THE BOTTLENECK TRAVELING SALESMAN PROBLEM AND SOME VARIATIONS

by

John LaRusic
BCS, University of New Brunswick, 2005

A Thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science
in the Department
of
Mathematics

SIMON FRASER UNIVERSITY
Spring 2010

APPROVAL

Name: John LaRusic

Degree: Master of Science

Title of Thesis: The Bottleneck Traveling Salesman Problem and Some Variations

Examining Committee: Dr. Tamon Stephen
Chair

Dr. Abraham Punnen, Senior Supervisor

Dr. Zhaosong Lu, Supervisor

Dr. Snezana Mitrovic-Minic, External Examiner

Date Approved: January 8, 2010

Declaration of
Partial Copyright Licence

Abstract

We present powerful heuristics for the *bottleneck traveling salesman problem* (BTSP) and closely related problems such as the *maximum scatter traveling salesman problem* (MSTSP) and the *balanced traveling salesman problem*, the later being a new problem which we introduce. Extensive computational results are presented. In particular, our BTSP heuristic produces provably optimal solutions for nearly every problem considered in a very reasonable running-time, both on problems with symmetric cost matrices and problems with asymmetric cost matrices.

We also provide some theoretical analysis of lower bounds for the BTSP and introduce two new lower bound schemes for the BTSP on asymmetric cost matrices. We compliment this with new approximation algorithm with a guaranteed performance ratio for the BTSP on problems satisfying the triangle-inequality.

For two Jeanettes

"Traveling salesmen... lived like artists, like actors whose product is first of all themselves, forever imagining triumphs in a world that either ignores them or denies their presence altogether. But just as often enough to keep the game going one of them makes it and swings to the moon on a thread of dreams unwinding out of himself."

— Arthur Miller, *Timebends: A Life*, 1987

Acknowledgments

Thank you first and foremost to my supervisor, Dr. Abraham Punnen, for all his guidance and patience. I met Abraham at a very fortuitous time during my undergrad days at the University of New Brunswick, and I would not be where I am today, figuratively and literally speaking, without him.

I'd also like to thank everyone at SFU for their help, advice, and friendship. In particular, thank you to my colleagues Sophie Burrill, Karel Casteels, Arman Kaveh, Sara Taghipour, Brad Woods, Annie Zhang, and Hua Zheng.

Big thank yous to all my friends, especially Preet Bhogal, Ian Bryce, Nick Chernoff, Wallace Cheung, Craig Chouinard, Aneil Singh, Casey Smith, Carmel Teasdale, Brandon Yan, and Nathan Zeitner.

And last but certainly not least, thank you to my Dad, my brother Eddie, my sisters Janis and Kim, my step-mom Selina, cousin Jamie, his wife Amie, my aunt Carolyn, and the rest of my extended family. I could not have done this without your love and support.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The traveling salesman is, perhaps, one of the few professions celebrated in both popular culture as well as in math and computer science journals. Timothy Spears writes the traveling salesman is an "intriguing, almost mythic figure" that "established the foundations of 'scientific salesmanship' and helped develop modern consumer culture" [102]. One could describe the impact of the formal math problem we call the *traveling salesman problem* (TSP) with similar, glowing words. The question that has intrigued mathematicians for over sixty years: given a collection of cities, what is the shortest possible tour that visits each city exactly once? The TSP is quite difficult in a complexity sense, and is the motivation behind thousands of articles on topics such as integer programming, complexity theory, graph theory, heuristics, and approximation algorithms. In short, it has helped shape modern operations research, perhaps more than any other problem. But for all the attention these wayfaring peddlers get, few consider the plight of a salesman who is prone to car sickness.

To be more precise, we mean a salesman who is more concerned about minimizing the longest distance he travels between any pair of cities in a tour rather than minimizing the total distance. This 'minimax' problem is called the *bottleneck traveling salesman problem*, and is the problem of focus in this thesis. We also concern ourselves with the plight of a (hopefully innocent) fugitive, as well as one headache of aircraft maintenance persons.

But first we offer a short tour of the traveling salesman problem. Following this, we formally define the problems facing our queazy merchant, our hoodlum on the run, and our beleaguered plane mechanic.

1.1 The Traveling Salesman Problem

Given a number of cities to visit, a traveling salesman plans a route that takes him to each city exactly once and returns him home. It is obvious that some orders of touring the cities are shorter than others, and our salesman desires an order that minimizes the total distance he must travel. Although this problem is simple to describe, the *traveling salesman problem* (TSP) is a 'hard' combinatorial optimization problem. Its simple description, beguiling difficulty, and endless utility has made it one of the most well known problems in mathematics.

A Hamiltonian cycle is a cycle which visits each vertex in a graph exactly once. Formally defined, the TSP considers a directed or undirected graph $G = (V, E)$ with a cost (weight) c_{ij} perscribed for each edge $(i, j) \in E$. If $\Pi(G)$ is the set of all Hamiltonian cycles (tours) in G, then the TSP is to find a tour in $\Pi(G)$ whose sum edge cost is as small as possible, i.e.

$$\begin{aligned} &\text{minimize} \sum_{(i,j)\in H} c_{ij} \\ &\text{subject to } H \in \Pi(G). \end{aligned} \tag{1.1}$$

A solution to the TSP is of interest to more than just salesman and their kin that provide delivery and repair services. The TSP also finds applications in genome sequencing [10], printed circuit board manufacturing [72, 40, 71], aiming space telescopes [8], turbine engine maintenance [81], and sequencing pleasing iPod playlists [82]. Interested readers may consult the books of Applegate et al. [5] or Gutin and Punnen [41] for further details on these applications, among others.

1.1.1 Dantzig, Fulkerson, and Johnson's 42-City Problem

In their ground breaking 1954 paper, *Solution of a large-scale traveling-salesman problem* [26], Dantzig, Fulkerson and Johnson introduce many important ideas on solving the TSP that are still used today. To demonstrate their methods, they constructed a 49-city instance consisting of Washington D.C. and one city from each of the 48 mainland states (Alaska and Hawaii did not become states until 1959). Using an atlas, they constructed the cost between pairs of cities as the road distance between them, rounding to the nearest integer (1 unit $\approx$ 17 miles). As all the calculations were being done by hand, they decided to

remove 7 cities along the Washington D.C. and Boston corridor and solve a 42-city problem. Figure 1.1 gives a list of the 42 included cities and shows their locations on a map.

