2.1, which can be used to solve the minimum $\hat{U}$-tight odd join problem.

Theorem 3.3.7. *Let $G = (V, E)$ be a connected graph with edge cost function $c \in \mathbf{R}^E$ and let T denote the set of all odd degree nodes in G. Consider a vector x^* which is feasible for the $\hat{U}$-tight odd join polytope (3.3.1) and let $\hat{U}$ be the set of nodes u in G for which $x^*(\delta(u)) = 1$. Construct the graph $G' = (V', E')$, which consists of replacing every degree-two node in G with a diamond, and every other node not in $\hat{U}$ with a clique of size equal its degree. Assign a cost of zero to each edge in E' that is not in E, and assign to each edge in E' that is in E its corresponding cost in G. Then, the problem of finding a minimum cost $\hat{U}$-tight odd join for G can be reduced to finding a minimum cost perfect matching for G'.*

Proof. We begin by observing that as $x^*(\delta(v)) = 1$ for each node u in $\hat{U}$, it must be that both the minimum cost $\hat{U}$-tight odd join for G and the minimum cost perfect matching for G' intersect exactly one edge incident to u. Consequently, the result follows directly from the arguments in Theorem 3.2.2. □

An example of this reduction can be seen in Figure 3.3.1, where the vector $x^* \in \mathbf{R}^E$ is the one-third vector. Additionally, the solid nodes are the odd nodes of the graph, the black edges are edges of cost one, the thick nodes are gadget nodes and the thick edges are gadget edges of cost zero.

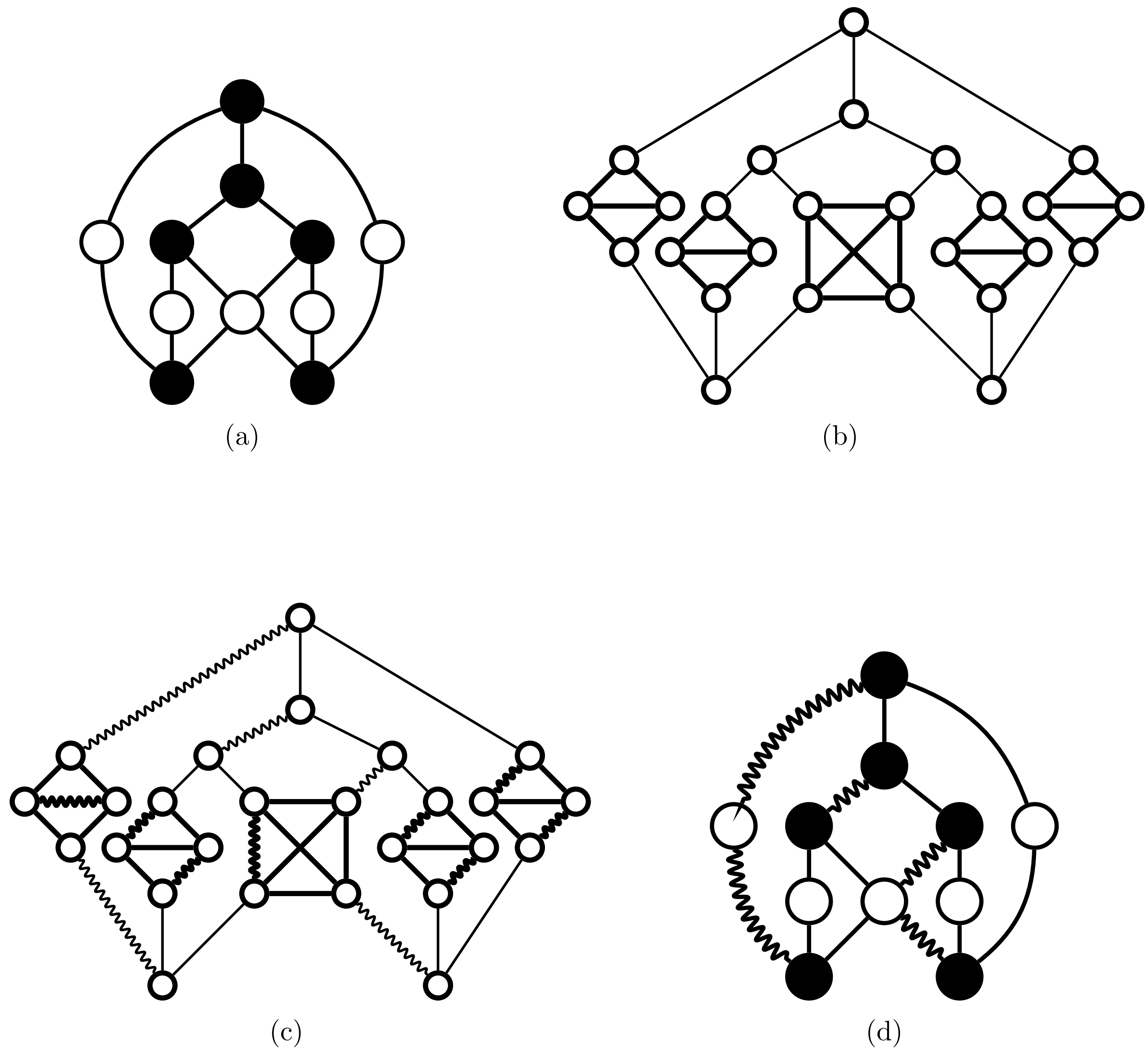


Figure 3.3.1: Example of our proposed reduction of the minimum cost $\hat{U}$-tight odd join problem to the minimum cost perfect matching problem on graph (a) Input graph $G = (V, E)$ where the solid nodes are in the set $\hat{U}$. (b) Expansion G into its corresponding clique graph $G' = (V', E')$ (c) Minimum cost perfect matching of G', represented by the undulated edges. (d) Corresponding minimum cost $\hat{U}$-tight odd join of G, represented by the undulated edges, obtained by shrinking the gadgets.

Chapter 4

A new heuristic for metric TSP

In this chapter, we describe and test a new heuristic for metric TSP. We make use of some of the clever ideas proposed by Mömke and Svensson [44] for approximating graph-TSP, and show how these ideas can be adapted to provide a heuristic for the much more general case of metric TSP. In Section 4.1, we discuss an approach equivalent to that of Christofides' to obtain a $\frac{3}{2}$-approximation for metric TSP. In Section 4.2, we describe a method to generate a collection of spanning trees, and show that the number of spanning trees in this collection can be exponential in size. In Section 4.3, we describe our proposed heuristic for metric TSP based on these spanning trees, and the approach described in Section 4.1, which only requires a single application of the minimum cost $\hat{U}$-tight odd join, on a slightly larger graph, to produce a TSP tour. In Section 4.4, we describe an experiment to test our heuristic against the results of an experimental evaluation of the standard Christofides' algorithm, as well as variants of the Best-of-Many Christofies' algorithm by Genoval and Williamson [19]. Finally, in Section 4.5 we discuss our results.

4.1 Christofides' algorithm and removing edges

Let $G = (V, E)$ be a connected graph with metric edge costs $c \in \mathbf{R}^E$, and let $T = (V, F)$ be a spanning tree of G. Recall that $\bar{F} = E \setminus F$, and that the edges in F are tree edges and those in $\bar{F}$ are non-tree edges. Christofides [9] proposed a method to form a spanning Eulerian multi-subgraph of G, which consists of adding edges to T in order to correct the nodes of T with the wrong parity. More specifically, let K be the set of odd degree nodes of T, and let D be a minimum cost K-join of G. Then, the *Christofides Method* for forming a spanning Euclidean multi-graph of G consists of adding all edges of D to T. Observe that this graph is connected, as T is itself connected, and that it has cost $c(F) + c(D)$. An example of this method applied to a particular graph is presented in Figure 4.1.1(a). We now show that the resulting graph is in fact a spanning Eulerian multi-subgraph of G.

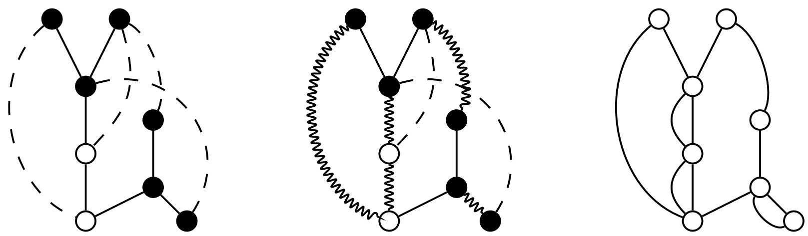
Lemma 4.1.1. *Let $G = (V, E)$ be a connected graph, let $T = (V, F)$ be a spanning tree of G, and let $H' = (V, E')$ be the spanning multi-subgraph of G formed using the Christofides Method. Then, graph G' is Eulerian.*

Proof. Let K be the set of odd degree nodes in T, and let D be a minimum cost K-join in G. Then, by definition of E', it follows that for any $u \in V$,

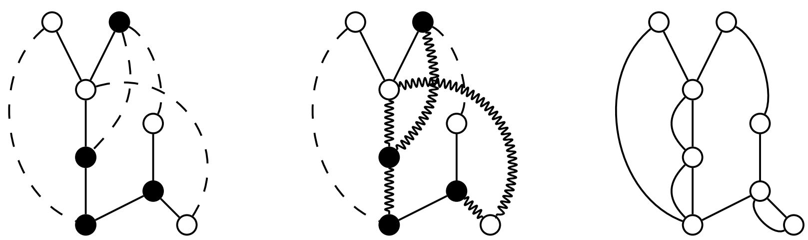
$$|\delta_{H'}(u)| = |\delta_T(u)| + |D \cap \delta_G(u)|.$$

As D is a K-join for G, it must be that both $|\delta_T(u)|$ and $|D \cap \delta_G(u)|$ have the same parity, and thus $|\delta_{H'}(u)|$ is even. Note that H' is connected, as it contains spanning tree T as a subgraph. $\square$

In fact, Mömke and Svensson [44] have also proposed a method to form a spanning



(a)



(b)

Figure 4.1.1: Example of applying (a) the Christofides Method and (b) the Mömke and Svensson Method to a given graph using a specified tree. The solid nodes represent the odd degree nodes in each respective join, and the undulated edges represent the edges of each join.

Eulerian multi-subgraph of G, which differs from Christofides in that not only may edges be added to G, but they may also be removed from it. Indeed, let M be a minimum cost odd join for G found using the costs:

$$c'_e = \begin{cases} c_e & \text{if } e \in F, \\ \\ -c_e & \text{if } e \notin F. \end{cases}$$

Then, the *Mömke and Svensson Method* for forming a spanning Eulerian multi-subgraph $H' = (V, E')$ of G consists of adding $M \cap F$ to G and removing $M \cap \bar{F}$ from it. This spanning Eulerian multi-subgraph has cost $c(E)+c'(M)$. Note that the modified edge costs in G reflect whether the edge is added to G, or removed from it. An example of this method applied to a particular graph is presented in Figure 4.1.1(b). Mömke and Svensson [44] show that the resulting graph is a spanning Eulerian multi-subgraph of G, however, we provide a proof for the purpose of completeness.

Lemma 4.1.2. *[44] Let $G = (V, E)$ be a connected graph, and let $T = (V, F)$ be a spanning tree of G and let M be an odd join for G. Then, the spanning multi-subgraph $H' = (V, E')$ obtained by setting*

$$E' = E \cup (M \cap F) - (M \setminus F),$$

is Eulerian.

Proof. We use a similar argument to that used by Yao [16]. By definition of E' we have that for any $u \in V$,

$$\begin{aligned} |\delta_{H'}(u)| &= |\delta_G(u)| + |\,(M \cap \delta_G(u)) \cap F| - |\,(M \cap \delta_G(u)) \setminus F| \\ &= |\delta_G(u)| + 2|(M \cap \delta_G(u)) \cap F| - |M \cap \delta_G(u)|. \end{aligned}$$

As M is an odd join for G, it must be that both $|\delta_G(u)|$ and $|M \cap \delta_G(u)|$ have the same parity, and therefore it follows that $|\delta_{H'}(u)|$ is even. Note that H' is connected, as edge set F forms a spanning tree of G, and by definition of E' none of these edges are removed from G. □

Figure 4.1.1 provides an example of both the Christofides Method and the Mömke and Svensson Method applied to the same spanning tree. Observe that the final spanning Eulerian multi-subgraphs in Figure 4.1.1 are identical. In fact we will show that both methods are equivalent, that is, any spanning Eulerian multi-subgraph obtained by applying the Christofides Method to a graph G and spanning tree T can be obtained by applying the Mömke and Svensson Method, and vice versa. Note that, to the best of our knowledge, this relationship has not previously been demonstrated.

Given a spanning tree $T = (V, F)$ for G, let K be the odd nodes in T. A key idea behind our argument relies on showing that for any odd join for G, we can find a corresponding K-join for G, and vice versa, such that both intersect the same tree edges, but don't intersect the same non-tree edges.

Theorem 4.1.3. *Let $G = (V, E)$ be a connected graph, and let $T = (V, F)$ be a spanning tree of G with odd degree nodes K. If D is a K-join for G, then $M^* = (D \cap F) \cup (\bar{F} \setminus D)$ is an odd join for G. Furthermore, if M is an odd join for G then $D^* = (M \cap F) \cup (\bar{F} \setminus M)$ is a K-join for G.*

Proof. We prove the first assertion. Observe that by definition of M, we have that for any node $u \in V$,

$$
\begin{aligned}
&|M^* \cap \delta_G(u)| \\
&= |D \cap F \cap \delta_G(u)| + |(\bar{F} \setminus D) \cap \delta_G(u)| \\
&= |D \cap F \cap \delta_G(u)| + |D \cap \bar{F} \cap \delta_G(u)| - |D \cap \bar{F} \cap \delta_G(u)| + |(\bar{F} \setminus D) \cap \delta_G(u)| \\
&= |D \cap \delta_G(u)| - |D \cap \bar{F} \cap \delta_G(u)| + |(\bar{F} \setminus D) \cap \delta_G(u)| \\
&= |D \cap \delta_G(u)| + |\bar{F} \cap \delta_G(u)| - 2|D \cap \bar{F} \cap \delta_G(u)| \\
&= |D \cap \delta_G(u)| + |\bar{F} \cap \delta_G(u)| + |F \cap \delta_G(u)| - |F \cap \delta_G(u)| - 2|D \cap \bar{F} \cap \delta_G(u)| \\
&= |D \cap \delta_G(u)| + |\delta_G(u)| - |\delta_T(u)| - 2|D \cap \bar{F} \cap \delta_G(u)|.
\end{aligned}
$$

Now, note that $|D \cap \delta_G(u)|$ and $|\delta_T(u)|$ are of the same parity since D is a K-join for G. Therefore, it follows from the above equation that $|M^* \cap \delta_G(u)|$ has the same parity as $|\delta_G(u)|$ and thus M is an odd join for G.

Using similar reasoning, we prove the second assertion. Observe that by definition of D, we have that:

$$
\begin{aligned}
&|D^* \cap \delta_G(u)| \\
&= |M \cap F \cap \delta_G(u)| + |(\bar{F} \setminus M) \cap \delta_G(u)| \\
&= |M \cap F \cap \delta_G(u)| + |M \cap \bar{F} \cap \delta_G(u)| - |M \cap \bar{F} \cap \delta_G(u)| + |(\bar{F} \setminus M) \cap \delta_G(u)| \\
&= |M \cap \delta_G(u)| - |M \cap \bar{F} \cap \delta_G(u)| + |(\bar{F} \setminus M) \cap \delta_G(u)| \\
&= |M \cap \delta_G(u)| + |\bar{F} \cap \delta_G(u)| - 2|M \cap \bar{F} \cap \delta_G(u)| \\
&= |M \cap \delta_G(u)| + |\bar{F} \cap \delta_G(u)| + |F \cap \delta_G(u)| - |F \cap \delta_G(u)| - 2|M \cap \bar{F} \cap \delta_G(u)| \\
&= |M \cap \delta_G(u)| + |\delta_G(u)| - |\delta_T(u)| - 2|M \cap \bar{F} \cap \delta_G(u)|.
\end{aligned}
$$

Now, note that $|M \cap \delta_G(u)|$ and $|\delta_G(u)|$ are of the same parity since M is an join for G. Therefore, it follows from the above equation that $|D^* \cap \delta_G(u)|$ has the same parity as $|\delta_T(u)|$ and thus D^* is a K-odd join for G. □

The transformation from one join to another can be seen in Figure 4.1.1. We are now ready to show that both methods are equivalent, that is, the costs of the resulting spanning Eulerian multi-subgraphs are identical.

Theorem 4.1.4. *Let $G = (V, E)$ be a graph with edge costs $c \in \mathbf{R}^E$, and let $T = (V, F)$ be a spanning tree of G. Then, the spanning Eulerian multi-subgraph formed by the Christofides Method and the Mömke and Svensson Method have the same cost.*

Proof. Let D be the minimum cost K-join found for G using the Christofides Method, where K is the set of odd degree nodes in T. Then, the cost of the spanning Eulerian multi-subgraph $H' = (V, E')$ obtained by applying the Christofides Method is $c(F) +$

$c(D)$. Define $M = (F \cap D) \cup (\bar{F} \setminus D)$. Then it follows from Theorem 4.1.3 that M is an odd join for G. If we add the edges of $F \cap M$ to G and remove those of $\bar{F} \cap M$, as done in the Mömke and Svensson Method, we obtain a spanning Eulerian multi-subgraph of cost:

$$\begin{aligned}
c(E) + c'(M) &= c(F) + c(\bar{F}) + c(M \cap F) - c(M \cap \bar{F}) \\
&= c(F) + c(\bar{F}) + c(D \cap F) - c(\bar{F} \setminus D) \\
&= c(F) + c(D \cap F) + c(\bar{F} \cap D) \\
&= c(F) + c(D),
\end{aligned}$$

which is the same as the cost of H'.