This problem is part of Reinelt's TSPLIB collection [95] under the name `dantzig42`. We will use this instance throughout this chapter to demonstrate the sorts of tours we will be finding. Figure 1.2 shows the optimal TSP tour on dantzig42 that Dantzig et al. found. Note that the edge between Washington DC and Boston, MA run through the seven cities that were removed, making this 42-city solution optimal for the original 49-city problem. However, this may not be true for the problems of interest to us, so we confine ourselves to just the listed 42-cities.

1.1.2 Complexity of the TSP

It is convenient to separate TSP instances into *symmetric* and *asymmetric* instances. Symmetric instances correspond to undirected graphs where $c_{ij} = c_{ji}$ for all $i, j \in V$. Asymmetric instances correspond to directed graphs (digraphs) where $c_{ij} \neq c_{ji}$ for some $i, j \in V$.

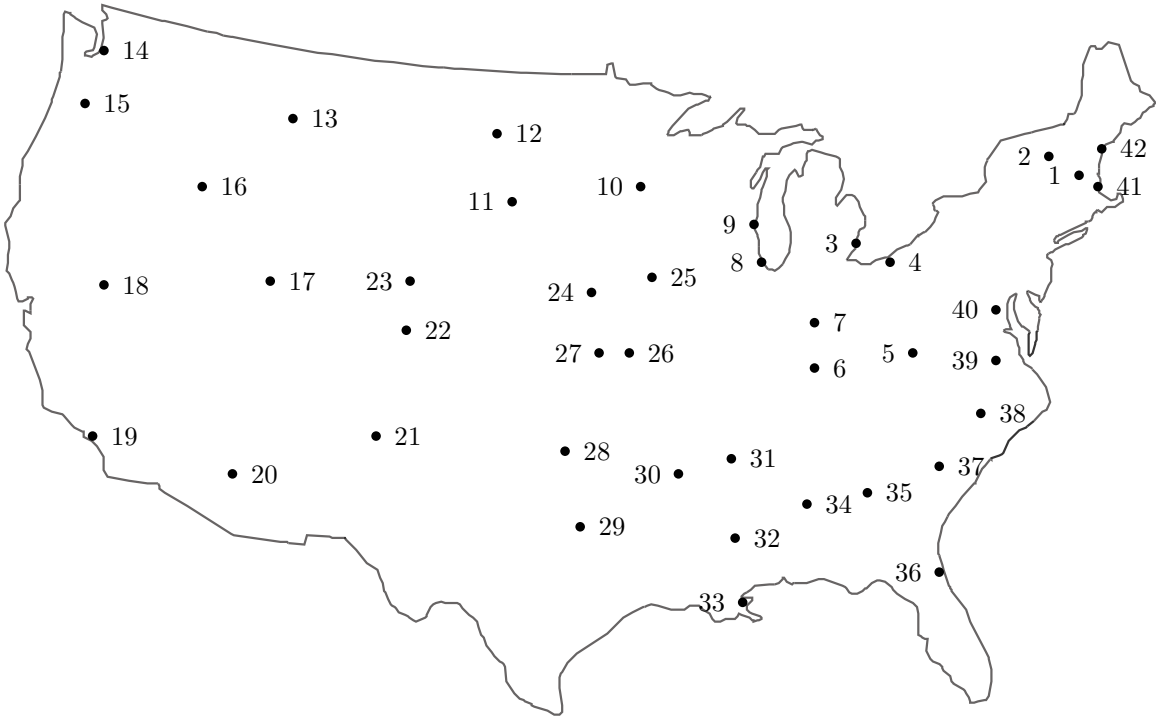
The TSP is well known to be NP-Hard [57] which means, unless the famous 'P=NP?' question is true, a 'good' algorithm for solving the TSP is unlikely to exist. By a 'good' algorithm, we mean one whose running-time is bounded by some polynomial factor with respect to the input size of the problem, such as the number of cities in a TSP instance. A full treatment of complexity theory is beyond the scope of this thesis, but it is interesting to note that, due to its popularity, the TSP is often used in proof attempts to show that P=NP [110]. Despite this negative result, there are some special cases where the TSP can be solved in polynomial time [53].

Instead of asking if a polynomial-time algorithm exists for the TSP, one could instead ask if a polynomial-time algorithm exists with a bounded performance guarantee.

Definition 1. Let α be a polynomial-time algorithm that returns a (possibly non-optimal) guess at the solution to an NP-Hard minimization problem, such as the TSP, and let $\epsilon \geq 1$ be a constant. Further, let z^* be the optimal objective value to some NP-Hard problem instance, and let z be the objective value returned by algorithm α. If

$$\frac{z}{z^*} \leq \epsilon$$

for any given problem instance, then we refer to α as an ϵ-approximation algorithm.



1.	Manchester, NH	15.	Portland, OR	29.	Dallas, TX
2.	Montpelier, VT	16.	Boise City, ID	30.	Little Rock, AR
3.	Detroit, MI	17.	Salt Lake City, UT	31.	Memphis, TN
4.	Cleveland, OH	18.	Carson City, NV	32.	Jackson, MS
5.	Charleston, WV	19.	Los Angeles, CA	33.	New Orleans, LA
6.	Louisville, KY	20.	Phoenix, AZ	34.	Birmingham, AL
7.	Indianapolis, IN	21.	Santa Fe, NM	35.	Atlanta, GA
8.	Chicago, IL	22.	Denver, CO	36.	Jacksonville, FL
9.	Milwaukee, WI	23.	Cheyenne, WY	37.	Columbia, SC
10.	Minneapolis, MN	24.	Omaha, NE	38.	Raleigh, NC
11.	Pierre, SD	25.	Des Moines, IA	39.	Richmond, VA
12.	Bismarck, ND	26.	Kansas City, MO	40.	Washington DC
13.	Helena, MT	27.	Topeka, KS	41.	Boston, MA
14.	Seattle, WA	28.	Oklahoma City, OK	42.	Portland, ME

Figure 1.1: 42-city instance `dantzig42`

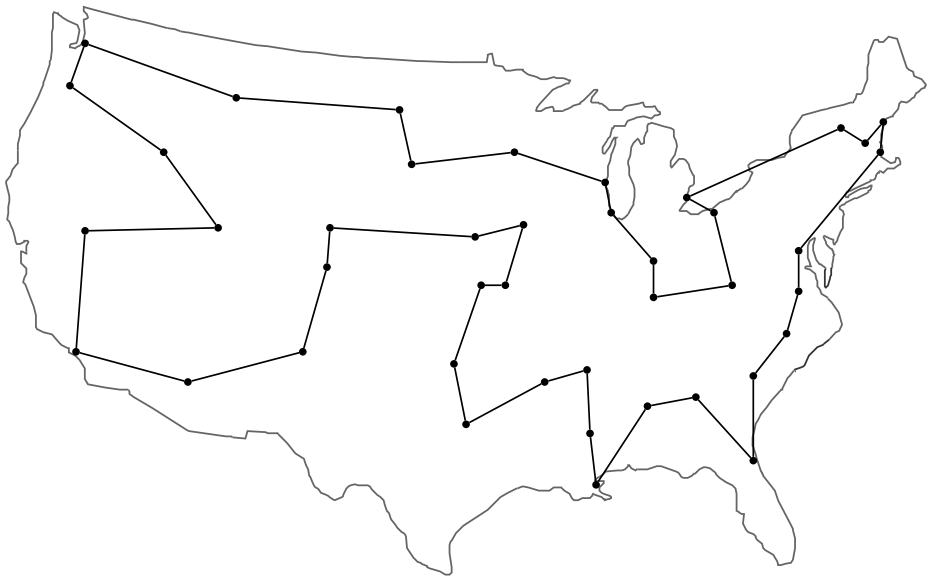


Figure 1.2: `dantzig42` TSP tour (tour length: 699; min cost: 3; max cost: 45)

For example, suppose we have an algorithm that returns TSP tours that are guaranteed to be at most twice as long as the optimal TSP tour. Such an algorithm would be a 2-approximation for the TSP.

Unfortunately, Sahni and Gonzalez [96] show the existence of any ϵ-approximation algorithm for the TSP implies that P=NP for any $\epsilon \geq 1$. This result suggests a polynomial-time approximation algorithm is unlikely to exist for all TSP instances, unless P=NP. Further, if P=NP, then optimal TSP tours could be found in polynomial time for any TSP instance, negating the need to study heuristics for the TSP.

Progress on the question of the approximability of the TSP can be made if we impose some conditions on the structure of the problem and its cost matrix. Assume our graph $G = (V, E)$ is complete (i.e. $(i, j) \in E$ for all $i, j \in V$) and that the cost matrix $C = (c_{ij})_{n \times n}$ is nonnegative and satisfies the triangle inequality, i.e.

$$c_{ij} \leq c_{ik} + c_{kj} \text{ for all } i, j, k \in V. \tag{1.2}$$

Table 1.1 details the progress made into ϵ-approximation algorithms for the TSP as well as the Max-TSP (where the sum of costs in a Hamiltonian cycle is *maximized*). Of particular note, Christofides [20] gives a $\frac{3}{2}$-approximation for symmetric TSP instances satisfying the triangle inequality, while Frieze et al. [33] give a $\log n$-approximation for asymmetric

instances. Despite these results being decades old, no one has asymptotically improved these performance guarantees (Kaplan et al. improve on Frieze et al.'s asymmetric TSP approximation to $0.841 \log n$ [56]). It is also not known if these are the best approximations possible.

A detailed treatment of solution methods for the TSP is unnecessary for understanding the algorithms we develop later, so we skip such a discussion. For the curious, effective strategies for solving the symmetric TSP is the subject of Applegate et al.'s book [5], and further theory can be found in the books of Lawler et al. [68] and Gutin and Punnen [41].

1.2 Problem Definitions

We now define four problems that are of focus in this thesis. Each problem instance consists of a directed or undirected graph $G = (V, E)$ with a cost c_{ij} prescribed for each edge $(i, j) \in E$. Finally, let $\Pi(G)$ denote the collection of (directed) Hamiltonian cycles in G.

1.2.1 The Bottleneck Traveling Salesman Problem

We recall the problem of our salesman who is unfortunately prone to car sickness. He would like to tour a set of cities in such a way that he spends as little time traveling between any pair of cities, therefore minimizing the amount of time driving for any one stretch of his journey. Formally defined, the BTSP is to find a Hamlitonian cycle in G whose largest edge cost is as small as possible, i.e.

$$\begin{aligned} &\text{minimize } \max\{c_{ij} : (i, j) \in H\} \\ &\text{subject to } H \in \Pi(G). \end{aligned} \tag{1.3}$$

Figure 1.3 gives an example of a BTSP tour on the `dantzig42` problem.

To the best of our knowledge, Gilmore and Gormory first formally defined the problem in 1964 [39]. Like its more famous other brother, the TSP, the BTSP is NP-Hard, so is in a complexity sense quite difficult to solve in general. This thesis focuses on practical solutions to the BTSP, involving study into lower bounds, approximation algorithms, heuristics, and a comprehensive computational study.

The *constrained bottleneck traveling salesman problem* (Constrained BSTP) places an additional restriction on the total weight of the tour. For each edge $(i, j) \in E$ prescribe a

Table 1.1: Progress on ϵ-appoximation algorithms for TSP and Max-TSP

Symmetric TSP with Triangle Inequality		
1976	Christofides [20]	$3/2 = 1.500$
Asymmetric TSP with Triangle Inequality		
1982	Frieze, Galbiati, Maffioli [33]	$\log n$
2003	Bläser [12]	$0.999 \log n$
2005	Kaplan, Lewenstein, Shafrir, Sviridenko [56]	$0.841 \log n$
Symmetric Max-TSP		
1984	Serdyukov [100]	$3/4 = 0.750$
1994	Kosaraju, Park, Stein [61]	$19/27 \approx 0.704$
2000	Hassin, Rubinstein [43]	$\epsilon < 25/33 \approx 0.758$
2005	Chen, Okamoto, Wang [19]	$61/81 \approx 0.753$
2008	Paluch, Mucha, Madry [77]	$7/9 \approx 0.778$
Symmetric Max-TSP with Triangle Inequality		
1985	Kostochka, Serdyukov [62]	$5/6 \approx 0.833$
2002	Hassin, Rubinstein [44]	$7/8 - O(n^{-1/2})$
2005	Chen, Nagoya [18]	$7/8 - O(n^{-1/3})$
2008	Kowalik, Mucha [64]	$7/8 = 0.875$
Asymmetric Max-TSP		
1979	Fisher, Nemhauser, Wolsey [31]	$1/2 = 0.500$
1994	Kosaraju, Park, Stein [61]	$38/63 \approx 0.603$
2003	Lewenstein, Sviridenko [70]	$5/8 = 0.625$
2004	Bläser [11]	$8/13 \approx 0.615$
2005	Kaplan, Lewenstein, Shafrir, Sviridenko [56]	$2/3 \approx 0.667$
Asymmetric Max-TSP with Triangle Inequality		
1985	Kostochka, Serdyukov [62]	$3/4 \approx 0.750$
2005	Kaplan, Lewenstein, Shafrir, Sviridenko [56]	$10/13 \approx 0.769$
2005	Chen, Nagoya [18]	$27/35 \approx 0.771$
2007	Kowalik, Mucha [63]	$35/44 \approx 0.795$
2009	Bläser, Ram, Sviridenko [13]	$31/40 = 0.775$