Similarly, let M be the minimum cost odd join for G found by the Mömke and Svensson Method using edge costs c'. The cost of the spanning Eulerian multi-subgraph $H' = (V, E')$ obtained by applying the Mömke and Svensson Method has cost $c(E) + c'(M)$. Define $D = (F \cap M) \cup (\bar{F} \setminus M)$. Then it follows from Theorem 4.1.3 that D is a K-join for G, where K is the set of odd degree nodes in T. If we add the edges of D to F, as done in the Christofides Method, we obtain a spanning Eulerian multi-subgraph of cost:

$$\begin{aligned}
c(F) + c(D) &= c(F) + c(D \cap F) + c(D \cap \bar{F}) \\
&= c(F) + c(\bar{F}) - c(\bar{F}) + c(M \cap F) + c(\bar{F} \setminus M) \\
&= c(E) + c(M \cap F) - c(M \cap \bar{F}) \\
&= c(E) + c'(M),
\end{aligned}$$

which is the same as the cost of H'. □

Recall that the Christofides algorithm, which makes use of the Christofides Method to form a spanning Eulerian multi-subgraph of G, yields a $\frac{3}{2}$-approximation algorithm for metric TSP. Therefore, in light of Theorem 4.1.4, it must be that applying the Mömke and Svensson Method to a minimum cost spanning tree T^* of G yields a $\frac{3}{2}$-approximation algorithm for metric TSP.

Theorem 4.1.5. *Given the complete graph $G = (V, E)$ with metric edge costs $c \in \mathbf{R}^E$, let $G_{x^*} = (V, E^*)$ be the support graph of the corresponding optimal subtour LP solution x^*, and let $T^* = (V, F^*)$ be a minimum cost spanning tree of G_{x^*}. Then, the Mömke and Svensson Method to form a spanning Eulerian multi-subgraph of G_{x^*} using T^* yields a $\frac{3}{2}$-approximation for metric TSP.*

Proof. Recall from Theorem 2.2.4 that $\left(\frac{n-1}{n}\right) x^*$ is in the spanning tree polytope (2.2.1). Therefore, $\left(\frac{n-1}{n}\right) x^*$ can we written as a convex combination of incidence vectors of spanning trees of G_{x^*}. Then, by Theorem 2.2.1, the cost of a minimum cost spanning tree $T^* = (V, F^*)$ is such that

$$c(F^*) \leq \left(\frac{n-1}{n}\right) cx^* < cx^*.$$

Define the vector $x' \in \mathbf{R}^E$ as follows:

$$x'_e = \begin{cases} \dfrac{x^*_e}{2} & \text{if } e \in F^*, \\ 1 - \dfrac{x^*_e}{2} & \text{if } e \notin F^*, \end{cases}$$

for all edges $e \in E$. It follows from Corollary 3.3.6, that x' is in the odd join polytope (3.2.1) for G_{x^*}, and therefore x' can be written as a convex combination of incidence vectors of odd joins. Using the edge costs c' for the odd join in the Mömke and

Svensson Method, it follows from Theorem 2.2.1 that the cost of a minimum cost odd join M^* for G_{x^*} is as follows:

$$c'(M^*) \leq c\left(\frac{x^*}{2}\right)(F^*) - c\left(\mathbf{1} - \frac{x^*}{2}\right)(\bar{F}^*).$$

Applying the Mömke and Svensson Method to both G_{x^*} and M^*, we obtain a spanning Eulerian multi-subgraph of G of cost $c(E) + c'(M^*)$. Therefore, by short-cuting this one, we obtain a TSP tour Q of cost:

$$\begin{aligned} c(Q) &\leq c(E) + c\left(\frac{x^*}{2}\right)(F^*) - c\left(\mathbf{1} - \frac{x^*}{2}\right)(\bar{F}^*) \\ &= c(F^*) + c(\bar{F}^*) + c\left(\frac{x^*}{2}\right)(F^*) - c(\bar{F}^*) + c\left(\frac{x^*}{2}\right)(\bar{F}^*) \\ &= c(F^*) + c\left(\frac{x^*}{2}\right) \\ &< \frac{3}{2}cx^*. \end{aligned}$$

As cx^* is a lower bound for the optimal TSP solution, the result follows. □

4.2 Generating a family of spanning trees through depth-first search

Let $G_{x^*} = (V, E)$ be the support graph of some point x^* in the subtour polytope (2.2.2). In this section, we discuss the depth-first search algorithm for traversing graphs, and discuss how we can use this algorithm to obtain a special spanning tree of G_{x^*}. Moreover, we show how this particular tree can be used to retrieve and represent an exponential sized collection of spanning trees of G_{x^*}. Furthermore, we show that such a collection of trees can be exponential in size.

4.2.1 Depth-first search traversal

Let graph $G = (V, E)$ be a connected graph. *Depth-first search*, henceforth DFS, is a means through which we can traverse the edges of G such as to visit all of its nodes in $O(n + m)$ time [21]. Starting at a node $r \in V$, the algorithm visits an adjacent node u which has not yet been visited in the traversal. The algorithm repeats this same procedure at node u, and all subsequent nodes, until it reaches a node where all of its neighbours have already been visited. The algorithm then backtracks to the most recently visited node that still has an unvisited neighbour, and continues the traversal. The algorithm terminates once all the nodes have been visited. Let F be the set of edges traversed in the depth-first search traversal of G. We say that F is the edge set of a particular spanning tree of G known as a *depth-first search tree*, or DFS tree.

Let $T = (V, F)$ be a depth-first search tree of G rooted at node $r \in V$, known as the *root* node. We say that the edges in T are *tree edges* and that the edges in G but not in T are *back edges*. The latter set of edges we denote by $B(T)$. Let $\overrightarrow{T}$ be an orientation of T in which the tree edges are directed away from the the root and the back edges are directed towards the root, see Figures 4.2.1(b) and 4.2.1(c) in which the solid edges represent the tree edges and the dashed edges represent the back edges.. Let $T(u)$ denote the set of nodes in the subtree rooted at u in $\overrightarrow{T}$, which includes u. Then, for any node $v \in V \setminus \{u\}$, if $v \in T(u)$ we say that u is an *ancestor* of v in $\overrightarrow{T}$ and that v is a *descendant* of u in $\overrightarrow{T}$. For example, in Figure 4.2.1(b), node f is an ancestor of e and node e is a descendant of f.

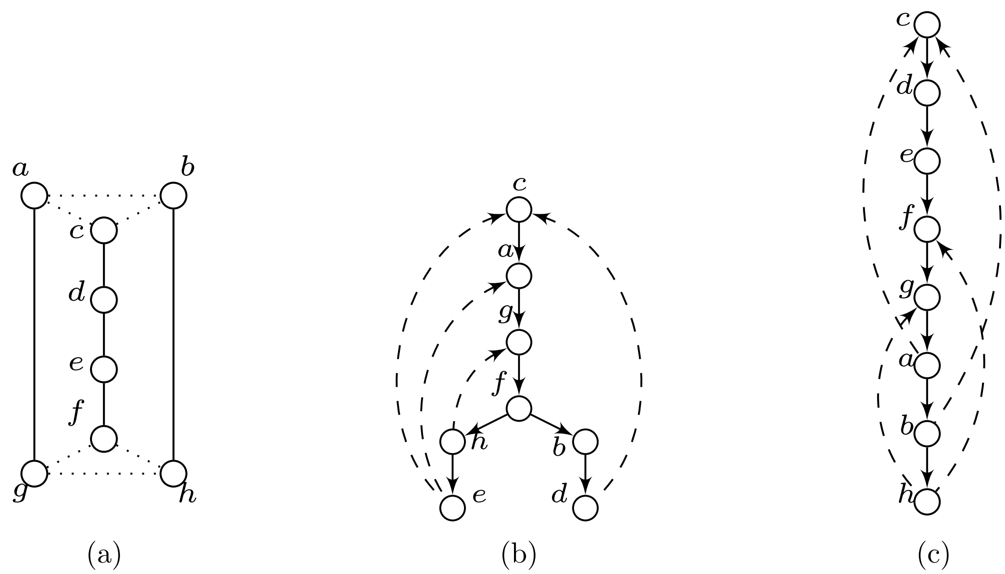


Figure 4.2.1: Examples of depth-first search trees of the support graph of a point x^* feasible for the subtour polytope (2.2.2). (a) The support graph G_{x^*}, where the solid edges are those with x^*-value one, and the dotted edges are those with x^*-value one-half. (b) A DFS tree of G_{x^*} rooted at node c. (c) A greedy DFS tree of G_{x^*} rooted at node c.

4.2.2 Greedy depth-first search

Throughout the remainder of this chapter, we assume that any vector x^* in the subtour polytope (2.2.2) is not a TSP tour. Thus, its support graph G_{x^*} will always have a node of degree at least three.

We begin by stating the following definition, which stems from the approach of Mömke and Svensson [44], and follows closely that of the definition given by Newman [34]:

Definition 4.2.1. *A* greedy depth-first search tree *of G_{x^*} rooted at node r of degree at least three, is one formed via depth-first search such that the highest x^*-value are always traversed first in the traversal.*

An example of a greedy DFS tree is presented in Figure 4.2.1(c). Following a similar approach to that of Newman [34], we show that for any such greedy DFS tree of G_{x^*}, it must be that every edge with x^*-value one, or *1-edge*, is a tree edge.

Lemma 4.2.2. *Let $x^* \in \mathbf{R}^E$ be a vector in the subtour polytope (2.2.2) and let $G_{x^*} = (V, E)$ be the corresponding support graph. Then, for any greedy DFS tree T of G_{x^*} rooted at node r, the 1-edges are contained in T.*

Proof. Suppose the lemma is not true. Then, it must be that there is a back edge $b =< v, u >$ with $x^*_b = 1$ that was not picked for T when traversing node u in the greedy depth-first search traversal. Note that node u has degree at least three since it is not a leaf node, and since the root r has degree at least three. Consequently, the edge picked for T when node u was traversed must also be a 1-edge, otherwise edge b would have been traversed. Therefore, it must be that $x^*(\delta(u)) > 2$, thus violating the degree constraint (2.2.2)(i) for node u, that is $x^*(\delta(u)) = 2$. □

4.2.3 Swap sets

Let $T = (V, F)$ be a depth-first search tree of a connected graph $G = (V, E)$ rooted at node $r \in V$. Recall the orientation $\overrightarrow{T}$ of T in which the tree edges are oriented away from the root and the back edges, that is the edges in $B(T)$, are directed towards the root. Given any tree edge $t_{uv} =< u, v >$ in $\overrightarrow{T}$, we define the *swap set* of t_{uv}, denoted by $S(t_{uv})$, as the set consisting of all back edges incident with u whose tail is a descendant of v. Figure 4.2.2 provides an example of the swap sets for each tree edge in a particular DFS tree.

Observe that for any tree edge t of $\overrightarrow{T}$ with non-empty swap set $S(t)$, removing t from T, and adding a back edge from $S(t)$ forms a new spanning tree, as seen in

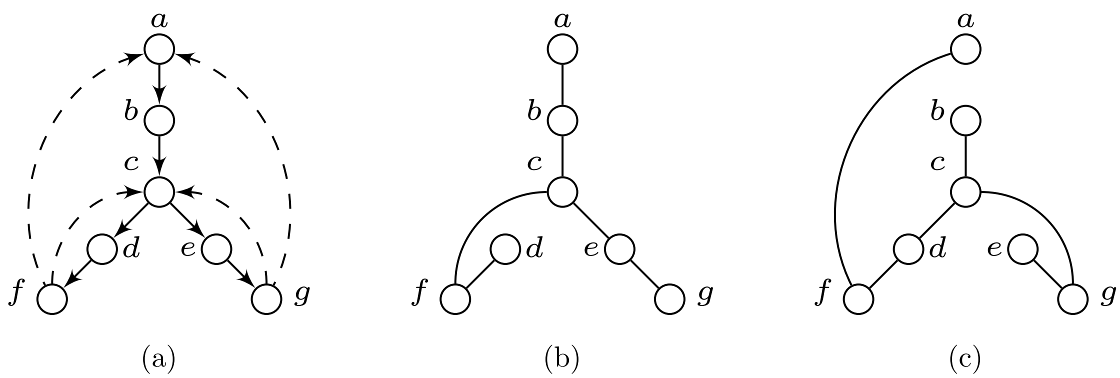


Figure 4.2.2: Example of swap sets and spanning trees, the latter obtained through edge swapping. (a) A DFS tree with swap sets $S(t_{ab}) = \{af, ag\}$, $S(t_{cd}) = \{cf\}$, $S(t_{ce}) = \{cg\}$ and $S(t) = \emptyset$ for all other tree edges t. (b) A spanning tree obtained by swapping tree edge $< c, d >$ with back edge $< f, c >$ in the original DFS tree. (c) A spanning tree obtained by swapping tree edges $< a, b >$ and $< c, e >$ with back edges $< f, a >$ and $< g, c >$ in the original DFS tree, respectively.

Figure 4.2.2. In fact, a stronger result, due to Mömke and Svensson [44], states that for any such number of swaps the resulting graph is a spanning tree, and therefore connected. Here, we include a proof due to Boyd *et al.* in an early version of [7], for completeness.

Theorem 4.2.3. *[44] Let $T = (V, F)$ be a depth-first search tree of some graph $G = (V, E)$, and let F_K be a subset of tree edges t of T with non-empty swap sets $S(t)$. Let T^* be the graph in which for each tree edge t in F_K, we have removed t from T and added a back edge from $S(t)$. Then, T^* is a spanning tree of G.*

Proof. As done by Boyd *et al.* in an early version of [7], we prove a stronger result, making the proof of this theorem easier:

Property P1: $T^*[T(u)]$ is a spanning tree of $T(u)$ for all nodes u in V, including the root node.

We proceed by induction on the number of swaps of edges in F_K. The property clearly holds for the initial tree T, as it is, by definition, a spanning tree. Now, suppose that Property P1 holds after having swapped a non-empty subset of tree edges $F_{K'} \subset F_K$, with a back edge in its swap set. Let $\hat{T}$ be the resulting spanning tree. Now, consider swapping tree edge $t =< u, v >$ in $F_K \setminus F_{K'}$ with back edge $b =< w, u >$ in $S(t)$. Recall that by definition of $S(t)$ it must be that $w \in T(v)$. By the induction hypothesis, we have that $\hat{T}[T(u)]$ and $\hat{T}[T(v)]$ are spanning trees of $T(u)$ and $T(v)$, respectively. Observe that $T(u) \setminus T(v)$ is connected, and may consist only of node u. Now, let T' be a spanning tree of $T(u) \setminus T(v)$, then we have that $\hat{T}[T(u)] = T' \cup \hat{T}[T(v)] \cup \{t_{uv}\}$. Remove edge t from $\hat{T}$. Then, $\hat{T}[T(u)]$ breaks into two connected components, $\hat{T}[T(v)]$ and T'. Add edge b. Then, it must be that b has one end in $\hat{T}[T(v)]$ and the other in T', namely nodes w and u, respectively. Therefore, Property P1 holds for node u after the edge swap, and holds for all other nodes in V. □

In light of Theorem 4.2.3, it follows that we can obtain a collection of spanning trees by swapping any number of tree edges with a corresponding back edge in their respective swap sets, provided this swap set is non-empty. Together, the DFS tree T and the swap sets implicitly represent a collection of spanning trees of G, which we denote by $\mathscr{T}_T$. Observe that there is at least one spanning tree in the collection, namely the depth-first search tree. However, there can be an exponential number of such spanning trees in the collection, even if we can assume that the number of edges is sparse, as is the case for the support graph of the subtour polytope (2.2.2) [8].

Theorem 4.2.4. *Let $G = (V, E)$ be a connected graph, let $T = (V, F)$ be a the depth-first tree of G, and let $\mathscr{T}_T$ represent the collection of spanning trees for T. Then, there may be an exponential number of spanning trees in $\mathscr{T}_T$.*

Proof. It may be that there are $O(n)$ tree edges with non-empty swap set. Therefore, for each tree edge with non-empty swap set, there are at least two possible choices, either it is kept in the tree or it is swapped with a back edge. As such, we may form $O(2^n)$ spanning trees. $\square$

4.3 Heuristic procedure

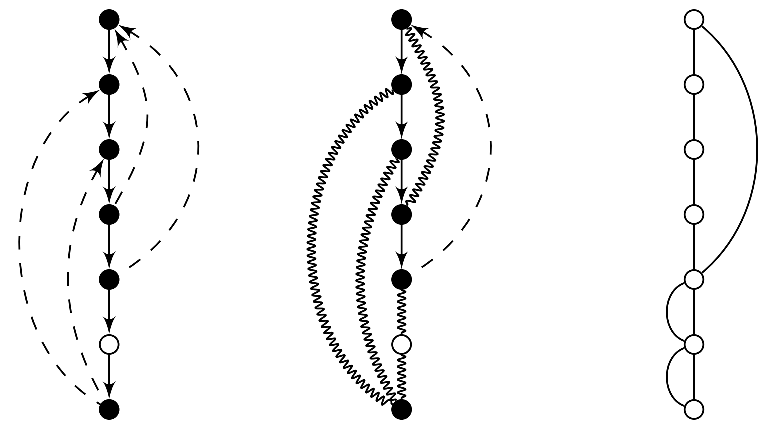
Let $K = (V, E)$ be the complete graph with metric edge costs $c \in \mathbf{R}^E$. We begin by finding the optimal subtour LP relaxation vector $x^* \in \mathbf{R}^E$ for K. Recall that we assume, without loss of generality, that x^* does not represent a TSP tour. We then find a greedy depth-first search tree T_{x^*} of the support graph G_{x^*} starting at root node r of degree at least three, following the heaviest x^*-values. Note that although T_{x^*} is undirected, we illustrate the tree, including the back edges, with its orientation in the figures for convenience for the reader, as it is easier to identify the swap sets. For every tree edge t in this depth-first search tree, we find its corresponding swap set $S(t)$. Recall that this greedy depth-first search tree, together with the swap sets, implicitly represent a collection of spanning trees $\mathscr{T}_{T_{x^*}}$ of G_{x^*}. In the footsteps of the Best-of-Many Christofides algorithm [20], we wish to find the cheapest TSP tour we can form by applying the Christofides Method, or equivalently the Mömke and Svensson Method, to all spanning trees in our collection $\mathscr{T}_{T_{x^*}}$. Unfortunately, by Theorem 4.2.4, there may be an exponential number of spanning trees in $\mathscr{T}_{T_{x^*}}$, and thus finding this TSP tour by applying the Mömke and Svensson Method to each individual tree is impractical. However, we have found a way to find a TSP tour by applying a restricted form of the Mömke and Svensson Method to this collection of trees, one which requires only a single application of the minimum cost $\hat{U}$-tight odd join algorithm on a slightly larger graph. As such, we do not examine each tree individually.