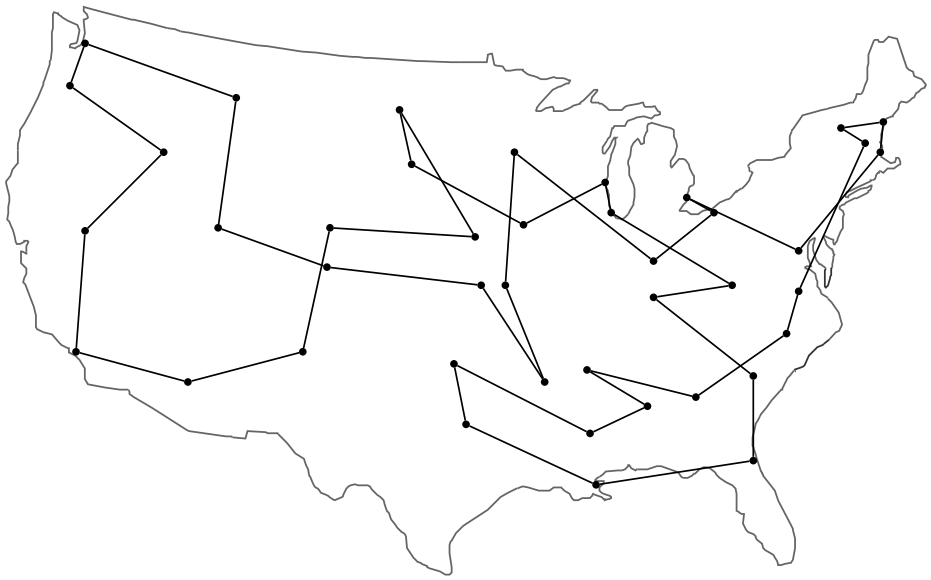


Figure 1.3: `dantzig42` BTSP tour (tour length: 964; min cost: 5; max cost: 35)

weight w_{ij}. The Constrained BTSP is to find a Hamiltonian cycle in G whose largest edge cost is as small as possible, and whose total weight is no more than a given value $\bar{w}$, i.e.

$$\begin{aligned} &\text{minimize } \max\{c_{ij} : (i,j) \in H\} \\ &\text{subject to } H \in \Pi(G), \\ &\qquad \sum_{(i,j)\in H} w_{ij} \leq \bar{w}. \end{aligned} \tag{1.4}$$

Optimal BTSP tours may be significantly expensive in terms of total cost, as the BTSP tour in Figure 1.3 demonstrates, so this additional constraint where $c_{ij} = w_{ij}$ for all $(i,j) \in E$ is useful in practical applications. Clearly, the problem is infeasible if $\bar{w}$ is of smaller value than the length of the optimal TSP tour using w_{ij} as the edge costs. Figure 1.4 gives a Constrained BTSP tour on `dantzig42` where the length is at most 716.

1.2.2 The Maximum Scatter Traveling Salesman Problem

Falsely accused of a crime you did not commit and facing the death penalty, you escape from the arms of the police and set out on an epic journey across the country to avoid capture. With your name splashed across TV, you know if you stay in the same general area for too long you will rouse the suspicions of the locals who will in turn report you to

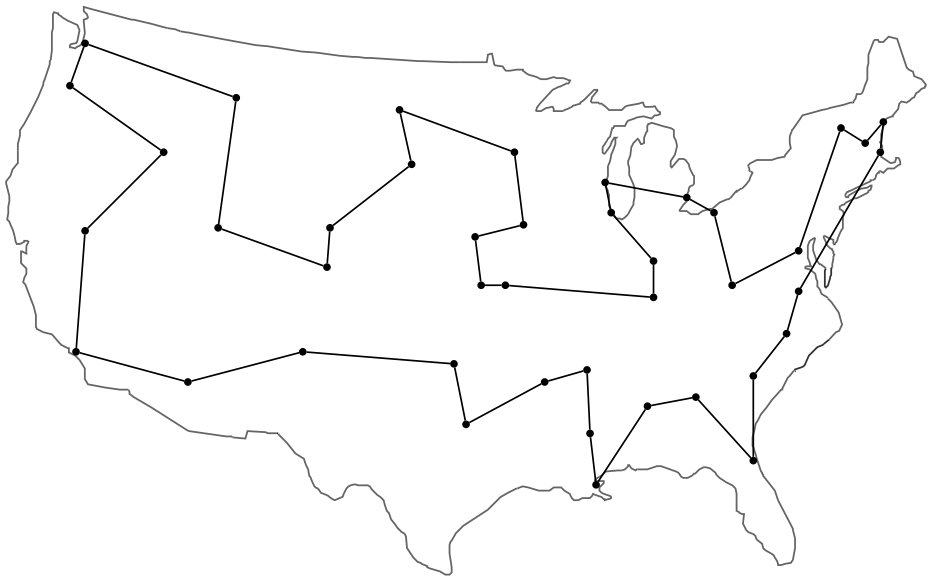


Figure 1.4: `dantzig42` constrained BTSP tour (tour length: 716; min cost: 3; max cost: 35)

the police. With the help of a circle of friends scattered across the country who believe in your innocence, you wish rotate from safe house to safe house in such a way that you are as far away from the previous safe house as possible.

In more precise terms, you are looking for the tour through your network of safe houses such that the smallest distance between consecutive locations is as large as possible. This model is known as the *maximum scatter traveling salesman problem* (MSTSP), and may be formally defined as follows:

$$\begin{aligned} &\text{maximize } \min\{c_{ij} : (i,j) \in H\} \\ &\text{subject to } H \in \Pi(G). \end{aligned} \tag{1.5}$$

The problem's namesake is due to Arkin et al. [6], and comes from an application of the problem in rivetting sheets of metal together for aircraft bodies. We discuss this application in the next section. Figure 1.5 presents the MSTSP tour on `dantzig42`.

This problem, in an optimization sense, is equivalent to the BTSP. Let

$$\bar{d} \geq \max\{c_{ij} : (i,j) \in E\} \tag{1.6}$$

Figure 1.5: `dantzig42` MSTSP tour (tour length: 3913; min cost: 73; max cost: 142)

and let

$$d_{ij} = \bar{d} - c_{ij} \text{ for all } (i,j) \in E. \tag{1.7}$$

Solving the BTSP with costs d_{ij} is equivalent to solving the MSTSP with costs c_{ij}, and vice-versa. While this transformation preserves the optimality of solutions, it does not preserve performance guarantees of approximation-algorithms (i.e. an ϵ-approximation algorithm for the BTSP does not imply an ϵ-approximation algorithm for the MSTSP merely by applying this transformation).

1.2.3 The Balanced Traveling Salesman Problem

So far we have discussed problems that involve finding a Hamiltonian cycle whose largest edge cost is minimized (the BTSP) and whose smallest edge cost is maximized (the MSTSP). Some models might instead demand that the difference between the largest edge cost and the smallest edge cost is minimized. For such models, we introduce the *balanced traveling salesman problem* (Balanced TSP), which is formally defined as

$$\begin{aligned} &\text{minimize } \max\{c_{ij} : (i,j) \in H\} - \min\{c_{ij} : (i,j) \in H\} \\ &\text{subject to } H \in \Pi(G). \end{aligned} \tag{1.8}$$

Figure 1.6: `dantzig42` Balanced TSP tour (tour length: 2131; min cost: 45; max cost: 58)

We explore an application of the Balanced TSP in the next section. It is clear the problem is at least as difficult as solving the BTSP and MSTSP, and so is NP-Hard. Figure 1.6 shows a Balanced TSP tour on the `dantzig42` problem, where the smallest cost and largest cost in the tour have a difference of 13 units.

1.3 Applications

To provide some motivation for studying these problems, we first discuss some applications of the BTSP found in literature. Afterwards, we present a new application applicable to the Balanced TSP model.

1.3.1 Applications Discussed in Literature

Assembly line sequencing (Garfinkel and Gilbert, 1978 [37])

Consider a circular assembly line with n jobs to be sequenced in any order. Let t_j be the time to complete a job j, and let t_{ij} be the setup time from job i to job j. If one wishes to sequence the jobs so as to minimize the cycle time (i.e. the amount of time necessary to move onto the next job in the line), one can solve the BTSP on cost matrix $C = (c_{ij})_{n \times n}$ where $c_{ij} = t_j + t_{ij}$.

Sequencing a one-state variable machine (Gilmore and Gomory, 1964 [39])

This is an application of the Gilmore-Gomory class of problems, for which the TSP is solvable in polynomial time (see Kabadi's book chapter [53] for a full discussion). Given a single machine, such as a furnace, one wishes to sequence n jobs. To start a job i, the machine must be in state A_i, and at completion the machine will be in state B_i. Let f be an integrable function that describes changing the state of the machine from u to v where $v \geq u$. Similarly, let g be an integrable function that describes changing the state of the machine from u to v where $u < v$. Further, let $f(x) + g(x) \geq 0$ for all x. The cost c_{ij} of performing a job j immediately after job i is given by

$$c_{ij} = \begin{cases} \displaystyle\int_{B_i}^{A_j} f(x)\,\mathrm{d}x & \text{if } A_j \leq B_i, \\ \\ \displaystyle\int_{A_j}^{B_i} g(x)\,\mathrm{d}x & \text{if } A_j < B_i. \end{cases} \tag{1.9}$$

In the example of a furnace, f may represent the cost of raising the temperature of the furnace from A_i to B_j, while g is the cost of lowering the temperature. Minimizing the largest change of state is an application for the BTSP model.

Reconstructing Sequential Orderings from Inaccurate Adjacency Information (Kao and Sanghi, 2006 [55])

Suppose n people stand in a circle in a clockwise orientation. Each person i reports their height A_i as well as estimates the height of the person clockwise adjacent to them B_i. Solely from this information one wishes to determine an ordering of the n people that minimizes the maximum difference between the reported heights A_i and the estimated heights B_i.

Kao and Sanghi discuss how this problem is used in biology for determining a protein's structure with spectroscopy data from a nuclear magnetic resonance (NMR) image. Proteins are made up of chains of amino acids (known as polypeptide chains). Knowing the sequence of amino acids is very useful to biologists. A NMR experiment records 'spectral peaks' which correspond to pairs of chemical shifts of atoms in adjacent amino acids. These chemical shifts serve to identify atoms in the protein, which in turn identify the amino acids. Relating to the 'people ordering' problem introduced above, the spectral peaks (which identify the amino

acids) are the people, and the chemical shifts associated with each spectral peak are the height estimates. While amino chains are linear and not circular, Kao and Sanghi show how to construct a linear ordering in polynomial time from a circular ordering.

Sequencing Rivet Operations (Arkin et al., 1999 [6])

On an aircraft body, two sheets of metal are joined together by rivets. The insertion of a rivet generates heat, and the sheet metal may deform if multiple rivets are placed consecutively in the same area. It is useful to find a sequence of rivet operations that maximize the distance between any consecutive pair of rivets so as to allow recently riveted areas to cool. This is an application for the MSTSP model.

Arkin et al. also discuss a similar application in medical imaging. A dynamic spatial reconstructor (DSR) is used in imaging physiological functions using pairs of radiation sources and sensors. Radiation sources are placed along the top half of a circular ring and corresponding sensors are placed directly opposite along the bottom half of the ring. However, an activated source scatters some amount of radiation to other nearby sources, so it is desirable to find a firing sequence of the radiation sources that maximizes the distance between consecutive sensors.

1.3.2 Nozzel Guide Vane Assembly in Gas Turbine Engines

The gas turbine engines that power military and commercial aircraft require extensive maintenance to ensure efficiency and reliability. One such stage of maintenance, the nozzle guide vane assembly, is formulated by Plante, Lowe and Chandrasekaran [81] as a product matrix TSP. We show how this problem may be formulated as both an asymmetric BTSP as well as an asymmetric Balanced TSP.

One of the main sections of the engine is the turbine, which consists of a series of stages configured with a nozzle and rotor pairing. The nozzle assembly consists of a sequence of nozzle guide vanes (hereafter referred to as vanes) affixed along the inner circumference of the nozzle diaphragm. Typically, there are 46 to 100 vanes in a given nozzle assembly. The nozzle assembly accelerates, deflects, and distributes the gases that drives the engine. An engine is more efficient if the distribution of gas flow throughout the nozzle assembly is 'uniform'.

The vanes experience wear and tear due to the high temperatures and velocities of the

gases pushed through the nozzle assembly. After disassembling the engine, a mechanic may choose to refurbish or discard some vanes, adding new or refurbished vanes to the collection as necessary. The mechanic then sequences the collection of vanes so as to obtain uniformity of flow areas between adjacent vanes. For a more detailed description of the problem, we refer to [49] and [81].