Given an odd join J for G_{x^*} and $K \subset E(G_{x^*})$, we say that J *saturates* K if $K \subseteq J$, and we say that an odd join J for G_{x^*} is a *restricted odd join* if J does not saturate any of the sets $(t \cup S(t))$, where t is a tree edge in T_{x^*} and $S(t) \neq \emptyset$. Given any spanning tree $T \in \mathscr{T}_{T_{x^*}}$, we now consider applying the Mömke and Svensson Method to T to obtain a spanning Eulerian multi-subgraph of G_{x^*}, but where we restrict ourselves to finding a restricted odd-join for G_{x^*}. More specifically, given a spanning tree $T = (V, F)$, the *restricted Mömke and Svensson Method* involves finding a restricted odd join for G_{x^*} using the costs c_e for edges in F and $-c_e$ otherwise, and forming a spanning Eulerian multi-subgraph by adding the edges of $F \cap J$ to G_{x^*} and removing the edges of $J \cap \bar{F}$. Note that this may not find the same solution that would have been obtained had we applied the Mömke and Svensson Method to every tree in $\mathscr{T}_{T_{x^*}}$ because of our use of a restricted odd-join. For example, Figure 4.3.1 presents such a scenario for a particular depth-first search tree. However, the trade-off gained here is that we do not have to apply our method to each tree individually.

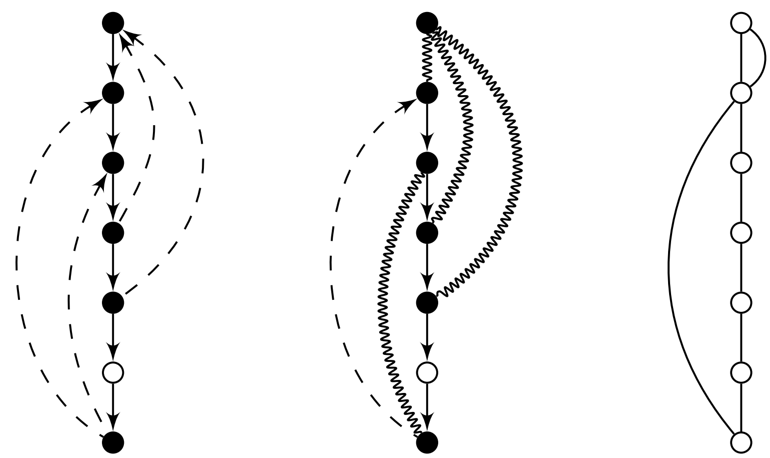
In the next section, we describe how this can be accomplished by only finding one restricted odd join for G_{x^*}. Then, in Section 4.3.2, we show that such a restricted odd join exists and how to find it by solving a $\hat{U}$-tight odd join problem on a slightly larger graph.

4.3.1 Using one particular restricted odd join on $\mathscr{T}_{T_{x^*}}$

In this section, we describe how our heuristic produces a TSP tour by finding just one restricted odd join. Given the support graph $G_{x^*} = (V, E)$ of a vertex x^* of the



(a)



(b)

Figure 4.3.1: Example illustrating a key difference of our heuristic using a restricted odd-join, for a depth-first search representation of a graph G, using the Mömke and Svensson method. The solid nodes represent the odd degree nodes G, and the undulated edges represent the edges of each odd join. All the edges shown have cost 1. (a) A restricted odd join of cost -1, and resulting spanning Eulerian multi-subgraph of cost 9. (b) A minimum cost odd join of cost -2, and resulting spanning Eulerian multi-subgraph of cost 8.

subtour polytope (2.2.2) and a greedy depth-first search tree $T_{x^*} = (V, F)$ of G_{x^*}, let F_{swap} be the set of tree edges in F with non-empty swap set $S(t)$, that is:

$$F_{\text{swap}} = \{t \in F : S(t) \neq \emptyset\}.$$

Recall by Theorem 4.2.3, that swapping an edge $t \in F_{\text{swap}}$ with an edge in $S(t)$ forms a new spanning tree. We now define a new set of costs c^* for $E(G_{x^*})$, where we take the usual costs used for the odd join in the Mömke and Svensson Method (i.e., c_e for edges in the tree, $-c_e$ otherwise), and also negate the costs of the edges in F_{swap}:

$$c^*_e = \begin{cases} c_e & \text{if } e \in F \setminus F_{\text{swap}}, \\ -c_e & \text{otherwise,} \end{cases}$$

for all edges $e \in E$. Our heuristic proceeds as follows. Let J be a restricted odd join for G_{x^*}. Let $H^* = (V, E^*)$ be the graph we obtain from G_{x^*} by adding the edges of $(F \setminus F_{\text{swap}}) \cap J$ to G_{x^*} and removing all other edges of J from G_{x^*}. Clearly:

$$c(H^*) = c(E) + c^*(J).$$

Below we show that the graph H^* found by our heuristic is a spanning Eulerian multi-subgraph of G_{x^*}.

Lemma 4.3.1. *The graph $H^* = (V, E^*)$ be the graph found by our heuristic. Then, H^* is a spanning Eulerian multi-subgraph of G_{x^*}.*

Proof. Let J be the restricted odd join for G_{x^*} found by our heuristic. In light of Theorem 4.2.3, it is possible to obtain a spanning tree $T^* = (V, F^*)$ in $\mathscr{T}_{T_{x^*}}$ of G_{x^*}, such that J only intersects F^* in edges that are in $F \setminus F_{\text{swap}}$. Indeed, T^* can be obtained by removing all the edges in $F_{\text{swap}} \cap J$ and replacing them with an edge from their respective swap set that is not in J. Note that we know such an edge exists as J is a restricted odd join for G_{x^*}. Now, obtain the graph H^* by applying the Mömke and Svensson Method to tree T^* i.e., add the edges of $F^* \cap J$ to G_{x^*} and remove the edges of $\bar{F}^* \cap J$. It follows from Lemma 4.1.2 that H^* is a spanning Eulerian multi-subgraph of G_{x^*}. □

4.3.2 Finding a restricted odd join for G_{x^*}

In order to find a restricted odd join for G_{x^*} for our heuristic, we transform the problem into a $\hat{U}$-tight odd join problem. We start by forming an auxiliary graph G'_{x^*} which consists of introducing a gadget in G_{x^*} for each tree edge in T_{x^*} with a non-empty swap set. Such gadgets will be used to ensure that the $\hat{U}$-tight odd join we find corresponds to a restricted odd join in G_{x^*}. Let $t =< u, v >$ be a tree edge in T_{x^*} with non-empty swap set $S(t)$. Then the gadgets added are as follows:

Case 1. Node u corresponds to root node r, that is $u = r$. Note that in T_{x^*} there is exactly one tree edge incident with r, namely t_{uv}, or equivalently t_{rv}. By assumption, there are at least two edges in $S(t_{rv})$, as r has degree at least three. We introduce a gadget consisting of a *gadget node* x as well as a *gadget edge* xr, in which exactly one back edge in $S(t_{rv})$ is incident to x and all remaining back edges are incident to node r.

Case 2. Node u is not the root node. We introduce a gadget consisting of gadget nodes x and y, as well as edge xv and gadget edges ux and xy, in which tree edge t_{uv}

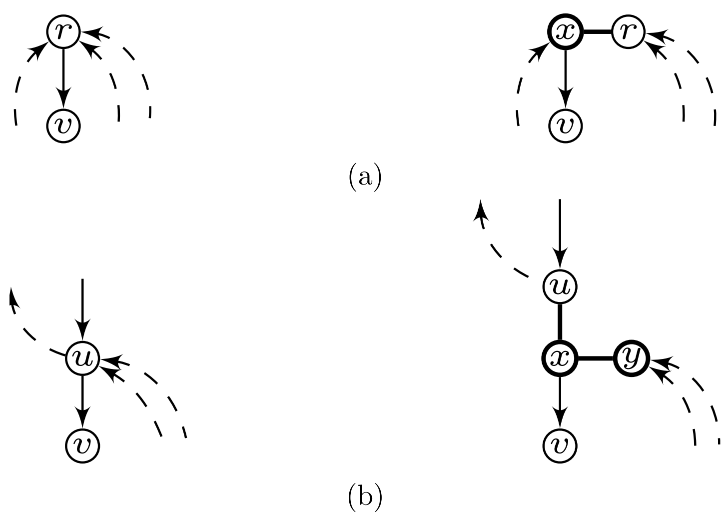


Figure 4.3.2: Gadgets introduced in G_{x^*} in order to form G'_{x^*}, where a solid edge is a tree edge, a dashed edge is a back edge, a thick edge is a gadget edge, and a thick node is a gadget node.

is mapped to edge xv and each back edge in $S(t_{uv})$ is incident to node y.

The gadgets added to G_{x^*} for Case 1 and Case 2 are shown in Figures 4.3.2(a) and 4.3.2(b), respectively. Note that any edge incident to node u in G_{x^*} unaffected by the introduction of the gadgets remains incident to node u in G'_{x^*}. We also note that each non-gadget edge in G'_{x^*} corresponds to either a tree edge in T_{x^*} or a back edge in $B(T_{x^*})$. As G_{x^*} is 2-node connected, it must be that G'_{x^*} is 2-edge connected, otherwise this would imply the existence of a cut node in G_{x^*}.

We complete the transformation of G'_{x^*} by taking the costs c^* from Section 4.3.1 for the edges of G_{x^*} and setting the cost of the gadget edges in G'_{x^*} to zero, as they will be ignored later and will not contribute to the cost of the final spanning Eulerian multi-subgraph. More specifically, we define the costs c'_e for each edge e in G'_{x^*} as follows:

$$c'_e = \begin{cases} c_e & \text{if } e \in F \setminus F_{\text{swap}}, \\ 0 & \text{if } e \text{ is a gadget edge,} \\ -c_e & \text{otherwise.} \end{cases}$$

We now wish to use G'_{x^*} in order to find a restricted odd join for G_{x^*} that our heuristic will use to form a spanning Eulerian multi-subgraph of G_{x^*}. We show that to obtain such a restricted odd join, we simply need to find a $\hat{U}$-tight odd join in G'_{x^*} which is restricted i.e., one that does not intersect all edges in $(t \cup S(t))$ for each tree edge t with non-empty swap set $S(t)$.

Theorem 4.3.2. *Let $\hat{U}$ be the set of gadget nodes labelled x in G'_{x^*}, then there exists a $\hat{U}$-tight odd join for G'_{x^*}.*

Proof. Given G'_{x^*} is 2-edge connected and that each node $\hat{U}$ has degree three, it follows from Theorem 3.3.1 that the one-third vector is feasible for the $\hat{U}$-tight odd join polytope (3.3.1). □

We now show that such a $\hat{U}$-tight odd join for G'_{x^*} is such that it does not intersect all edges in $t \cup S(t)$ for each tree edge t with non-empty swap set $S(t)$.

Theorem 4.3.3. *Let $\hat{U}$ be the set of gadget nodes labelled x in G'_{x^*}, and let J be any $\hat{U}$-tight odd for G'_{x^*}. Then, for any tree edge t with non-empty swap set $S(t)$, there is at least one edge in J which is not in $(t \cup S(t))$.*

Proof. Consider the gadget node x adjacent to the root node. If J intersects tree edge $t =< x, v >$, then there is at least one back edge which does not intersect with J, namely the back edge incident with gadget node x. If t does not intersect J, then the properly trivially holds. Now, consider any other tree edge $t = < x, v >$ with non-empty swap $S(t)$. By nature of the gadgets introduced, we know that node x is in $\hat{U}$ and has degree three. If t is not in J then the property holds. Assume otherwise, that t is in J. Given x is in $\hat{U}$, there are no other edges incident to x that intersect with J, in particular edge xy, where y is the node into which we have directed all edges in $S(t)$. Since J is an odd join for G'_{x^*}, we know that $|\delta_{G'}(y)|$ and $|J \cap \delta_{G'}(y)|$ have the same parity. Thus, if $xy \notin J$, there must be at least one other edge incident to y not in J, namely one in $S(t)$. □

Let E_g represent the set of gadget edges in G'_{x^*} and let J be the $\hat{U}$-tight odd join of G'_{x^*} produced by our heuristic. We now show that $J \setminus E_g$ is a restricted odd join for G_{x^*}.

Theorem 4.3.4. *Let J be a $\hat{U}$-tight odd join for G'_{x^*} and let E_g represent the set of gadget edges in G'_{x^*}. Then, $J' = J \setminus E_g$ is an odd join for G_{x^*}.*

Proof. Let u be a node in G_{x^*}, and let V' be the set of gadget nodes, in addition to node u, which are introduced in G_{x^*} for each tree edge with non-empty swap set incident with u. Then, in graph G'_{x^*}, we have that:

$$|\delta(V')| = \sum_{v \in V'} |\delta(v)| - 2|\gamma(V')|.$$

Similarly, we have that:

$$|\delta(V') \cap J| = \sum_{v \in V'} |\delta(v) \cap J| - 2|\gamma(V') \cap J|.$$

Now, note from the above that both $|\delta(V')|$ and $\sum_{v \in V'} |\delta(v)|$ have the same parity, and that both $|\delta(V') \cap J|$ and $\sum_{v \in V'} |\delta(v) \cap J|$ have the same parity. Then, as J is a $\hat{U}$-tight odd join for G'_{x^*}, it follows that both $\sum_{v \in V'} |\delta(v)|$ and $\sum_{v \in V'} |\delta(v) \cap J|$ have the same parity, and thus, it follows that $|\delta(V')|$ and $|\delta(V') \cap J|$ have the same parity. Thus, if we shrink V' back to its original node u, it must be that $|\delta_{G_{x^*}}(u)|$ and $|\delta_{G_{x^*}}(u) \cap (J \setminus E_g)|$ have the same parity. Consequently, $J \setminus E_g$ is an odd join for G_{x^*}. $\square$

In light of Theorem 4.3.3 and Theorem 4.3.4, we obtain the following corollary.

Corollary 4.3.5. *Let J be a $\hat{U}$-tight odd join for G'_{x^*} such that $\hat{U}$ is the set of gadget nodes labelled x in G'_{x^*}, and and let E_g represent the set of gadget edges in G'_{x^*}. Then, $J' = J \setminus E_g$ is a restricted odd join for G_{x^*}.* $\square$

4.3.3 Modified procedure

In this section, we describe a modification to our heuristic in which we are able to reduce the size of the initial support graph. Let $G_{x^*} = (V, E)$ be the support graph of a vertex x^* of the subtour polytope (2.2.2), and let $T_{x^*} = (V, F)$ be a greedy depth-first search tree of G_{x^*} rooted at node r of degree at least three. We say that a path P in G_{x^*} is a *1-path* if it consists exclusively of 1-edges, that is $x^*_e = 1$ for all edges e in P, and it is not properly contained in another such path. We observed that many optimal subtour LP solutions x^* have support graphs with long 1-paths, thus by handling these as single edges in our heuristic we could greatly improve the run time.

Recall that we define the set of tree edges in F with non-empty sets $S(t)$ by F_{swap} and that by Theorem 4.2.2 all 1-edges are in T_{x^*}. Let t be a 1-edge in a 1-path P of G_{x^*} such that $t \in F_{\text{swap}}$. Note that in the directed form $\overrightarrow{T_{x^*}}$ of our greedy DFS tree,

t is the first edge of P, since we travel from the root and $P \subset F$. Then, removing t from G_{x^*} and adding edge $b \in S(t)$ forms a spanning tree. In fact, let t^* be the most expensive edge in P, then removing t^* from G_{x^*}, rather than t, and adding $b \in S(t)$ also forms a spanning tree.

In an effort to reduce the size of G_{x^*}, we use the above idea and propose a modification to our heuristic which consists of replacing each 1-path P of G_{x^*} with a single gadget edge of a particular cost. We denote the resulting graph by $G^P = (V', E')$. Note that G^P will not have any nodes of degree-two as such nodes are the ends of 1-edges in a 1-path. Now, define vector $y^* \in \mathbf{R}^{E'}$ as follows:

$$y_e^* = \begin{cases} 1 & \text{if } e \text{ is an edge replacing a 1-path} \\ x_e^* & \text{if } e \text{ correspond to an edge in } E \end{cases} \tag{4.3.1}$$

for all edges e in E'. We show that y^* is feasible for the subtour polytope (2.2.2) for G^P.

Lemma 4.3.6. *Let $G_{x^*} = (V, E)$ be the support graph of a vector $x^* \in \mathbf{R}^E$ in the subtour polytope (2.2.2). Consider the graph $G^P = (V', E')$ in which the 1-paths of G_{x^*} have each been replaced with a single gadget edge of cost equal that of the sum of the edges in their corresponding 1-paths, that is $c(P)$. Let vector $y^* \in \mathbf{R}^{E'}$ be as defined in (4.3.1). Then, y^* is a feasible vector for the subtour polytope (2.2.2) for G^P.*

Proof. It is clear from the 1-path reduction that for any node $u \in V'$, $y^*(\delta(u)) = 2$. Let $U \subset V$, and consider $U' \subseteq U$ which represents the corresponding subset of nodes in the shrunk graph $G^P_{x^*}$. Consider cuts $\delta(U)$ and $\delta(U')$. By nature of the 1-path reduction, it must be that $|\delta(U')| = |\delta(U)|$. Now, by definition of y^*, each edge in

$\delta(U')$ corresponds to an edge in $\delta(U)$ with the same x^*-value. Therefore, it must be that $y^*(\delta(U')) = x^*(\delta(U)) \geq 2$. Finally, it follows by definition of y^* that $0 \leq y^*_e \leq 1$ for all edges e in E'. □

Let vector y^* be as defined in (4.3.1) and let G_{y^*} be its corresponding support graph, the latter obtained by replacing each 1-path in G_{x^*} with a gadget edge. We now show that no parallel edges are introduced in G_{y^*}.

Lemma 4.3.7. *Support graph G_{y^*} is simple.*

Proof. Assume towards a contradiction that G_{y^*} contains parallel edges. As there are no parallel edges in G_{x^*}, it must be that parallel edges in G_{y^*} were introduced after substituting a 1-path P in G_{x^*} with an edge $e_P = uv$. In fact, e_P must be one of these parallel edges. Consider the set $U = \{u, v\}$ in G_{y^*}, then $y^*(\gamma(U)) > 1$ as $y^*_{e_P} = 1$. Thus:

$$y^*(\delta(U)) = y^*(\delta(u)) + y^*(\delta(v)) - 2y^*(\gamma(U)) < 2,$$

contradicting the feasibility of y^* for the subtour polytope (2.2.2). □

We now follow the steps of the regular procedure and find a greedy depth-first-search tree $T_{y^*} = (V', F')$ of G_{y^*}. Note that each edge e_P will be a tree edge in T_{y^*} as they correspond to 1-edges in G_{y^*}. Let F'_{swap} denote the set of tree edges in T_{y^*} with non-empty swap set $S(t)$.