Let n be the number of vanes required for the nozzle assembly. Each vane has a convex and concave side which is measured against a master vane to derive two contribution values, A_i and B_i, for each vane i. Figure 1.7 illustrates this measurement. A positive value A_i or B_i indicates the vane contributes a larger area for gas flow than the master vane; a negative value indicates a small area contribution compared to the master vane. The measurement $(A_i + B_j)$ indicates the area between vanes i and j, with vane j placed clockwise-adjacent to vane i. In general $(A_i + B_j) \neq (A_j + B_i)$.

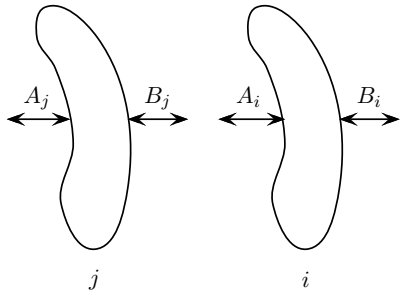


Figure 1.7: Area between vanes i and j

For a set of vanes V, $n = |V|$, we can construct a complete graph $G = (V, V \times V)$ where Hamiltonian cycles in G correspond to vane sequences in the nozzle assembly. For a Hamiltonian cycle H, let

$$x_{ij} = \begin{cases} 1 & \text{if } (i,j) \in H \\ 0 & \text{otherwise} \end{cases}$$

which corresponds to placing vane j clockwise-adjacent to vane i in the nozzle assembly. Let the total nozzle flow area be denoted by

$$T = \sum_{i=1}^{n} \sum_{j=1}^{n} (A_i + B_j) x_{ij} = \sum_{i=1}^{n} (A_i + B_i) \tag{1.10}$$

(each vane is included exactly once, so its value is constant). Consider its mean value

$$\bar{d} = \frac{\sum_{i=1}^{n}(A_i + B_i)}{n}.$$

An assembly is perfectly uniform if $(A_i + B_j) = \bar{d}$ whenever $x_{ij} = 1$. However, it is unlikely such an assembly is possible due to imprecise manufacturing tolerances. Instead, one can determine vane placement by attempting to maximize 'uniformity' by subjective criteria.

Construct cost matrix $C = (c_{ij})_{n\times n}$ where

$$c_{ij} = (\bar{d} - (A_i + B_j))^2. \tag{1.11}$$

The entries of cost matrix C represent the square deviations of $(A_i + B_j)$ from the mean nozzle flow area $\bar{d}$. We note that in general $(A_i + B_j) \neq (A_j + B_i)$, so C is an asymmetric matrix.

What constitutes a 'uniform' assembly is somewhat subjective. Plante et al.'s approach is to minimize the sum of square deviations of $(A_i + B_j)$ from $\bar{d}$, e.g.

$$\begin{aligned} &\text{minimize} \sum_{(i,j)\in H} (\bar{d} - (A_i + B_j))^2 \\ &\text{subject to } H \in \Pi(G). \end{aligned} \tag{1.12}$$

An alternative approach is to use the BTSP model, which minimizes the largest squared deviation from the mean nozzle flow area, e.g.

$$\begin{aligned} &\text{minimize } \max\{(\bar{d} - (A_i + B_j))^2 : (i,j) \in H\} \\ &\text{subject to } H \in \Pi(G). \end{aligned} \tag{1.13}$$

The Balanced TSP on C also provides a suitable model, where the difference between the largest and smallest squared deviation from the mean is minimized:

$$\begin{aligned} &\text{minimize } U - L \\ &\text{subject to } U \geq \max\{(\bar{d} - (A_i + B_j))^2 : (i,j) \in H\}, \\ &\qquad\qquad L \leq \min\{(\bar{d} - (A_i + B_j))^2 : (i,j) \in H\}, \\ &\qquad\qquad H \in \Pi(G). \end{aligned} \tag{1.14}$$

Note this Balanced TSP problem formulation is equivalent to that of Equation (1.8).

1.4 Problem Complexity

It is well known that the BTSP is NP-Hard [54], easily verified by a reduction from the Hamiltonian Cycle problem (see Garey and Johnson's classic book [36] on NP-Completeness theory for more information). Unless P=NP, these results indicate a polynomial time algorithm for solving the general BTSP is unlikely to exist. Clearly, the constrained version, the Constrained BTSP, is NP-Hard as well. Arkin et al. [6] show the MSTSP is also NP-Hard. Finally, it is trivial to show the Balanced TSP is NP-Hard via a reduction from the BTSP or the MSTSP.

Certain problem and cost matrix structures may lend themselves to specialized polynomial time solutions to the BTSP. The Gilmore-Gomory class of problems [39] introduced in the previous section are one example. When a cost matrix is defined by Equation 1.9 and

$$f(x) \geq 0 \text{ and } g(x) = 0 \tag{1.15}$$

(or vice versa) then Gilmore and Gomory show that the BTSP may be solved in polynomial time. Various extensions to the Gilmore-Gomory class of problems also exist [54, 53, 60, 106, 108]. The BTSP on a Halin graph (a planar graph constructed by connecting the leaf nodes of a finite tree with no vertices of degree 2 with a cycle) is another polynomially solvable instance [80]. For a thorough discussion on solvable cases of the BTSP, we refer to the book chapter of Kabadi and Punnen [54].

The approximability of the general BTSP or MSTSP is just as dire as that of the general TSP. It can be shown that, unless P=NP, no polynomial time ϵ-approximation algorithm exists for the BTSP or the MSTSP for any $1 \leq \epsilon \leq \infty$ [54].

Instead, as with the TSP, one may look at developing approximation algorithms for problem instances that satisfy certain conditions. Kao and Sanghi show that if a cost matrix satisfies Equation 1.9 (the Gilmore-Gomory cost matrix) for a general $f(x), g(x) > 0$ that an $O(n \log n)$ time $(2 + \gamma)$-approximation algorithm exists where $\gamma \geq \frac{f(x)}{g(x)} \geq \frac{1}{\gamma}$ [55]. When $f(x) = g(x) = 1$, as in the sequential orderings application in the previous section, their algorithm yields a 3-approximation.