Let E_1 be the set of replacement 1-path edges in G_{y^*}. Consider edge $e_P \in E_1$, and let P be its corresponding 1-path in G_{x^*}.

If $e_P \notin F'_{\text{swap}}$ and it intersects the restricted odd join for G_{y^*} found by our heuristic, then we will add the edges of path P in G_{x^*}, at a cost of $c(P)$. If $e_P \in F'_{\text{swap}}$ and it intersects the restricted odd join, we will remove the most expensive edge in P from G_{x^*} and add all other edges in P. We reflect this by using the following costs for edges $e_P \in E_1$ when we apply our heuristic to G_{y^*}: For $e_P \in E_1 \setminus F'_{\text{swap}}$, use $c(P)$, and for edges $e_P \in E_1 \cap F'_{\text{swap}}$ use $c(P) - 2c_{e^*_P}$, where e^*_P is the most expensive edge in P.

Finally, let J be a restricted odd join for G_{y^*}. Note that we can extend J to a restricted odd join J' for G_{x^*} by replacing each 1-edge in J with the 1-edges in their respective 1-path in G_{x^*}. Once the heuristic obtains the proper spanning tree, it makes use of J' to form the final spanning Eulerian multi-subgraph. Recall that if J' intersects with the edges of a 1-path P, we remove the most expensive edge from P and add the remaining edges to G_{x^*}. The final TSP tour is found by shortcutting an Euler tour of this spanning Eulerian multi-subgraph.

4.3.4 Avoiding unnecessary gadgets

A second modification we propose to our heuristic concerns the introduction of gadgets to G_{x^*} only when it is necessary. In doing so, we avoid introducing degree-two nodes in G'_{x^*}. Indeed, let $t_{uv} =< u, v >$ be a tree edge in T_{x^*} with non-empty swap set $S(t_{uv})$, the modifications we propose are as follows:

Case 1: If u has degree three in G_{y^*}, then we do not add any gadget node or edges.

Case 2: If $u = r$, then we follow the same procedure described in Case 1 detailed Subsection 4.3.2. We introduce a gadget consisting of gadget node x as well as edge xv and gadget edge xr, in which all but one edge in $S(t_{rv})$ is incident to node x, and all other edges in $S(t_{rv})$ are incident to r. Recall that in this scenario, tree edge t_{uv}, or rather t_{rv}, is mapped to edge xv.

Case 3: If $u \neq r$ and $|S(t_{uv})| = 1$, then we introduce a gadget consisting of gadget node x as well as edge xv and gadget edge ux, in which the single edge in $S(t_{uv})$ is incident to x. In this case, tree edge t_{uv} is mapped to edge xv.

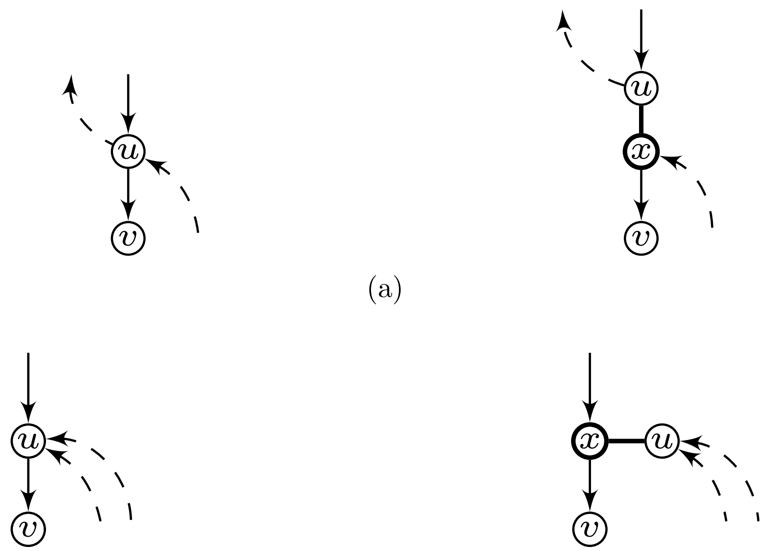
Case 4: If $u \neq r$, $|S(t_{uv})| \geq 2$ and there are no other edges incident to u. Then, we introduce a gadget consisting of gadget node x, as well as edge xv and gadget edge ux, in which the edges of $S(t_{uv})$ are incident to u. Here, tree edge t_{uv} is mapped to edge xv.

Case 5: If $u \neq r$ and $|S(t_{uv})| \geq 2$, then as done in Section 4.3.2, we introduce a gadget consisting of gadget nodes x and y, as well as an edge xv and gadget edges ux, and xy, in which the edges of $S(t_{uv})$ are incident to y. Recall that in this scenario, tree edge t_{uv} is mapped to edge xv.

The gadgets introduced in Case 2 and Case 5 are shown in Figures 4.3.2(a) and 4.3.2(b) respectively. Additionally, The gadgets introduced in Case 3 and Case 4 are shown in Figures 4.3.3(a) and 4.3.3(b), respectively. As noted in Subsection 4.3.2, any edge incident to node u unaffected by the introduction of the gadgets to G_{x^*} remain incident to node u in G'_{x^*}.

Theorem 4.3.8. *Auxiliary graph G'_{x^*} has $O(n)$ nodes and $O(n)$ edges, in the worst case.*

Proof. Observe that in the worst case we introduce a gadget in G_{x^*} which consists



(a)

(b)

Figure 4.3.3: Additional gadgets introduced in G_{x^*} in order to form G'_{x^*}, where a solid edge is a tree edge, a dashed edge is a back edge, a thick edge is a gadget edge, and a thick node is a gadget node.

of two gadget nodes and two gadget edges, for every tree edge in T_{x^*}. Thus, we have that

$$|V(G'_{x^*})| \leq |V(G_{x^*})| + 2|F| \leq n + 2(n-1) = 3n - 2,$$

and, from the sparsity of G_{x^*}, we have that

$$|E(G'_{x^*})| \leq |E(G_{x^*})| + 2|F| \leq (2n-3) + 2(n-1) = 4n - 5.$$

Therefore, there are $O(n)$ nodes and $O(n)$ edges in G'_{x^*}. $\square$

Let $\hat{U}$ denote the set of gadget nodes labelled x in G'_{x^*}. Then, it follows from Theorem 3.3.7 and Theorem 4.3.3, that we can find a minimum cost $\hat{U}$-tight odd join J^* for G'_{x^*} by solving the minimum cost perfect matching problem on a slightly larger graph, G''_{x^*}. This auxiliary graph consists of expanding each node of G'_{x^*} that does not belong to the set $\hat{U}$ into a clique of size equal to its degree in G'_{x^*}. Given Theorem 3.2.3 and the sparsity of G_{x^*}, that is $|E(G_{x^*})| \leq 2n - 3$ [8], we obtain the following result.

Theorem 4.3.9. *In the worst case, auxiliary graph G''_{x^*} has $O(n)$ nodes and $O(n^2)$ edges.* □

4.3.5 Final remarks

In this section, we have described a restricted form of the Mömke and Svensson Method to find a TSP tour using a single application of the minimum cost $\hat{U}$-tight odd join algorithm. Our ideas are extensions of those used by Mömke and Svensson [44] for approximating graph-TSP. Indeed, we extend their idea of a swap pair to swap sets where we allow each tree edge to have more than one possible partner back edge with which we can swap to form a spanning tree. Using our definition of a swap sets, we avoid having to solve a minimum cost circulation problem on the graph to obtain said swap pairs, as done by Mömke and Svensson. Rather than solving the minimum cost problem on a cubic graph, we solve it on an auxiliary graph in which degree-two nodes are expanded into diamonds and all other nodes are expanded into a clique of size equal its degree. Moreover, we extend the general concept of these ideas to metric costs.

4.4 A description of an experiment to test our heuristic

The heuristic was implemented using the C programming language. The experiment was performed on a 2.4 GHz Intel Core i5 Macbook Pro with 4GB 1333 MhZ DDR3 memory. We make use of the Concorde TSP solver [1] to compute the subtour LP solution and use the BlossomV [28] computer code to compute in $O(n^2m)$ time a minimum cost perfect matching necessary to find a minimum cost $\hat{U}$-tight odd join in our auxiliary graph G'_{x^*}. Our test data was chosen to match those used by Genova and Williamson in their experimental evaluation of the Best-of-Many Christofides' algorithm [19], as we wish to compare their results with our own. More specifically, we considered 64 symmetric TSPLIB instances of Reinelt [37], 59 of which are two-dimensional Euclidean instances with up to 2103 nodes, averaging 524 nodes, and 5 of which are non-Euclidean instances with up to 1032 nodes, averaging 484 nodes. The test data also included 39 two-dimensional Euclidean VLSI instances of Rohe [38] with up to 3694 nodes, averaging 1473 nodes. Finally, the test data included 9 graph-TSP instances of the Koblenz Network Collection of Kunegis [29]. We note that each of these instances are undirected simple graphs, which are not necessarily connected. In the event of multiple connected components, we use the largest one, with respect to the number of nodes, and discard all other connected components. As such, the graph-TSP instances considered have up to 1615 nodes, and average 365 nodes, with respect to their respective largest connected component. Finally, note that we use Concorde to find the cost of an optimal TSP tour for these instances.

When we compare our results to those of Genova and Williamson [19], we focus our attention on the variants of this algorithm which explicitly find a convex combination of spanning trees of a given point in the subtour polytope (2.2.2), namely the column generation and splitting-off variants.

We perform ten iterations of our heuristic for each instance, and keep the cheapest solution found. We only call upon Concorde once to compute the optimal subtour LP solution of the given input graph. Furthermore, our heuristic uses a different root node to form a greedy DFS tree of the support graph at each iteration. We note that for graph-TSP instances, we follow Genova and Williamson [19] and first find a largest connected component. We then use the Floyd-Warshall algorithm [14] to solve in $O(n^3)$ time an all-pairs shortest path problem on the largest connected component to find its metric completion. We then use the resulting graph as input to our heuristic. With regards to shortcutting, we follow the method proposed by Genova and Williamson [19]; when we reach a node u previously visited in our traversal of the Euler tour, we determine whether it would be less expensive to skip over u at this point or to remove u from its previous location in the TSP tour and insert u at this current location.

4.5 Experimental results

In this section, we provide a summary of our experimental results for our proposed heuristic, and compare them to those obtained by Genova and Williamson [19] for the (standard) Christofides' algorithm, and the column generation and splitting off variants of the Best-of-Many Christofides' algorithm. For further details on the results of the experimental evaluation of the Best-of-Many Christofides' algorithm, including other variants than the ones considered here, we refer the reader to Genova and

Williamson [19].

Note that for all the problems tested by Genova and Williamson [19], the optimal TSP solution is known. When examining the quality of a particular TSP tour produced by our heuristic, we compare its cost with the cost of an optimal TSP tour, and report the percent error as defined by the following formula:

$$\text{percent error} = \frac{(\text{experimental value} - \text{cost of optimal TSP tour})}{\text{cost of optimal TSP tour}} \times 100\%.$$

In the context of our experiment, the percent error quantifies the difference between the cost of the TSP tour obtained by our heuristic and the cost of an optimal TSP tour.

We distinguish between four collections of instances, namely two-dimensional Euclidean TSPLIB instances, which we denote by TSPLIB (E), non-Eulclidean TSPLIB instances, which we denote by TSPLIB (N), as well as VLSI instances, which we denote by VLSI, and graph-TSP instances, which we denote by Graph. For each collection, we average the percentage error with respect to the cost of the cheapest TSP tour produced by our heuristic and the cost of an optimal TSP tour over all of its instances. We summarize these results in Table 4.5.1 (Note that a more detailed description of our results can be found in Appendix A). First, we note that on average, the cost of the solutions produced by our heuristic is 4.78% away from the cost of an optimal solution. Second, we see that the quality of our solutions are much better than the quality of those produced by the (standard) Christofides' algorithm. Finally, our results show that the collection of spanning trees generated by the greedy depth-first search tree, on average, allow for some very good TSP tours when compared to the column generation and splitting off variants of the Best-of-Many Christofides'

	CHR	Best-of-Many Christofides (Genova and Williamson) ColGen	Split	Our heuristic
TSPLIB (E)	9.56%	4.03%	5.23%	4.41%
VLSI	9.73%	7.00%	6.60%	6.56%
TSPLIB (N)	5.40%	2.73%	2.92%	2.40%
Graph	12.43%	0.57%	0.88%	0.76%
Over all instances	9.66%	4.93%	5.25%	4.78%

Table 4.5.1: Summary of the average percent error of the standard Christofides' algorithm, the column generation and the splitting off variants of the Best-of-Many Christofides' algorithm [19], as well as the percent error our heuristic, with respect to the cost of an optimal TSP solution.

algorithm. Indeed, this can be seen by computing the percent error between the cost the solution produced by our heuristic and the cost of the solution produced for the column generation variant, and similarly for the splitting off variant. Specifically, over all instances, the cost of the solution produced by our heuristic is on average 0.83% and 0.33% away from the cost of the solution produced by the column generation and splitting off variants, respectively.

We now turn our attention to running time. The average running times of our heuristic as well as the Christofides algorithm and the relevant variants of the Best-of-Many Christofides algorithm [19] are summarized in Table 4.5.2. Observe that the average running time of our heuristic, over ten iterations, is extremely fast compared to the running times of the column generation and splitting-off variants, which were not implemented to be as fast as possible [19]. Specifically, on average, our heuristic produces a solution in 0.1 seconds, compared to 0.2 seconds for the standard Christofides' algorithm, and 118.2 seconds for the column generation variant and 8,347.0 seconds for the splitting off variant. Indeed, having to explicitly find the convex combination of spanning trees, and apply Christofides algorithm' on each

		Best-of-Many Christofides (Genova and Williamson)		
	CHR	ColGen	Split	Our heuristic
TSPLIB (E)	0.1	82.2	3470.0	0.01
VLSI	0.2	208.7	18128.2	0.03
TSPLIB (N)	0.1	177.3	25.5	0.04
Graph	1.4	21.0	2556.0	0.6
Over all instances	0.2	118.2	8347.0	0.1

Table 4.5.2: Average running time (user time) in seconds for the (standard) Christofides' algorithm [19], and the column generation and splitting off variants of the Best-of-Many Christofides' algorithm, as well as our heuristic. The time required to compute the optimal subtour LP solution using Concorde is excluded from the running times.

spanning tree is quite time consuming. In fact, with regards to the column generation variant, it was reported by Genova and Williamson [19] it could take up to ten hours to find the convex combination of spanning using the linear programming solver Gurobi [35] for instances on 500 nodes. This is not an issue for our heuristic, as there is no need to individually test every tree in our collection. Indeed, we need only use a single application of the minimum cost perfect matching for our heuristic to produce a solution. Note that the running time of our heuristic is dominated by the time required to find the optimal subtour solution or the time to find a minimum cost perfect matching. The running time to find the optimal subtour solution using Concorde and the average elapsed time required to find the restricted odd join are summarized in Table 4.5.3.

We now take the time to investigate the size of the initial support graph compared to that of the auxiliary graphs, namely the support graph in which all 1-paths have been replaced by a single 1-edge and the graph on which we find a minimum cost perfect matching, the latter we refer as the *clique graph*. To do so, we compute

	Our heuristic		
	Subtour	Restricted odd join	Iterations (ten)
TSPLIB (E)	0.2003	0.0003	0.0120
VLSI	0.5445	0.0008	0.0351
TSPLIB (N)	0.6152	0.0002	0.0349
Graph	0.3583	0.0001	0.5410
Over all instances	0.3528	0.0004	0.0640

Table 4.5.3: Average elapsed time (real time) in seconds for one application of Concorde to compute the optimal subtour LP solution, and average elapsed time (user time) in seconds for ten iterations of the heuristic, which includes the application of BlossomV to compute a minimum cost perfect matching. Note that the average time for ten iterations does not include the time to compute the optimal subtour LP solution using Concorde.

the ratio between the number of nodes in the initial support graph and the number of nodes nodes in the auxiliary graph of interest. Similarly, we compute the ratio between the number of edges in the initial support graph and the number of edges in the auxiliary graph of interest. For each collection of instances, we compute these ratios over all its instances, and report the average. These results are summarized in Table 4.5.4 (Note that a more detailed description of our results can be found in Appendix B). On average, over all instances, there is a significant reduction in the size of the graph, with respect to the number of nodes and edges, when substituting each 1-path in the support graph with a single 1-edge. In fact, this seems to suggest that the support graph of the optimal subtour LP solution obtained by Concorde contains a considerable number of long 1-paths. Similarly, the number of nodes and edges that make up the auxiliary clique graph, on average, do not exceed the number of nodes and edges that make up the initial support graph. Indeed, over all instances, the number of edges in the support graph in which all 1-paths have replaced by a single edge is on average within 0.28 of the number of edges in the initial support graph on

	Support graph (no 1-paths)		Clique graph		Spanning Eulerian multi-subgraph	
	Nodes	Edges	Nodes	Edges	Single edges	Double edges
TSPLIB (E)	0.19	0.26	0.40	0.55	0.77	0.11
VLSI	0.23	0.31	0.49	0.67	0.78	0.13
TSPLIB (N)	0.24	0.33	0.56	0.73	0.81	0.08
Graph	0.15	0.20	0.34	0.45	0.67	0.04
Over all instances	0.20	0.28	0.44	0.59	0.77	0.11

Table 4.5.4: Average of the ratio between the number of nodes and edges of various graphs and the nodes and edges of the initial support graph, over ten iterations of the heuristic.

average. Additionally, over all instances, the number of edges in the auxiliary graph on which we solve the minimum cost perfect matching problem is on average within 0.59 of the number of edges in the initial support graph on average.