We discussed approximation algorithms for the TSP on problem instances that satisfy

the triangle inequality (1.2). A generalization of the triangle inequality is the τ-triangle inequality

$$c_{ij} \leq \tau(c_{ik} + c_{kj}) \text{ for all } i, j, k \in V \tag{1.16}$$

for some $\tau \geq \frac{1}{2}$. When $\tau = \frac{1}{2}$, it forces all c_{ij} to be the same cost, while when $\tau = 1$ it reduces to the standard triangle inequality. It is known when the edge costs are symmetric and satisfy the τ-triangle inequality that a polynomial 2τ-approximation exists for both the BTSP, due to Rardin and Parker [79] for the case of $\tau = 1$ (also proved independently by Doroshko and Sarvanov [27]), and the MSTSP, due to Arkin et al. [6] when $\tau = 1$ (see Kabadi and Punnen's book chapter for a proof of the general case when τ is arbitrary [54]). Further, unless P=NP, a 2τ-approximation is the best result possible for the symmetric case of the BTSP or MSTSP [54]. In Chapter 3 we discuss an approximation algorithm for the asymmetric version of the BTSP when the edge costs satisfy the τ-triangle inequality.

1.5 Contributions of this Thesis

In this thesis we develop heuristic algorithms for solving the BTSP, MSTSP, and Balanced TSP. These algorithms can be easily adapted to solve their constrained versions where the total tour length must be less than a given parameter. In Chapter 2, extensive computational results are presented for the symmetric BTSP on problems of sizes up to 31,623 vertices for all available TSP benchmark problems from Reinalt's TSPLIB library [95], random instances due to Johnson and McGeoch [50, 51], real-world National TSP instances [24], and instances based upon real-world integrated circuit board problems (VLSI instances) [22]. Our heuristic algorithm uses randomization in a controlled way to guide the heuristic search, often yielding provably optimal solutions.

The asymmetric BTSP is the topic of Chapter 3. We define new lower bounds for this problem and establish a comparison between several well-known lower bounds. A new theorem is also provided which can be used in a recursive way to improve most lower bounding schemes. Extensive computational results are presented using our heuristic algorithm that produced provably optimal solutions in many instances. We establish a new asymmetric BTSP approximation algorithm with worst-case performance ratio of $\lceil \frac{n}{2} \rceil$ when the problem costs satisfy the triangle inequality. Further, we show that any asymmetric BTSP instance on n vertices can be represented as a symmetric BTSP on $3n$ vertices. We also consider the

MSTSP and present extensive computational results for the asymmetric version.

In the final chapter, we consider the Balanced TSP. This problem has not been studied in literature so far, but has useful applications such as in the nozzle guide vane problem introduced earlier in this chapter. We develop four heuristic for solving the Balanced TSP, two of which are shown to be effective. Extensive computational results are given using these two promising heuristic. As with the BTSP, we produce provably optimal solutions for many of these problems.

Chapter 2

The Symmetric Bottleneck TSP

Let $G = (V, E)$ be a graph on n nodes. For each edge $(i, j) \in E$, a cost c_{ij} is prescribed. If $\Pi(G)$ is the set of all Hamiltonian cycles (tours) in G, then the *bottleneck traveling salesman problem* (BTSP) is to find a Hamiltonian cycle in G whose largest edge cost is as small as possible, i.e.

$$\begin{aligned} &\text{minimize } \max\{c_{ij} : (i, j) \in H\} \\ &\text{subject to } H \in \Pi(G). \end{aligned} \tag{2.1}$$

The BTSP is the correct model to use when one wants to minimize the largest distance travelled between any pair of stops in a tour. In Chapter 1 we invoked the image of a traveling salesman who gets car sick easily, and so would benefit from a tour that kept his travel time between stops to an absolute minimum. Many TSP models have a natural interpretation in the BTSP context, and we refer the reader to Chapter 1 for an overview of some interesting applications.

Assumptions

We assume the graph is *undirected* and $c_{ij} = c_{ji}$ for all $(i, j) \in E$. We refer to this problem as the *symmetric BTSP*. When the graph is directed, that is to say $c_{ij} \neq c_{ji}$ for some $(i, j) \in E$, we refer to the problem as the *asymmetric BTSP*. The asymmetric BTSP is the topic of Chapter 3.

Without loss of generality we assume G is a complete graph. If for any $i, j \in V$ it is true that $(i, j) \notin E$ we can let $c_{ij} = \infty$ (i.e. a sufficiently large cost) so as to effectively

exclude it from the solution. We see the complete cost matrix $C = (c_{ij})_{n\times n}$ encodes all the necessary information for formulating the BTSP problem.

We also assume without loss of generality that c_{ij} is nonnegative. It is easy to prove that the numerical edge costs are unimportant and that only the ordering of the edge costs matters [54]. If our original problem has negative edge costs we can simply create a cost matrix $D = (d_{ij})_{n\times n}$ where $d_{ij} = c_{ij} + M$ and M is a sufficiently large positive constant such that $c_{ij} + M \geq 0$ for all i, j. Solving the BTSP with cost matrix D is equivalent to solving the BTSP with cost matrix C.

Finally, we assume all costs c_{ij} are integer, as is standard for many TSP solvers. As noted above, only the ordering of the edge costs matter, so an equivalent (in terms of BTSP optimality) integer matrix D can be produced for any real number matrix C. However, our approach when dealing with real numbers is generally to round to the nearest integer as defined by the TSPLIB file format specification [95]. It should be noted that this is not an assumption of our heuristic algorithm, but rather of the problems in our test set.

Previous work

To the best of our knowledge, Gilmore and Gomory introduced the BTSP in 1964 [39]. Their treatment of the problem was restricted to what is commonly referred to as the Gilmore-Gomory class of problems (see Equation (1.9) in Chapter 1 for further details). The BTSP with a general cost matrix was first studied by Gabovich, Ciz, and Jalas in 1971 [34].

The Gilmore-Gomory class of problems has been a popular starting point for extensions to special cases of the BTSP, many of which can be solved in polynomial time [53, 54, 60, 73, 106, 108]. Phillips, Punnen, and Kabadi also showed the BTSP on Halin graphs can be solved in polynomial time [80]. For a state-of-the-art discussion on polynomially solvable cases of the BTSP we refer to the book chapter by Kabadi and Punnen [54].

Algorithms and computational study on a general cost matrix has been limited. In 1978, Garfinkel and Gilbert discussed a branch-and-bound based exact algorithm to solve the BTSP, and reported computational results with a construction heuristic on randomly generated problems of sizes up to 100 vertices [37]. Carpento et al. reported experimental results with a branch-and-bound algorithm on problems of sizes up to 200 vertices in 1984 [17]. Sergeev proposed a dynamic programming approach in 1996, but gave no computational results [101]. Finally, Ramakrishnan et al. reported experimental results with a threshold heuristic on 72 symmetric TSPLIB problems of size up to 783 vertices [91] in

2000.