Encouraged by our fast running time, we decided to also test our heuristic on the largest symmetric TSPLIB instance, pla85900, which consists of 85,900 nodes. This problem is far beyond the scope of Genova and Williamson for whom the biggest problem had 3694 nodes. Over ten iterations of the heuristic, the cheapest TSP tour produced for this instance was 6.33% away from the cost of an optimal TSP tour. Additionally, we observed that it took Concorde 131.17 seconds to compute the optimal subtour LP solution, and 1.78 seconds to perform ten iterations of our heuristic, the latter exlucludes the time it took for Concorde to compute the optimal subtour LP solution. We also observed a significant reduction of the size of the initial support graph when substituting each of its 1-paths by a single edge. Indeed the number of edges in the support graph in which all 1-paths have replaced by a single edge is within 0.13 of the number of edges in the initial support graph. Furthermore, the graph on which we solve the minimum cost perfect matching problem is much smaller than the size of the initial support graph. Indeed, on average, the number of edges in the auxiliary graph on which we solve the minimum cost perfect matching problem is within 0.25 of the number of edges in the initial support graph, over ten iterations of the heuristic.

Finally, we make a small observation with regards to the structure of the spanning Eulerian multi-subgraphs produced by our heuristic. As seen in Table 4.5.4, over all instances, there are on average more single edges in the spanning Eulerian multi-subgraphs than there are double edges. This suggests that there are far more edges removed than extra copies added to the initial support graph.

Chapter 5

Integrality gap of graph-TSP: A computational experiment

There has been very little progress made in proving the integrality gap for SEP for metric TSP is less than $\frac{3}{2}$. However, for the special case of graph-TSP, it was shown by Sebő and Vygen [42] that the integrality gap for SEP is at most $\frac{7}{5}$. Still, we don't know the exact integrality gap for small values of n, and how it compares to metric TSP.

In this chapter, we investigate the exact integrality gap for small instances of graph-TSP, and compare our results to those reported by Benoit and Boyd [3] for small instance of metric TSP. In doing so, we set up a computational experiment in which we explicitly find both the optimal value of the ILP and the optimal value of the LP for each graph G on n nodes, and evaluate the corresponding ratio. We focus on the following three classes of graphs:

1. All general connected graphs with n nodes, where $6 \leq n \leq 10$.

2. All cubic connected graphs with n nodes, where $6 \leq n \leq 16$.

3. All subcubic connected graphs with n nodes, where $6 \leq n \leq 16$.

In Section 5.1, we discuss the methods through which we acquire our data and outline the procedure used to evaluate the integrality gap. In Section 5.2, we present the results obtained from our experiment for each class of graphs, discuss their significance, and compare our results to the results reported by Boyd and Benoit [3] from a similar experiment for metric TSP. In Section 5.3, we propose a stronger conjecture to that of Conjecture 2.4.1 for graph-TSP, which depends on the number of nodes n. Also, we present an infinite family of graph-TSP instances for specific values of n, which we conjecture to reach an integrality gap of 4/3 asymptotically.

5.1 Preliminary details and notation

Let $K_n = (V, E)$ be the complete graph on n nodes with non-negative edge costs $c \in \mathbf{R}^E$, where c is a graph metric i.e., the cost of each edge uv in K_n is equal to the cost of a shortest path between u and v in an underlying undirected unweighted graph. Recall the integer linear programming (ILP) formulation for TSP:

$$
\begin{aligned}
(5.1.1) \qquad\qquad & \text{minimize} & & \sum_{e \in E} c_e x_e & & \\
& \text{s.t.} & & x(\delta(v)) = 2 & & \text{for all } v \in V, \\
& & & x(\delta(S)) \geq 2 & & \text{for all } S \subset V,\ 3 \leq |S| \leq n-3, \\
& & & 0 \leq x_e \leq 1 & & \text{for all } e \in E, \\
& & & x_e \text{ integer} & & \text{for all } e \in E.
\end{aligned}
$$

Recall also that the we obtain the LP relaxation for ILP (5.1.1) by relaxing the integer requirement. For fixed n, let $GTSP_n$ and $GTSP_n^{LP}$ represent the cost of an optimal ILP solution and the cost of an optimal subtour LP solution, respectively. The integrality gap, $\alpha GTSP_n$, for a fixed n is defined as follows:

$$\alpha GTSP_n = \sup_{c \text{ graph metric}} \left(\frac{GTSP_n}{GTSP_n^{LP}} \right).$$

We wish to investigate the value of $\alpha GTSP_n$ for particular classes of graphs. We need only focus our attention on graph-TSP instances of size $n \geq 6$, as it is known that SEP and metric TSP are equivalent for $n \leq 5$ [24], and thus $\alpha GTSP_n = 1$ for such values of n.

5.1.1 Experiment and procedure

We implement our program for this experiment using the C programming language and perform the experiment on a 2.4 GHz Intel Core i5 Macbook Pro with 4GB 1333 MhZ DDR3 memory.

We focus our attention on three classes of graphs: general connected graphs on $6 \leq n \leq 10$ nodes, cubic connected graphs and subcubic connected graphs on $6 \leq n \leq 16$ nodes. We use the nauty software package developed by Brendan D. McKay [32] to generate all non-isormophic connected graphs on $6 \leq n \leq 10$, and non-isormophic cubic and subcubic connected graphs on $6 \leq n \leq 16$ nodes. Note that we do not investigate general connected graph-TSP instances beyond $n = 10$ nodes. In fact, beyond $n = 10$ nodes, there are too many graphs to investigate using our proposed method. Indeed, at $n = 11$ nodes, there are nearly one billion non-isomorphic connected graphs. As such, to evaluate the integrality gap for values beyond $n = 10$ nodes, we have restricted our graph-TSP instances to cubic and subcubic connected graphs.

Our procedure for the general connected graphs on $6 \leq n \leq 10$ nodes is as follows: For each general connected graph-TSP instance $G = (V, E)$ on n nodes, we first compute its metric completion by solving the all-pairs shortest path problem on G using the Floyd-Warshall algorithm [14]. We build the corresponding LP for G and find the exact cost of its optimal subtour LP solution, $GTSP_n^{LP}$, using the QSopt_ex rational LP solver [2]. It is important to note that QSopt_ex works in exact rational arithmetic, which ensures that no rounding errors occur in the calculation of the LP solutions, and so they are exact. We then form the corresponding TSPLIB file for G, and find the cost of the cheapest TSP tour, $GTSP_n$, using Concorde. We examine the ratio between $GTSP_n$ and $GTSP_n^{LP}$ and over all instances, and obtain a complete set of all graphs that give $\alpha GTSP_n$.

The procedure for cubic and subcubic connected graphs on $6 \leq n \leq 16$ nodes is exactly as described for general connected graphs, with the exception that we use Concorde to find both $GTSP_n^{LP}$ and $GTSP_n$, as QSopt_ex was too time-consuming for finding $GTSP_n^{LP}$ for all of these instances. Note that this means there is some small possibility that rounding errors have occurred, and that $GTSP_n^{LP}$ is not exact in these cases.

5.2 Results and analysis

The results of the computational study are presented in Tables 5.2.1 to 5.2.4. Each table shows the value of $\alpha GTSP_n$ for specified n for each class of non-isomorphic connected graphs on n nodes. Additionally Figures 5.2.1 to 5.2.5 in Section 5.2.1 show the graph-TSP instance(s) that give the value $\alpha GTSP_n$, for each class of graphs and values of n.

n	$\alpha GTSP_n$	$\alpha MTSP_n$
6	1	10/9
7	16/15	9/8
8	10/9	8/7
9	10/9	7/6
10	8/7	20/17

Table 5.2.1: Integrality gap for all non-isomorphic connected graph-TSP and metric TSP instances (see Benoit and Boyd [3] for the latter) on $6 \leq n \leq 10$ nodes. Note that we denote the integrality gap for SEP for metric TSP for graphs on n nodes by $\alpha MTSP_n$.

n	Number of graphs tested	Number of graphs that give $\alpha GTSP_n$	$GTSP_n$	$GTSP_n^{LP}$	$\alpha GTSP_n$	Elapsed time
6	112	112	6, 7, 8, 9 or 10	6, 7, 8, 9 or 10	1	0.27 seconds
7	853	4	8	15/2	16/15	2.68 seconds
8	11117	2	10	9	10/9	50.11 seconds
9	261080	4	10	9	10/9	2,358.09 seconds
10	11716571	1	12	21/2	8/7	approximately 3 days

Table 5.2.2: Detailed summary of integrality gap for all non-isomorphic connected graphs on $6 \leq n \leq 10$ nodes.

n	Number of graphs tested	Number of graphs that give $\alpha GTSP_n$	$GTSP_n$	$GTSP_n^{LP}$	$\alpha GTSP_n$	Elapsed time
6	2	2	6.0	6.0	1	0.007 seconds
8	5	5	8.0	8.0	1	0.013 seconds
10	19	1	11.0	10.0	11/10	0.120 seconds
12	85	1	13.0	12.0	13/12	0.434 seconds
14	509	5	15.0	14.0	15/14	2.964 seconds
16	4060	1	18.0	16.5	12/11	30.507 seconds

Table 5.2.3: Detailed summary of integrality gap for all non-isomorphic cubic connected graphs on $6 \leq n \leq 16$ nodes.

n	Number of graphs tested	Number of graphs that give $\alpha GTSP_n$	$GTSP_n$	$GTSP_n^{LP}$	$\alpha GTSP_n$	Elapsed time
6	29	29	6.0, 7.0, 8.0 9.0 or 10.0	6.0, 7.0, 8.0, 9.0 or 10.0	1	0.05 seconds
7	64	1	8.0	7.5	16/15	0.12 seconds
8	194	1	10.0	9.0	10/9	0.37 seconds
9	531	1	10.0	9.0	10/9	1.19 seconds
10	1733	1	12.0	10.5	8/7	5.79 seconds
11	5524	1	14.0	12.0	7/6	19.76 seconds
12	19430	2	14.0	12.0	7/6	80.12 seconds
13	69322	1	16.0	13.5	32/27	313.68 seconds
14	262044	1	18.0	15.0	6/5	1,392.72 seconds
15	1016740	1	18.0	15.0	6/5	6,238.77 seconds
16	4101318	1	20.0	16.5	40/33	31,470 seconds

Table 5.2.4: Detailed summary of integrality gap for all non-isomorphic subcubic connected graphs on $6 \leq n \leq 16$ nodes.

The results show that for each class of graphs, and value of n, the integrality gap does not exceed $\frac{4}{3}$. In fact, such results provide a verification of Conjecture 2.4.1 for graph-TSP, for connected general graphs up to $n = 10$ nodes and for connected cubic and subcubic graphs up to $n = 16$ nodes. Note that there is no gap for $n = 6$ for general graphs, which is not the case for metric metric TSP. However, looking at the rate at which the integrality gap increases, which is clearly much slower here, we can conclude that in general, the subtour LP for TSP provides a better lower bound for graph-TSP than for metric TSP.

Observe that for each value of n, the graph-TSP instances that give the gap $\alpha GTSP_n$ are 2-node connected, subcubic, and their corresponding optimal subtour LP solution is half-integer i..e, each edge has value 0, 1 or $\frac{1}{2}$ in the subtour LP solution. In fact, with the exception of a single graph, the optimal subtour LP solution for each graph share a common structure; the $\frac{1}{2}$-edges form two disjoint triangles joined by three disjoint paths of 1-edges. Indeed, this observation seems to suggest that for each fixed value of n there is such a graph-TSP instance which gives the value

$\alpha GTSP_n$. However, there is one subcubic connected graph-TSP instance on $n = 12$ nodes whose optimal subtour LP solution does not follow this structure, but gives the value $\alpha GTSP_n$ over all non-isomorphic subcubic connected graphs on $n = 12$ nodes. In fact, this graph is the support graph of a vertex of the subtour polytope (2.2.2) (see Figure 5.2.5(a)). Its $\frac{1}{2}$-edges form a single cycle, such that two nodes on this cycle are adjacent to a degree two node, and its 1-edges form three disjoint 1-paths of length 2. Furthermore, this graph is distinct from the other graphs, as its optimal subtour LP solution only uses edges that are in its original edge set, in other words, the cost of its optimal solution is n. Upon close examination, there seems to be a pattern with respect to the structure of the graphs that give the integrality gap for values of n modulo 3. In fact, we observe that for our specified values of n, the integrality gap increases for values of $n \equiv 0 \pmod 3$, whereas the integrality gap for values of $n \equiv 1 \pmod 3$ and $n \equiv 2 \pmod 3$ are identical. We discuss this further in Section 5.3.

5.2.1 Graphs that give integrality gap for small values of n

Here, we present the graphs that give the value $\alpha GTSP_n$ for general connected graph-TSP instances on $7 \leq n \leq 10$ nodes. We also include the vertex of the subtour polytope that gives the value of $\alpha GTSP_n$ for subcubic connected graph-TSP on $n = 12$ nodes. We note that in a similar experiment for all metric costs, Benoit and Boyd [3] report that for each value of $6 \leq n \leq 10$ there is a unique vertex which gives the integrality gap for SEP for metric TSP on n nodes i.e., $\alpha MTSP_n$. In fact, for values $n \equiv 0 \pmod 3$, the graph which gives the value $\alpha MTSP_n$ has the same structure as one of the graphs which gives the value $\alpha GTSP_n$. However, for those same values of

n, we find that there are multiple graph-TSP instances which give the value $\alpha GTSP_n$.

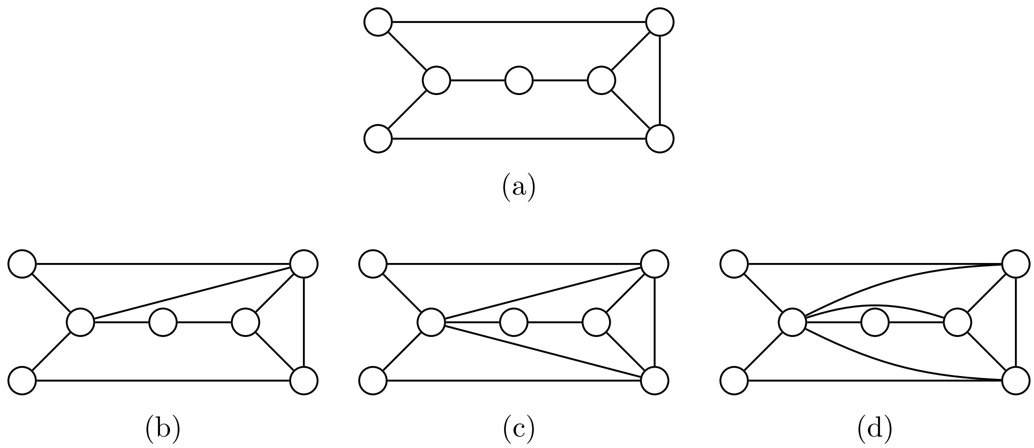


Figure 5.2.1: Graphs on $n = 7$ nodes which give the value $\alpha GTSP_7 = 16/15$.

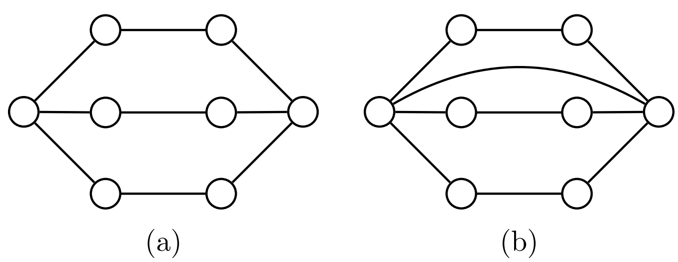
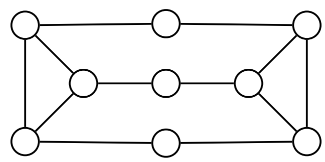
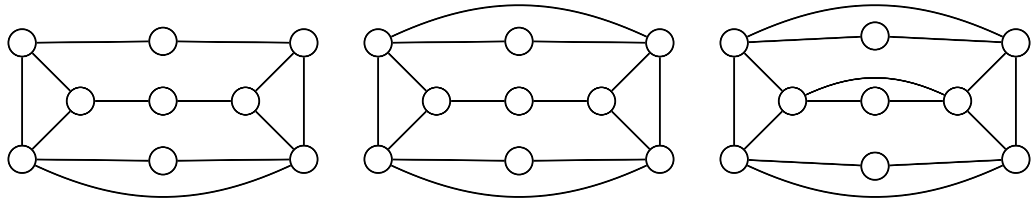


Figure 5.2.2: Graphs on $n = 8$ nodes which give the value $\alpha GTSP_8 = 10/9$.

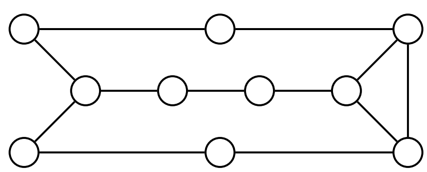


(a)



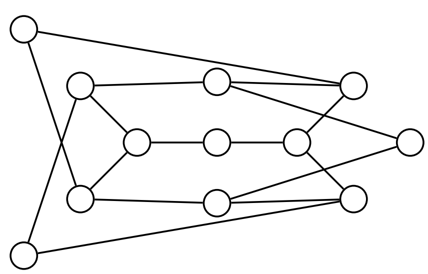
(b) (c) (d)

Figure 5.2.3: Graphs on $n = 9$ nodes which give the value $\alpha GTSP_9 = 10/9$.



(a)

Figure 5.2.4: Graph on $n = 10$ nodes which gives the value $\alpha GTSP_{10} = 8/7$.



(a)

Figure 5.2.5: Special subcubic graph on $n = 12$ nodes which gives the value $\alpha GTSP_{12} = 7/6$ over all connected subcubic graph-TSP instances. Note that this graph is the support graph of a vertex of the subtour polytope (2.2.2)

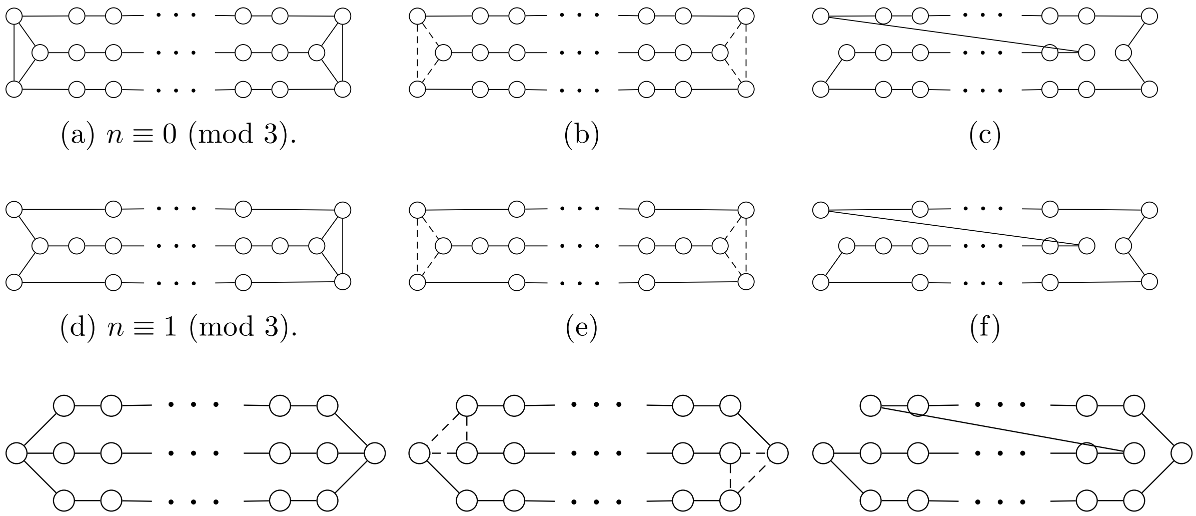
5.3 Stronger conjecture

In the previous section it was noted that for each value of n mod 3, the structure of the graphs that give the integrality gap, $\alpha GTSP_n$, follow a pattern. Indeed, consider the graphs in Figure 5.2.1(a) and 5.2.4(a), where $n \equiv 1 \pmod 3$. Observe that the latter graph is simply the former one, in which one 1-edge in each 1-path is subdivided. A similar pattern can be concluded by examining each graph that gives the integrality gap for values of n mod 3. In fact, observe that by extrapolating each graph to higher values of n, by means of subdividing one 1-edge in each 1-path, we form an infinite family of graphs for each value of n mod 3. These families, along with their conjectured optimal subtour LP solution and conjectured optimal TSP tour, can be seen in Figure 5.3.1. Upon close examination of the underlying structure of each graph in these families, and their conjectured optimal subtour LP solution and optimal TSP tour, we propose the following conjecture for the value of $\alpha GTSP_n$.

Conjecture 5.3.1. *For all integers $n \geq 6$, $\alpha GTSP_n = \alpha' GTSP_n$ where $\alpha' GTSP_n$ is defined as follows:*

$$\alpha' GTSP_n = \begin{cases} \dfrac{4n-6}{3n} & \text{if } n \equiv 0 \ (mod\ 3) \\[2ex] \dfrac{4(n-1)}{3(n+\frac{1}{2})} & \text{if } n \equiv 1 \ (mod\ 3) \\[2ex] \dfrac{4n-2}{3n+3} & \text{if } n \equiv 2 \ (mod\ 3). \end{cases}$$

It would follow from Conjecture 5.3.1 that $\alpha GTSP_n = \frac{4}{3}$ as $n \to \infty$. In fact, under certain assumptions, we can show that for each value of n modulo 3, one of the graphs in Figure 5.3.1 is such that $GTSP_n / GTSP_n^{LP} = \alpha' GTSP_n$ for $n \geq 6$.



(a) $n \equiv 0 \pmod 3$. (b) (c)

(d) $n \equiv 1 \pmod 3$. (e) (f)

(g) $n \equiv 2 \pmod 3$. (h) (i)

Figure 5.3.1: Patterns developed from subdividing 1-edges in subcubic graph which give the value $\alpha GTSP_n$ for $n \equiv 0 \pmod 3$, $n \equiv 1 \pmod 3$ and $n \equiv 2 \pmod 3$, along with their corresponding conjectured optimal subtour LP solution and conjectured optimal TSP tour. Note that for graphs (b), (e), (h), the solid edges represent 1-edges and the dashed edges represent $\frac{1}{2}$-edges.

Theorem 5.3.2. *Consider the graphs presented in Figures 5.3.1(a), 5.3.1(d) and 5.3.1(g). Assume that the structure of their respective optimal subtour LP solution and their optimal TSP tour correspond to the ones in Figure 5.3.1. Then, for $n \geq 6$, there exists a graph-TSP instance on n nodes for which*

$$\frac{GTSP_n}{GTSP_n^{LP}} = \alpha' GTSP_n,$$

where $\alpha' GTSP_n$ is as defined in Conjecture 5.3.1.

Proof. We show that the graphs presented in Figures 5.3.1(a), 5.3.1(d) and 5.3.1(g) give the value $\alpha' GTSP_n$ for each value of n. **Case 1.** Let $n \equiv 0 \pmod 3$. Consider the graph $G = (V, E)$ in Figure 5.3.1(a), and let k be the length of its longest 1-path.

Under the assumption that the structure of the optimal subtour LP solution and the optimal TSP tour correspond to the ones in Figures 5.3.1(b) and 5.3.1(c), respectively, we can conclude that $GTSP_n = 4k+2$ and $GTSP_n^{LP} = 3k+3$. As $n = 3k+3$, we have that:

$$\frac{GTSP_n}{GTSP_n^{LP}} = \frac{4k+2}{3k+3} = \frac{4\left(\frac{1}{3}(n-1)\right)+2}{3\left(\frac{1}{3}(n-1)\right)+3} = \frac{4n-6}{3n},$$

as required.

Case 2. Let $n \equiv 1 \pmod 3$. Consider the graph $G = (V, E)$ in Figure 5.3.1(d), and let k be the length of its longest 1-path. Under the assumption that the structure of the optimal subtour LP solution and the optimal TSP tour correspond to the ones in Figures 5.3.1(e) and 5.3.1(f), we can conclude that $GTSP_n = 4k$ and $GTSP_n^{LP} = 3k + 3/2$. As $n = 3k+1$, we have that:

$$\frac{GTSP_n}{GTSP_n^{LP}} = \frac{4k}{3k+\frac{3}{2}} = \frac{4\left(\frac{1}{3}(n-1)\right)}{3\left(\frac{1}{3}(n-1)+\frac{3}{2}\right)} = \frac{4(n-1)}{3(n+\frac{1}{2})},$$

as required.

Case 3. Let $n \equiv 2 \pmod 3$. Consider the graph $G = (V, E)$ in Figure 5.3.1(g), and let k be the length of its longest 1-path. Under the assumption that the structure of the optimal subtour LP solution and the optimal TSP tour correspond to the ones in Figures 5.3.1(h) and 5.3.1(i), we can conclude that $GTSP_n = 4k-2$ and $GTSP_n^{LP} = 3k$. As $n = 3k-1$, we have that:

$$\frac{GTSP_n}{GTSP_n^{LP}} = \frac{4k-2}{3k} = \frac{4\left(\frac{1}{3}(n+1)\right)-2}{3\left(\frac{1}{3}(n+1)\right)} = \frac{4n-2}{3n+3},$$

as required. $\square$

Therefore, there is an infinite family of graphs, each an instance of graph-TSP, for $n \equiv 0 \pmod 3$, $n \equiv 1 \pmod 3$ and $n \equiv 2 \pmod 3$, for which the value of $\alpha' GTSP_n$ in Conjecture 5.3.1 is tight. In fact, if the conjecture holds true, then we could conclude that integrality gap for SEP is 4/3 for the graph-TSP.

Chapter 6

A 4/3-approximation algorithm for special subquartic subtour support graphs

In Chapter 5, we have shown that the integrality gap for SEP for graph-TSP never exceeds $\frac{4}{3}$ for small instances of different classes of graphs. In this Chapter, we propose a simplified $\frac{4}{3}$-approximation algorithm for graph-TSP instances that correspond to subquartic support graphs G_{x^*} of vertices x^* in the subtour polytope (2.2.2) for which $OPT_{LP}(G_{x^*}) = n$. Newman [34] has shown that the Mömke and Svensson approximation algorithm for graph-TSP [44] yields a performance guarantee of $\frac{4}{3}$ for 2-edge connected subquartic graph-TSP instances with optimal subtour LP solution of cost n. Our approximation algorithm, however, removes the need to solve a minimum cost circulation problem. In fact, it uses a single application of the minimum cost $\hat{U}$-tight odd join algorithm to find a restricted odd join in G_{x^*} and a particular spanning tree, amongst a collection of potentially exponentially many, with which it produces a TSP tour of cost at most $\frac{4}{3}n - \frac{2}{3}$. In Section 6.1 we give an overview of the approximation algorithm and in Section 6.2 we prove its performance guarantee. In Section 6.3 we discuss the difficulty of extending our approach beyond subquartic support graphs.

6.1 Procedure

Let $G_{x^*} = (V, E)$ correspond to a subquartic support graph of a vertex x^* of the subtour polytope (2.2.2) such that x^* is not a TSP tour, and let $c \in \mathbf{R}^E$ be a graph metric. We restrict our support graphs to those where $OPT_{LP}(G_{x^*}) = n$. As such, each edge of G_{x^*} has cost one i.e., $c_e = 1$ for all $e \in E$. Now, we find a greedy DFS tree $T_{x^*} = (V, F)$ rooted at node r of degree-four in G_{x^*}. Denote the set of back edges by the set $B := B(T_{x^*})$. Now, for each tree edge t in F we find its corresponding swap set $S(t)$. As done previously, we denote the set of tree edges with non-empty swap sets by F_{swap}. Using the usual costs used in the Mömke and Svensson Method, we define a new set of costs c^* for $E(G_{x^*})$ for which also negate the costs of the edges in F_{swap}:

$$c^*_e = \begin{cases} c_e & \text{if } e \in F \setminus F_{\text{swap}}, \\ -c_e & \text{otherwise,} \end{cases}$$

for all edges e in E. We now wish to find a restricted odd join J for G_{x^*} of a particular cost, namely, one bounded by the number of nodes and the number of back edges. Consider the spanning tree $T^* = (V, F^*)$ obtained by removing all of the edges in $F_{\text{swap}} \cap J$ and replacing them with an edge from their respective swap set, namely one that is not in J. Then, by applying the Mömke and Svensson Method to T^*, we obtain a spanning Eulerian multi-subgraph $H^* = (V, E^*)$ of G_{x^*}. We compute an Euler tour of H^*, shortcut it, and return the resulting TSP tour.

The challenge is to show there exists a restricted odd join for G_{x^*}, such that H^* contains no more than $\frac{4}{3}n - \frac{2}{3}$ edges. In the next section, we show that such a restricted odd join exists, and that the number of edges in H^* using this restricted odd join is as desired.

6.2 Performance guarantee

Before proceeding, we prove the following result, which ensures the feasibility of a $\hat{U}$-tight odd join vector defined in this section.

Lemma 6.2.1. *No back-edge is in a 2-edge cut of G_{x^*}.*

Proof. Assume the lemma is false, and suppose e_1 is a back edge of G_{x^*} in a 2-edge cut $\delta(U) = \{e_1, e_2\}$. Recall, by Lemma 4.2.2, that all edges for which $x^*_e = 1$ are in the greedy DFS tree. Therefore, $x^*_{e_1} < 1$ and $x^*_{e_2} \leq 1$. Thus, we have that

$$x^*(\delta(U)) = x^*_{e_1} + x^*_{e_2} < 1 + 1 = 2,$$

violating the subtour elimination constraint (2.2.2)(ii.). $\square$

Following the ideas from the heuristic described in Chapter 4, we find a restricted odd join for G_{x^*} by finding a minimum cost $\hat{U}$-tight odd join on auxiliary graph G'_{x^*} formed by adding gadgets to G_{x^*}, namely those introduced in Chapter 4 Section 4.3.2 (see Figure 4.3.2). Note that G'_{x^*} is 2-edge connected since G_{x^*} is 2-node connected.

Also note that the only back edges in 2-edge cuts in G'_{x^*} are those incident to degree two nodes created by the gadgets.

Lemma 6.2.2. *No back-edge is in a 2-edge cut in G'_{x^*}, with the exception of those incident to a gadget node of degree two.*

Proof. Assume the lemma is false, and let e be a back edge in such a 2-edge cut in G'_{x^*}. It follows from Lemma 6.2.1 that e is in a 3-edge cut in G_{x^*}, namely one in which the edges other than e are incident to a node of degree four. Specifically, such a node u of degree-four has one incoming tree edge, one outgoing tree edge, one incoming back edge and one outgoing back edge, denoted a, b, c and d respectively. Now, without loss of generality, it follows from Lemma 4.2.2 that $x^*_b \geq x^*_c$ and $x^*_e < 1$. As x^* is in the subtour polytope (2.2.2), we have that $x^*_b + x^*_c + x^*_e \geq 2$, and thus $x^*_b + x^*_c > 1$. Then, by the degree constraint (2.2.2)(i) for node u, it must be that $x^*_a + x^*_d < 1$, and thus $x^*_a + x^*_d + x^*_e < 2$, violating the subtour elimination constraint (2.2.2)(ii). $\square$

Let E_g be the set of gadget edges in G'_{x^*}. Using the usual costs from the Mömke and Svensson Method, we define a new set of costs c' for $E(G'_{x^*})$, where we assign a cost of zero to the gadget edges:

$$c'_e = \begin{cases} c_e & \text{if } e \in F \setminus F_{\text{swap}}, \\ 0 & \text{if } e \in E_g, \\ -c_e & \text{otherwise.} \end{cases}$$

We now wish to make use of Lemma 6.2.2 to find a feasible $\hat{U}$-tight odd join vector for G'_{x^*}. Let X be the set of back edges incident to a gadget node of degree three, or incident to node r, in G'_{x^*}. Then,

$$|F_{\text{swap}}| = |B| - \frac{1}{2}|X| - 1,$$

where we subtract one as to not count the tree edge incident to the root node twice. Now, define the vector j^* as follows:

$$j^*_e = \begin{cases} \dfrac{2}{3} & \text{if } e \in X, \\ \\ \dfrac{1}{3} & \text{if } e \in E(G'_{x^*}) \setminus X, \end{cases} \tag{6.2.1}$$

for all edges e in $E(G'_{x^*})$. In light of Lemma 6.2.2, it must be that $E(G'_{x^*}) \setminus X$ contains each edge in a 2-edge cut, and those edges in X are contained in cuts of size at least three. Therefore, by invoking Theorem 3.3.3 we obtain the following corollary:

Corollary 6.2.3. *Let $\hat{U}$ be the set of gadget nodes labelled x in G'_{x^*} and let vector j^* be as defined in (6.2.1). Then, j^* is feasible for the $\hat{U}$-tight odd join polytope (3.3.1).*

A direct consequence of Corollary (6.2.3) is the following:

Lemma 6.2.4. *The minimum cost $\hat{U}$-tight odd join for G'_{x^*} has cost at most $\frac{1}{3}n - |B| + \frac{1}{3}$.*

Proof. Let vector j^* be as defined in (6.2.1), then it follows from Corollary 6.2.3 that j^* is feasible for the $\hat{U}$-tight odd join polytope (3.3.1). Therefore, j^* can be written as a convex combination of incidence vectors of $\hat{U}$-tight odd joins. Recall that each non-gadget edge in G'_{x^*} corresponds to a tree edge in F or a back edge in B. Now, it follows from Theorem 2.2.1 that there must be at least one $\hat{U}$-tight odd join J of cost

$$\begin{aligned}
c'(J) &\leq j^*c'(E_g) + j^*c'(F) + j^*c'(B) \\
&= \frac{1}{3}(0) + \frac{1}{3}|F \setminus F_{\text{swap}}| - \frac{1}{3}|F_{\text{swap}}| - \frac{1}{3}|B \setminus X| - \frac{2}{3}|X| \\
&= \frac{1}{3}|F| - \frac{2}{3}|F_{\text{swap}}| - \frac{1}{3}|B| - \frac{1}{3}|X| \\
&= \frac{1}{3}|F| - \frac{2}{3}\left(|B| - \frac{1}{2}|X| - 1\right) - \frac{1}{3}|B| - \frac{1}{3}|X| \\
&= \frac{1}{3}|F| - |B| + \frac{2}{3} \\
&= \frac{1}{3}(n-1) - |B| + \frac{2}{3} \\
&= \frac{1}{3}n - |B| + \frac{1}{3}.
\end{aligned}$$

Therefore, for a minimum cost $\hat{U}$-tight odd join J^*, it must be that $c(J^*) \leq \frac{1}{3}n - |B| + \frac{1}{3}$, as required. $\square$

Let J^* be a minimum cost $\hat{U}$-tight odd join, then it follows from Corollary 4.3.5 that $J = J^* \setminus E_g$ is a restricted odd join for G_{x^*} of cost the same as $c^*(J^*)$. Let T^* be the spanning tree of G_{x^*} obtained by removing all edges in $F_{\text{swap}} \cap J$ and replacing each one with a back edge, from the respective swap sets, that does not intersect J. Then, using the usual Mömke and Svensson Method, we form a spanning Eulerian multi-subgraph $H^* = (V, E^*)$ by adding the edges of $F \cap J$ to G_{x^*} and removing the edges in $\bar{F} \cap J$. We are now ready to prove our main result.

Theorem 6.2.5. *The spanning Eulerian multi-graph $H^* = (V, E^*)$ produced by the algorithm has cost at most $\frac{4}{3}n - \frac{2}{3}$.*

Proof. Let J^* be a minimum cost $\hat{U}$-tight odd join for G'_{x^*}. It follows from Theorem 4.3.5 that $J = J^* \setminus E_g$ is a restricted odd join for G_{x^*} of cost the same as $c^*(J^*)$. Let

$T = (V, F^*)$ be the correspond spanning tree for J. Therefore, we have that

$$\begin{aligned} c(H^*) &= |E| + c^*(J) \\ &\leq |F^*| + |B| + \frac{1}{3}n - |B| + \frac{1}{3} \\ &= (n-1) + \frac{1}{3}n + \frac{1}{3} \\ &= \frac{4}{3}n - \frac{2}{3}, \end{aligned}$$

as required. □

Given that the TSP tour obtained by the algorithm cannot be more expensive that the cost of H^*, it must be that its cost is at most $\frac{4}{3}n - \frac{2}{3}$.

6.3 Beyond subquartic subtour support graphs

We now discuss some challenges of extending our approach to graphs beyond subquartic subtour support graphs for which the optimal subtour LP solution is n.

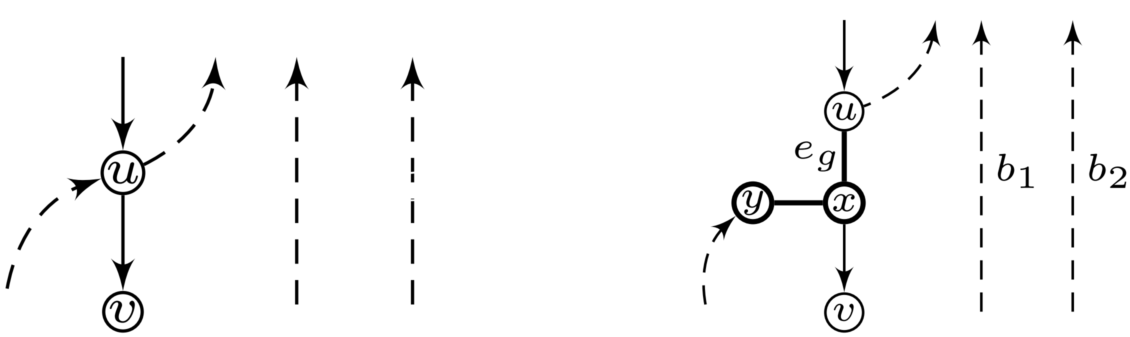
Consider subquartic subtour support graphs G_{x^*} of a vertex x^* of the subtour polytope (2.2.2) such that $OPT_{LP}(G_{x^*}) > n$. Then, it must be that the cost of some edges of G_{x^*} will be greater than one, and therefore, the arguments in the previous section fail for these particular graphs.

Now, consider subtour support graphs in which every node has degree at most $k > 4$, and such that the cost of their corresponding optimal subtour LP solution is exactly n. Without loss of generality, consider the case $k = 5$ i.e., *subquintic* subtour support graphs. Let G_{x^*} be such a subquintic support graph for x^* in the subtour polytope (2.2.2). Following our approach in the previous section, let T_{x^*} be a greedy

DFS tree of G_{x^*} rooted at a node of degree five in G_{x^*}. Define X_1 to be the set of back edges incident to gadget nodes of degree four, including the root node, in G'_{x^*}. Similarly, define X_2 to be the set of back edges that are incident to the gadget nodes of degree-three in G'_{x^*}. Note that sets X_1 and X_2 are disjoint. Let vector j^* be defined as follows

$$j^*_e = \begin{cases} \dfrac{7}{9} & \text{if } e \in X_1 \\[2ex] \dfrac{2}{3} & \text{if } e \in X_2 \\[2ex] \dfrac{1}{3} & \text{if } e \in E(G'_{x^*}) \setminus (X_1 \cup X_2). \end{cases} \tag{6.3.1}$$

for all edges e in $E(G'_{x^*})$. If j^* were to be feasible for the $\hat{U}$-tight odd join polytope (3.3.1), then, using similar arguments used in the previous section, it would be possible to conclude that there is a restricted odd join of cost at most $\frac{1}{3}n - |B| + \frac{1}{3}$. However, j^* is not feasible for the $\hat{U}$-tight odd join polytope (3.3.1). Indeed, consider the cut $\delta(U)$ which consists of two back edges $b_1, b_2 \in X_1$ and one gadget edge $e_g \in E(G'_{x^*}) \setminus (X_1 \cup X_2)$. Figure 6.3.1 illustrates such a cut.



(a) Snapshot of cut in G_{x^*}. (b) Snapshot of cut in G'_{x^*}.

Figure 6.3.1: Example of a problematic cut.

Now, consider the set $W = \{b_1, b_2\}$, and note that $j^*(b_1) = j^*(b_2) = \frac{7}{9}$ and $j^*(e_g) = \frac{1}{3}$. Then

$$j^*(\delta(U) \setminus W) + |W| - j^*(W) = \frac{1}{3} + 2 - \left(\frac{7}{9} + \frac{7}{9}\right) = \frac{7}{9} < 1.$$

Consequently, j^* violates constraint (3.3.1) (ii) of the $\hat{U}$-tight odd join polytope (3.3.1), given that $|\delta(U) \setminus W|$ is odd. One way to remedy this problem, would be to find a greedy DFS tree such that the problematic cut depicted in Figure 6.3.1(b) is not possible. Alternatively, it may be possible to define a new vector, or even offset the values of j^* defined in (6.3.1), such that it is feasible for the $\hat{U}$-tight odd join polytope (3.3.1).

Chapter 7

Conclusion and future work

In this thesis, we showed that the Mömke and Svensson Method for forming a spanning Eulerian multi-graph is equivalent to the Christofides Method, and we show that it yields a $\frac{3}{2}$-approximation algorithm for metric TSP when applied to a minimum cost spanning tree. We extended ideas used by Mömke and Svensson [44] for approximating graph-TSP, and developed a new heuristic for metric TSP. We implemented this heuristic, and used it to test the quality of the TSP tours produced by a different collection of spanning trees than that used in the Best-of-Many Chrisfotides' algorithm [20]. We showed that on average our heuristic is quite fast and significantly reduces the size of the problem instance prior to producing the TSP tour. Indeed, over all the test data, the number of edges in the auxiliary graph on which we solve the minimum cost perfect matching problem is on average within 0.59 of the number of edges in the initial support graph. We have also shown that the collection of trees we have tested produce good solutions. Indeed, over all the test data, the cost of the solution produced by our heuristic is on average 4.78% away from the cost of an optimal TSP tour. Additionally, the cost of the solution produced by our heuristic, over all test data, is on average 0.83% and 0.33% away from the cost of the solution produced by the column generation and splitting off variant of the Best-of-Many Christofides' algorithm [19], respectively.

For the special case of graph-TSP, we have shown, through a computational experiment, that the integrality gap for SEP for general graph-TSP instances up to 10 nodes does not exceed $\frac{4}{3}$, improving on the general upper bound of $\frac{7}{5}$ by Sebő and Vygen [42] for the integrality gap for general graph-TSP. In fact, we also show that this is the case for cubic and subcubic graph-TSP instances up to 16 nodes. Additionally, we have conjectured that there are at least three infinite family of graph-TSP instances for which the integrality gap reaches $\frac{4}{3}$ asymptotically. Finally, we have provided a simpler $\frac{4}{3}$-approximation algorithm for subquartic support graphs of with optimal subtour LP solution of cost n.

We propose the following directions for future work:

- Similarly to the restricted Mömke and Svensson Method, it would be interesting to investigate a restricted Christofides Method that we can apply to our collection of trees.

- Investigate the quality of the solutions produced by our heuristic using different shortcutting methods, and providing this shortcutted TSP tour to a (TSP) tour improvement heuristic.

- The quality of the solutions obtained from the experimental evaluation of our heuristic suggests that the subtour LP solution seems to be important in obtaining a good collection of spanning trees. In fact, in a similar experiment in which we applied the Mömke and Svensson Method directly on the DFS tree, instead of our heuristic, we found that the percent error between the cost of the solution produced by this one compared to the cost of an optimal solution is on average 5.44% over all the test data. Therefore, it would be of interest to investigate the quality of different kinds of spanning trees based on the subtour LP solution.

- Further investigate the integrality gap for general connected graph-TSP instances beyond $n = 10$. One approach would be to reduce the number of graphs needed to be processed. In fact, it seems that for any general connected graph TPS instance with 1-edge cuts, there is a 2-edge connected graph which yields a larger gap.

- Investigate ways to extend our proposed $\frac{4}{3}$-approximation algorithm for sub-quartic subtour support instances to other classes of graphs.

Appendix A

Experimental evaluation our heuristic for metric TSP: Individual results

Here we provide individual results obtained when testing our heuristic on each instance in our data set, including two TSPLIB instances that are not in our data set, namely lin318 and pla85900.

In the table(s) below, we provide the name of the problem, its size with respect to the number of nodes, the cost of the cheapest TSP tour produced by our heuristic, the percent error between the cheapest TSP tour produced by our heuristic and the cost of an optimal TSP tour, the elapsed (real) time in seconds to compute an optimal subtour LP solution using Concorde, and the average elapsed (user) time in seconds for ten iterations of our heuristic (excluding time to compute the optimal subtour solution using Concorde).

A.1 Two-dimensional Euclidean TSPLIB instances

General information			Our heuristic: Results		Elapsed Time (seconds)	
Problem name	Size	Optimal	Cost of cheapest TSP tour	Percent error	Concorde (real time)	Ten Iterations of heuristic (user time)
eil51.tsp	51	426	449	5.4%	0.041	0.002
berlin52.tsp	52	7542	7542	0%	0.021	0.001
st70.tsp	70	675	706	4.59%	0.037	0.003
eil76.tsp	76	538	557	3.53%	0.018	0.002
pr76.tsp	76	108159	112077	3.62%	0.03	0.003
rat99.tsp	99	1211	1258	3.88%	0.018	0.004
kroA100.tsp	100	21282	22089	3.79%	0.023	0.003
kroB100.tsp	100	22141	22699	2.52%	0.026	0.003
kroC100.tsp	100	20749	22225	7.11%	0.022	0.003
kroD100.tsp	100	21294	21875	2.73%	0.028	0.004
kroE100.tsp	100	22068	23979	8.66%	0.027	0.002
eil101.tsp	101	629	629	0%	0.029	0.004
lin105.tsp	105	14379	14463	0.58%	0.028	0.002
pr107.tsp	107	44303	44303	0%	0.042	0.001
pr124.tsp	124	59030	60481	2.46%	0.035	0.003
bier127.tsp	127	118282	126637	7.06%	0.051	0.004
ch130.tsp	130	6110	6275	2.7%	0.03	0.004
pr136.tsp	136	96772	101322	4.7%	0.043	0.008
pr144.tsp	144	58537	59995	2.49%	0.075	0.004
ch150.tsp	150	6528	6674	2.24%	0.034	0.005
kroA150.tsp	150	26524	28398	7.07%	0.036	0.004
kroB150.tsp	150	26130	27047	3.51%	0.04	0.005
pr152.tsp	152	73682	76900	4.37%	0.11	0.005
u159.tsp	159	42080	42926	2.01%	0.032	0.004
rat195.tsp	195	2323	2368	1.94%	0.04	0.006
d198.tsp	198	15780	16197	2.64%	0.094	0.006
kroA200.tsp	200	29368	31579	7.53%	0.04	0.006
kroB200.tsp	200	29437	31530	7.11%	0.058	0.005
tsp225.tsp	225	3916	4220	7.76%	0.04	0.006
ts225.tsp	225	126643	132223	4.41%	0.028	0.006
pr226.tsp	226	80369	82577	2.75%	0.15	0.005
gil262.tsp	262	2378	2437	2.48%	0.062	0.005
pr264.tsp	264	49135	49825	1.4%	0.13	0.005
pr299.tsp	299	48191	50768	5.35%	0.062	0.006
fl417.tsp	417	11861	12180	2.69%	0.19	0.007

(Two-dimensional Euclidean TSPLIB instances continued)

General information			Our heuristic: Results		Elapsed Time (seconds)	
Problem name	Size	Optimal	Cost of cheapest TSP tour	Percent error	Concorde (real time)	Ten Iterations of heuristic (user time)
pr439.tsp	439	107217	109717	2.33%	0.12	0.011
pcb442.tsp	442	50778	52163	2.73%	0.07	0.012
d493.tsp	493	35002	36320	3.77%	0.188	0.015
u574.tsp	574	36905	39305	6.5%	0.118	0.016
rat575.tsp	575	6773	7249	7.03%	0.08	0.013
p654.tsp	654	34643	35033	1.13%	0.63	0.02
d657.tsp	657	48912	51200	4.68%	0.197	0.017
u724.tsp	724	41910	44115	5.26%	0.128	0.017
rat783.tsp	783	8806	9142	3.82%	0.132	0.018
pr1002.tsp	1002	259045	274326	5.9%	0.226	0.031
u1060.tsp	1060	224094	238621	6.48%	0.306	0.027
vm1084.tsp	1084	239297	252047	5.33%	0.204	0.023
pcb1173.tsp	1173	56892	62154	9.25%	0.158	0.024
d1291.tsp	1291	50801	53842	5.99%	0.678	0.029
rl1304.tsp	1304	252948	273114	7.97%	0.271	0.024
rl1323.tsp	1323	270199	292868	8.39%	0.227	0.022
nrw1379.tsp	1379	56638	59657	5.33%	0.167	0.038
fl1400.tsp	1400	20127	21505	6.85%	1.739	0.042
u1432.tsp	1432	152970	161825	5.79%	0.535	0.028
fl1577.tsp	1577	22249	23837	7.14%	1.784	0.025
d1655.tsp	1655	62128	64384	3.63%	0.514	0.03
u1817.tsp	1817	57201	61014	6.67%	0.765	0.034
d2103.tsp	2103	80450	82804	2.93%	0.605	0.031
lin318.tsp	318	42029	43384	3.22%	0.084	0.006

A.2 Non-Euclidean TSPLIB instances

General information			Our heuristic: Results		Elapsed Time (seconds)	
Problem name	Size	Optimal	Cost of cheapest TSP tour	Percent error	Concorde (real time)	Ten Iterations of heuristic (user time)
gr120	120	6942	7139	2.84%	0.044	0.006
si175	175	21407	21581	0.81%	0.078	0.008
si535	535	48450	48916	0.96%	0.631	0.034
pa561	561	2763	2965	7.31%	0.144	0.033
si1032	1032	92650	92710	0.06%	2.18	0.094

A.3 pla85900

General information			Our heuristic: Results		Elapsed Time (seconds)	
Problem name	Size	Optimal	Cost of cheapest TSP tour	Percent error	Concorde (real time)	Ten Iterations of heuristic (user time)
pla85900.tsp	85900	142382641	151393227	6.33%	131.711	1.785

A.4 Graph instances

Note that *MCC* means that the initial graph was not connected, and thus, the size of the problem reflects the size of the largest connected component.

General information			Our heuristic: Results		Elapsed Time (seconds)	
Problem name	Size	Optimal	Cost of cheapest TSP tour	Percent error	Concorde (real time)	Ten Iterations of heuristic (user time)
adjnounAdjacency.tsp	112	123	124	0.81%	0.131	0.008
contiguousUsa.tsp	49	51	51	0%	0.036	0.001
dolphins.tsp	62	71	71	0%	0.063	0.005
euroRoad.tsp(MCC)	1039	1415	1479	4.52%	0.716	1.113
maayan-pdzbase.tsp(MCC)	161	271	271	0%	0.208	0.013
maayan-Stelzl.tsp(MCC)	1615	2686	2714	1.04%	1.304	3.708
maayanFoodweb.tsp	183	215	216	0.47%	0.68	0.018
opsahlSouthernwomen.tsp	32	36	36	0%	0.07	0.002
ucidata-zachary.tsp	34	44	44	0%	0.017	0.001

A.5 Two-dimensional Euclidean VLSI instances

General information			Our heuristic: Results		Elapsed Time	
Problem name	Size	Optimal	Cost of cheapest TSP tour	Percent error	Concorde (real time)	Ten Iterations of heuristic (user time)
xqf131.tsp	131	564	603	6.91%	0.066	0.005
xqg237.tsp	237	1019	1066	4.61%	0.052	0.007
pma343.tsp	343	1368	1436	4.97%	0.183	0.009
pka379.tsp	379	1332	1379	3.53%	0.322	0.01
bcl380.tsp	380	1621	1753	8.14%	0.106	0.008
pbl395.tsp	395	1281	1318	2.89%	0.096	0.01
pbk411.tsp	411	1343	1375	2.38%	0.2	0.011
pbn423.tsp	423	1365	1472	7.84%	0.191	0.011
pbm436.tsp	436	1443	1521	5.41%	0.124	0.009
xql662.tsp	662	2513	2695	7.24%	0.189	0.019
rbx711.tsp	711	3115	3323	6.68%	0.238	0.017
rbu737.tsp	737	3314	3484	5.13%	0.183	0.017
dkg813.tsp	813	3199	3440	7.53%	0.286	0.021
lim963.tsp	963	2789	2934	5.2%	0.373	0.025
pbd984.tsp	984	2797	2939	5.08%	0.402	0.02
xit1083.tsp	1083	3558	3777	6.16%	0.622	0.031
dka1376.tsp	1376	4666	5050	8.23%	0.434	0.034
dca1389.tsp	1389	5085	5457	7.32%	0.344	0.031
dja1436.tsp	1436	5257	5577	6.09%	0.431	0.04
icw1483.tsp	1483	4416	4676	5.89%	0.394	0.032
fra1488.tsp	1488	4264	4497	5.46%	0.96	0.048
rbv1583.tsp	1583	5387	5797	7.61%	0.373	0.045
fnb1615.tsp	1615	4956	5223	5.39%	0.724	0.033
djc1785.tsp	1785	6115	6520	6.62%	0.999	0.044
dcc1911.tsp	1911	6396	6859	7.24%	0.783	0.048
dkd1973.tsp	1973	6421	6812	6.09%	1.241	0.053
djb2036.tsp	2036	6197	6665	7.55%	0.785	0.054
dcb2086.tsp	2086	6600	6996	6%	0.584	0.042
bva2144.tsp	2144	6304	6747	7.03%	0.767	0.047
xqc2175.tsp	2175	6830	7283	6.63%	1.366	0.057
bck2217.tsp	2217	6764	7202	6.48%	1.037	0.048
xpr2308.tsp	2308	7219	7690	6.52%	0.676	0.051
ley2323.tsp	2323	8352	8937	7%	0.579	0.047
rbw2481.tsp	2481	7724	8355	8.17%	0.78	0.052
pds2566.tsp	2566	7643	8348	9.22%	0.918	0.057
mlt2597.tsp	2597	8071	8859	9.76%	0.639	0.055
irw2802.tsp	2802	8423	9213	9.38%	0.503	0.052
dbj2924.tsp	2924	10128	10968	8.29%	1.065	0.071
dlb3694.tsp	3694	10959	11838	8.02%	1.217	0.101

Appendix B

Experimental evaluation our heuristic for metric TSP: More individual results

Here we provide more individual results obtained when testing our heuristic on each instance in our data set, including two TSPLIB instances that are not in our data set, namely lin318 and pla85900.

In the table(s) below, we provide the name of the problem, the number of nodes and edges in its initial support graph, the number of nodes and edges in the support graph with all the 1-paths replaced by a single edge and the number of nodes and edges in the clique graph on which we solve the minimum cost perfect matching. Additionally, we provide the number of single edges and double edges in the spanning Eulerian multi-subgraph produced by our heuristic. Note that, with the exception of the number of nodes and edges in the initial support graph, the other results are averages over ten iteration of the heuristic.

B.1 Two dimensional Euclidean TSPLIB instances

Info	Initial Support graph		Support graph (no 1-paths)		Clique graph		Spanning Eulerian multi-subgraph	
Problem name	nodes	edges	nodes	edges	nodes	edges	nodes	edges
eil51.tsp	51	54	6	9	12	18	43	8.3
berlin52.tsp	52	52	0	0	0	0	0	0
st70.tsp	70	76	12	18	24	36	58.6	12.3
eil76.tsp	76	84	16	24	32	48	65.8	11.1
pr76.tsp	76	85	18	27	36	54	63.8	13.8
rat99.tsp	99	115	29	45	64.4	98.8	91.2	9.1
kroA100.tsp	100	113	25	38	51.6	78.4	89.4	12.3
kroB100.tsp	100	109	18	27	36	54	83.4	17.9
kroC100.tsp	100	109	18	27	36	54	85.2	15.7
kroD100.tsp	100	112	23	35	48	72.4	88.5	12.3
kroE100.tsp	100	103	6	9	12	18	69.33	30.83
eil101.tsp	101	107	10	16	23.6	37.4	101	0.3
lin105.tsp	105	109	8	12	16	24	102.9	2.5
pr107.tsp	107	107	0	0	0	0	0	0
pr124.tsp	124	132	16	24	32	48	104.1	20.8
bier127.tsp	127	140	25	38	52.6	79.1	107.6	21.3
ch130.tsp	130	150	39	59	79.8	120.5	120.3	12.1
pr136.tsp	136	188	88	140	219	338	125.6	16.9
pr144.tsp	144	158	24	38	51.8	84.3	126.1	19.7
ch150.tsp	150	178	51	79	111	170.3	147	5.3
kroA150.tsp	150	166	31	47	62	95	135.7	16.9
kroB150.tsp	150	170	39	59	78.2	119.1	129.9	23.5
pr152.tsp	152	173	37	58	83.6	127.6	142.5	13.2
u159.tsp	159	169	20	30	40	60	148.1	12
rat195.tsp	195	225	57	87	119	180.9	186.4	12.3
d198.tsp	198	232	63	97	132.2	203.1	186.5	13.8
kroA200.tsp	200	222	41	63	88.6	135.1	165.4	37.2
kroB200.tsp	200	225	48	73	97.2	148.8	157.2	47.5
tsp225.tsp	225	252	50	77	102.4	160	205.6	22.9
ts225.tsp	225	243	36	54	72	108	185.8	41.9
pr226.tsp	226	249	43	66	93	140.1	183.5	44.2
gil262.tsp	262	278	31	47	65.6	98.6	239.4	24.4
pr264.tsp	264	273	18	27	36	54	250.5	14.4
pr299.tsp	299	316	34	51	68	102	244.1	57.6
fl417.tsp	417	438	41	62	83.2	126.2	363.7	54

(Two-dimensional Euclidean TSPLIB instances continued)

Info	Initial Support graph		Support graph (no 1-paths)		Clique graph		Spanning Eulerian multi-subgraph	
Problem name	nodes	edges	nodes	edges	nodes	edges	nodes	edges
pr439.tsp	439	509	116	186	269.6	426.8	419.5	24.8
pcb442.tsp	442	486	84	128	171.6	262.4	411.9	34.3
d493.tsp	493	555	118	180	250	378	453.3	50.2
u574.tsp	574	627	101	154	209.8	321.2	482.2	99.8
rat575.tsp	575	627	100	152	208.8	315.8	469.9	112.5
p654.tsp	654	739	165	250	339.4	511.5	643.4	15.4
d657.tsp	657	738	151	232	316.4	486.6	586.1	84.3
u724.tsp	724	798	141	215	290.8	444.4	638.7	96.6
rat783.tsp	783	840	108	165	226.4	345.4	664.2	125.4
pr1002.tsp	1002	1211	373	582	812.8	1259.7	936.9	91
u1060.tsp	1060	1207	268	415	593.4	904.9	953.7	131.5
vm1084.tsp	1084	1182	188	286	383.8	585.5	969.9	126.6
pcb1173.tsp	1173	1254	157	238	324.6	490.5	1011	178.9
d1291.tsp	1291	1415	206	330	489.4	770.2	1178	125.1
rl1304.tsp	1304	1398	174	268	366.6	565.4	1131.4	188.4
rl1323.tsp	1323	1386	124	187	251.8	379.7	1072.8	263.9
nrw1379.tsp	1379	1585	373	579	802.6	1244.7	1238.9	174
fl1400.tsp	1400	1623	422	645	885.2	1349.2	1171.8	263.3
u1432.tsp	1432	1567	248	383	529.8	816.1	1271.8	179.4
fl1577.tsp	1577	1634	114	171	228	342	1319.1	265.1
d1655.tsp	1655	1765	212	322	438.6	665.5	1511.1	166.7
u1817.tsp	1817	1932	222	337	460.2	696.7	1492.7	341.5
d2103.tsp	2103	2195	166	258	361.2	557.8	1995.9	120.3
lin318.tsp	318	342	43	67	93.8	146.9	301	19.2

B.2 Non-Euclidean TSPLIB instances

Info	Initial Support graph		Support graph (no 1-paths)		Clique graph		Spanning Eulerian multi-subgraph	
Problem name	nodes	edges	nodes	edges	nodes	edges	nodes	edges
gr120.tsp	120	147	45	72	108.8	168.8	108.3	12.8
si175.tsp	175	207	56	88	130.4	201.3	161.7	17.2
si535.tsp	535	627	169	261	364.6	557.9	516.3	27.3
pa561.tsp	561	621	113	173	237	363.5	454.5	114.2
si1032.tsp	1032	1038	12	18	24	36	1015.2	17.2

B.3 pla85900

Info	Initial Support graph		Support graph (no 1-paths)		Clique graph		Spanning Eulerian multi-subgraph	
Problem name	nodes	edges	nodes	edges	nodes	edges	nodes	edges
pla85900	85900	89674	7438	11212	15067.8	22701.7	71589.2	14901.9

B.4 Graph instances

Note that *MCC* means that the initial graph was not connected, and thus, the size of the problem reflects the size of the largest connected component.

Info	Initial Support graph		Support graph (no 1-paths)		Clique graph		Spanning Eulerian multi-subgraph	
Problem name	nodes	edges	nodes	edges	nodes	edges	nodes	edges
adjnounAdjacency.tsp	112	116	8	12	16	24	107.125	5.5
contiguousUsa.tsp	49	49	0	0	0	0	0	0
dolphins.tsp	62	78	25	41	60.8	97.9	61.2	2.2
euroRoad.tsp(MCC)	1039	1110	139	210	281.6	426.4	919.8	134.3
maayan-pdzbase.tsp(MCC)	161	169	15	23	32.4	49	160.6	1.5
maayan-Stelzl.tsp(MCC)	1615	1682	134	201	268	402	1375.1	253.6
maayanFoodweb.tsp	183	217	60	94	138.4	212.2	181.8	5.1
opsahlSouthernwomen.tsp	32	36	8	12	16	24	31.5	0.75
ucidata-zachary.tsp	34	34	0	0	0	0	0	0

B.5 Two-dimensional Euclidean VLSI instances

Info	Initial Support graph		Support graph (no 1-paths)		Clique graph		Spanning Eulerian multi-subgraph	
Problem name	nodes	edges	nodes	edges	nodes	edges	nodes	edges
xqf131.tsp	131	144	26	39	52	78	105	28.6
xqg237.tsp	237	277	71	111	156.4	244.1	214.8	28.1
pma343.tsp	343	397	101	155	216	331	305.2	46.6
pka379.tsp	379	449	121	191	270.4	422.9	361.3	25.1
bcl380.tsp	380	412	58	90	127.8	196.3	315.4	69.9
pbl395.tsp	395	449	103	157	215.4	327.1	356.1	47
pbk411.tsp	411	449	69	107	149	229.7	385.8	30.3
pbn423.tsp	423	473	90	140	194.2	299.9	373.5	55.5
pbm436.tsp	436	468	62	94	126	192	384	58.3
xql662.tsp	662	784	209	331	468.8	742.8	578.3	96.3
rbx711.tsp	711	783	137	209	287.2	437	614.9	102.1
rbu737.tsp	737	820	150	233	331.8	509	650.1	94.4
dkg813.tsp	813	929	197	313	450.4	708.6	718.3	110.1
lim963.tsp	963	1122	287	446	631.6	973.4	875.3	103.7
pbd984.tsp	984	1077	172	265	365	560.5	862.3	135
xit1083.tsp	1083	1281	336	534	769	1212.8	986.8	114.5
dka1376.tsp	1376	1540	309	473	659.8	1004.5	1178.1	221.5
dca1389.tsp	1389	1546	306	463	624	944	1211	204.7
dja1436.tsp	1436	1627	354	545	753.8	1158.5	1262	199.8
icw1483.tsp	1483	1625	274	416	559.6	850.8	1280	227.6
fra1488.tsp	1488	1724	427	663	933.4	1439	1333.9	185.6
rbv1583.tsp	1583	1805	407	629	875.8	1353.1	1372.4	235.7
fnb1615.tsp	1615	1800	338	523	724.2	1119.9	1423.4	218.1
djc1785.tsp	1785	2010	410	635	886	1366.1	1567.6	244.8
dcc1911.tsp	1911	2203	528	820	1158	1785.4	1690.4	270.7
dkd1973.tsp	1973	2281	556	864	1206.2	1865.5	1780.1	228.8
djb2036.tsp	2036	2315	503	782	1102.4	1703.6	1790.5	281.8
dcb2086.tsp	2086	2247	308	469	640.6	971.7	1816.4	298.6
bva2144.tsp	2144	2418	516	790	1092.8	1664.6	1901.2	275.1
xqc2175.tsp	2175	2477	545	847	1176.6	1826.1	1928.3	286.4
bck2217.tsp	2217	2485	494	762	1053.2	1619.8	1995.4	258.4
xpr2308.tsp	2308	2551	459	702	961	1466.5	2019.3	323.2
ley2323.tsp	2323	2560	441	678	936.8	1434.2	2037	317.3
rbw2481.tsp	2481	2684	377	580	793	1220.5	2095	420.2
pds2566.tsp	2566	2875	558	867	1214.6	1881.1	2096.2	506.3
mlt2597.tsp	2597	2806	403	612	832	1257.6	2146.4	492.8
irw2802.tsp	2802	3027	431	656	889.6	1353.6	2294.3	546.8
dbj2924.tsp	2924	3341	773	1190	1642	2524	2524.7	458.2
dlb3694.tsp	3694	4200	925	1431	1999.6	3077.6	3138.7	626.8

Bibliography

[1] D. Applegate, R. Bixby, V. Chvatal, and W. Cook. Concorde TSP solver 03.12.29 release. World Wide Web. `http://www.math.uwaterloo.ca/tsp/concorde.html`, 2003.

[2] D. Applegate, W. Cook, S. Dash, and D. Espinoza. Qsopt ex. World Wide Web. `http://www.dii.uchile.cl/~daespino/QSoptExact_doc/main.html`, 2010.

[3] G. Benoit and S. Boyd. Finding the exact integrality gap for small traveling salesman problems. *Mathematics of Operations Research*, 33(4):921–931, 2008.

[4] D. Bertsimas and R. Weismantel. *Optimization over integers*. Dynamic Ideas, Belmont, MA, 2005.

[5] A. Bondy and U. Murty. *Graph Theory (Graduate Texts in Mathematics)*, volume 244 of Graduate Texts in Mathematics. Springer, New York, 2008.

[6] S. Boyd and R. Carr. A new bound for the ratio between the 2-matching problem and its linear programming relaxation. *Mathematical programming, Series A*, 86(3):499–514, 1999.

[7] S. Boyd, R. Sitters, S. van der Ster, and L. Stougie. The traveling salesman problem on cubic and subcubic graphs. *Mathematical Programming, Series A*, 144(1):227–245, 2014.

[8] S. C. Boyd and W. R. Pulleyblank. Optimizing over the subtour polytope of the travelling salesman problem. *Mathematical programming*, 49(1):163–187, 1990.

[9] N. Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical Report 388, Graduate School of Industrial Administration, Carnegie Mellon University, 1976.

[10] W. Cook. *In Pursuit of the Traveling Salesman: Mathematics at the Limits of Computation*. Princeton University Press, 2012.

[11] W. J. Cook, W. H. Cunningham, W. R. Pulleyblank, and A. Schrijver. *Combinatorial Optimization*. Wiley, 1998.

[12] J. Edmonds and E. L. Johnson. Matching, Euler tours and the Chinese postman. *Mathematical programming*, 5:88–124, 1973.

[13] H. Fleischner. *Eulerian Graphs and Related Topics*. Annals of Mathematics. Elsevier, Burlington, MA, 1990.

[14] R. W. Floyd. Algorithm 97: Shortest path. *Communications of the ACM*, 5(6):345, 1962.

[15] A. Frank. A survey on T-joins, T-cuts, and conservative weightings. *Combinatorics, Paul Erdös is Eighty*, 2:213–252, 1994.

[16] Y. Fu. Applications of Circulations and Removable Pairings to TSP and 2ECSS. Master's thesis, University of Ottawa, 2014.

[17] H. N. Gabow. Data Structures for Weighted Matching and Nearest Common Ancestors with Linking. In *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 434–443, New York, 1990. Society for Industrial and Applied Mathematics.

[18] Garey, Michael R. and Johnson, David S. *Computers and Intractability: A Guide to the Theory of NP-completeness.* W.H. Freeman and Company, New York, New York, 1979.

[19] K. Genova and D. P. Williamson. An experimental evaluation of the best-of-many christofides' algorithm for the traveling salesman problem. In N. Bansal and I. Finocchi, editors, *Algorithms-ESA 2015*, pages 570–581. Springer, Berlin, 2015. Code and detailed results available at `https://github.com/kylegenova/best-of-many`.

[20] S. O. Gharan, A. Saberi, and M. Singh. A Randomized Rounding Approach to the Traveling Salesman Problem. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 550–559, 2011.

[21] M. T. Goodrich and R. Tamassia. *Data structures and algorithms in Java.* John Wiley & Sons, fith edition, 2008.

[22] M. Grigni, E. Koutsoupias, and C. Papadimitriou. An approximation scheme for planar graph TSP. In *Proceedings., 36th Annual Symposium on Foundations of Computer Science, 1995.*, pages 640–645, 1995.

[23] Grötschel, Martin and Lovász, Lászĺo and Schrijver, Alexander. *Geometric Algorithms and Combinatorial Optimization*, volume 2 of *Algorithms and Combinatorics*. Springer-Verlag, 1998.

[24] A. Hoffman, J. Wolfe, R. Garfinkel, D. Johnson, C. Papadimitriou, P. Gilmore, E. Lawler, D. Shmoys, R. Karp, J. Steele, et al. *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. J. Wiley and Sons, New York, 1985.

[25] D. S. Johnson and C. H. Papadimitriou. Performance guarantees for heuristics. In E. L. Lawler, J. K. Lenstra, A. H. G. Rinnoov Kan, and D. B. Shmoys, editors, *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, pages 145–180. Willey, Chichester, 1985.

[26] R. M. Karp. Reducibility among Combinatorial Problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972.

[27] L. G. Khachiyan. Polynomial algorithms in linear programming. *USSR Computational Mathematics and Mathematical Physics*, 20(1):53–72, 1980.

[28] V. Kolmogorov. Blossom V: a new implementation of a minimum cost perfect matching algorithm. *Mathematical Programming Computation*, 1(1):43–67, 2009. Code available at `http://pub.ist.ac.at/~vnk/software.html`.

[29] J. Kunegis. KONECT: The Koblenz Network Collection. In *Proc. Int. Web Observatory Workshop*, pages 1343–1350, 2013.

[30] A. N. Letchford, G. Reinelt, and D. O. Theis. A faster exact separation algorithm for blossom inequalities. In D. Bienstock and G. Nemhauser, editors, *Integer Programming and Combinatorial Optimization (IPCO IX)*, volume 10 of *Lecture Notes in Computer Science*, pages 196–205, Heidelberg, Germany, 2004. Springer.

[31] R. Matai, S. P. Singh, and M. L. Mittal. Traveling Salesman Problem: An Overview of Applications, Formulations, and Solution Approaches. *Traveling Salesman Problem, Theory and Applications*, pages 1–24, 2010.

[32] B. D. McKay and A. Piperno. Practical graph isomorphism, II. *Journal of Symbolic Computation*, 60:94–112, 2014.

[33] M. Mucha. A 13/9-Approximation for Graphic TSP. *Theory of Computing Systems*, 55(4):640–657, 2014.

[34] A. Newman. An improved analysis of the Mömke-Svensson algorithm for graph-TSP on subquartic graphs. In *Algorithms-ESA 2014-22th Annual European Symposium*, pages 737–749. Springer, 2014.

[35] G. Optimization. Gurobi 5.6.3. `http://www.gurobi.com`, 2014.

[36] M. D. Plummer and L. Lovász. *Matching theory*, volume 29 of *Annals of Mathematics*. Elsevier, 1986.

[37] G. Reinelt. TSPLIB - A traveling salesman problem library. *ORSA journal on computing*, 3:376–384, 1991.

[38] A. Rohe. Instances found at `http://www.math.uwaterloo.ca/tsp/vlsi/index.html`. Accessed July 17, 2016.

[39] A. Schrijver. Min-max results in combinatorial optimization. In A. Bachem, M. Grötschel, and B. Korte, editors, *Mathematical Programming: The State of the Art*, pages 439–500. Springer Verlag, Berlin, 1983.

[40] A. Schrijver. *Theory of Linear and Integer Programming.* Discrete Mathematics and Optimization. Wiley-Interscience, New York, 1986.

[41] A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*, volume 24. Springer, Berlin, 2003.

[42] A. Sebő and J. Vygen. Shorter Tours by Nicer Ears: 7/5-Approximation for the graph-TSP, 3/2 for the Path Version, and 4/3 for Two-edge-connected Subgraphs. *Combinatorica*, 34(5):597–629, 2014.

[43] D. B. Shmoys and D. P. Williamson. Analyzing the held-karp tsp bound: A monotonicity property with application. *Information Processing Letters*, 35(6):281–285, 1990.

[44] T. Mömke and O. Svensson. Approximating graphic TSP by matchings. In *In Proceedings of the 52nd Annual IEEE Symposium on Foundations of Computer Science*, pages 560–569, 2011.

[45] L. A. Wolsey. Heuristic analysis, linear programming and branch and bound. In *Mathematical Programming Study*, 13, pages 121–134. North-Holland Publishing Company, 1980.

Index