Problem complexity

The BTSP is well known to be NP-hard, but, like the TSP, several special cases of the problem can be solved to optimality in polynomial time [54].

Unless P=NP, no polynomially-solvable ϵ-approximation algorithm exists for the BTSP for any $\epsilon > 1$ [27, 79, 97]. However, polynomial-time ϵ-approximation algorithms exist in special cases [7, 27, 48, 54, 79]. One such special case is when a symmetric cost matrix satisfies the triangle inequality, which is to say

$$c_{ij} \leq c_{ik} + c_{kj} \text{ for all } i, j, k. \tag{2.2}$$

In such a case a 2-approximation exists. This result was obtained independently by Parker and Rardin [79] as well as by Doroshko and Sarvanov [27]. Further, it can be shown that this is the best possible approximation for problems in this class. In Chapter 3 we develop an ϵ-approximation algorithm for the asymmetric BTSP, including a generalization of this 2-approximation, so we defer further discussion of this approximation algorithm until then. Similar results on other bottleneck problems, including the BTSP, are discussed by Hochbaum and Shmoys [48].

It is well known that the Euclidean TSP can be solved by a fully polynomial approximation scheme [7]. However, it is NP-hard to obtain an ϵ-approximation algorithm for the BTSP for any $\epsilon < 2$, even with a Euclidean cost matrix [97]. In this sense the Euclidean version of the BTSP is harder than its TSP counterpart.

For a complete discussion on the complexity of the BTSP, we refer the reader to the book chapter by Kabadi and Punnen [54].

Our contributions

We develop several heuristics for the BTSP, one of which could be viewed as a generalization of Ramakrishnan et al.'s threshold heuristic [91]. Extensive computational results are presented for problems of up to 31,623 vertices. Our main heuristic algorithm produced optimal solutions, many with proof of optimality, for almost all problems considered within a very reasonable computational time.

Our set of benchmark test instances includes problems from Reinalt's TSPLIB library [95], random instances due to Johnson and McGeoch [50, 51], real-world National TSP instances [24], and instances based upon real-world integrated circuit board problems (VLSI instances) [22]. Further, we created specially structured problems that are designed to be hard for our heuristic.

Please note that results on two BTSP lower bound algorithms (the 2-max bound and the bottleneck biconnected spanning subgraph problem) originally appeared in the author's undergraduate thesis [65].

In Section 2.1 we discuss lower bounds for the BTSP, followed by a heuristic algorithm in Section 2.2. Extensive computational results are given in Section 2.3.

2.1 Lower Bounds for the Symmetric BTSP

Suppose for some cost matrix C we know the optimal BTSP objective value can be no less than some value z. If we can find a tour H where

$$\max\{c_{ij} : (i,j) \in H\} = z$$

then H is an optimal BTSP tour for cost matrix C. In this section we discuss two polynomial schemes to compute lower bounds on the optimal BTSP objective value. Both these bounds are well known, but we supplement them with some theoretical analysis.

2.1.1 2-Max Bound (2MB)

Although almost naïvely simple, we will see the 2-max bound (2MB), described in [54, 91], is a surprisingly strong lower bound for the BTSP through computational study. For each node i find the smallest and second-smallest edge cost incident on i (both may be the same cost). Let $\zeta(i)$ be the second-smallest edge cost incident on i, counting multiplicity (i.e. if the smallest cost is not unique, the second smallest cost is the same as the smallest cost). Clearly, any BTSP tour will in the best case use the edges of smallest and second-smallest cost incident on i, so

$$\max\{\zeta(i) : i \in V\} \tag{2.3}$$

is a lower bound on the BTSP objective value. The 2MB can clearly be computed in $O(m)$-time.

2.1.2 Bottleneck Biconnected Spanning Subgraph Problem (BBSSP) Bound

A *biconnected graph* (also known as a 2-connected graph) is a connected graph where the removal of any vertex does not disconnect the graph. Alternatively, one may consider it as a graph with two vertex-disjoint paths between any pair of vertices. As any Hamiltonian cycle is biconnected (the removal of any vertex of a Hamiltonian cycle results in a path connecting the remaining vertices), one necessary condition for Hamiltonicity is for the graph to be biconnected.

The bottleneck biconnected spanning subgraph problem (BBSSP) seeks to find a biconnected spanning subgraph $\bar{G}$ of G whose largest edge cost is as small as possible, i.e.

$$\begin{aligned} &\text{minimize } \max\{c_{i,j} : (i,j) \in \bar{G}\} \\ &\text{subject to } \bar{G} \text{ is a spanning subgraph of } G, \\ &\bar{G} \text{ is biconnected.} \end{aligned} \tag{2.4}$$

The objective value of the BBSSP is clearly a lower bound on the BTSP objective value.

Testing a graph for biconnectivity can be done in $O(m)$-time using Tarjan's depth-first search algorithm [103]. Solving Equation (2.4) for a graph G with cost matrix C can therefore be done in $O(m \log m)$-time by performing a binary search over distinct costs of C as follows.

1. Let $z_1 < z_2 < \cdots < z_k$ be the distinct entries of C sorted in non-increasing order.
2. Let $l \leftarrow 1$, $u \leftarrow k$.
3. Let $\delta \leftarrow (u - l)/2 + l$.
4. Let $\bar{G} = (V, \bar{E})$ where $\bar{E} = \{(i,j) \in E : c_{ij} \leq z_\delta\}$.
5. If $\bar{G}$ is biconnected, set $u = \delta$; otherwise, set $l = \delta + 1$.
6. If $l = u$ then return z_l; otherwise, go to step 3.

This simple implementation, as discussed by Parker and Rardin [79], is the one we use in our computational experiments. However, there exists an $O(m + n \log n)$-time algorithm due to Punnen and Nair [86] and an $O(m)$-time algorithm due to Manku [73] for solving the BBSSP, albeit with more difficult implementations.

Theorem 1. *The BBSSP objective value will always be greater than or equal to the 2MB objective value for any given instance $G = (V, E)$ with cost matrix C.*

Proof. By definition, any biconnected graph has at least two edges incident on each vertex. In the worst-case, these two edges will have costs equal to the smallest and second-smallest edge costs incident on each vertex (i.e. the two edges the 2MB identifies). □

We say the BBSSP bound *dominates* the 2MB because no instance exists where the 2MB produces a superior lower bound on the BTSP objective value compared to the BBSSP.

2.2 Heuristic Algorithms for the Symmetric BTSP

We now present our heuristic algorithms for solving the symmetric BTSP on a given graph G with cost matrix C. We assume that we have calculated a lower bound L on the optimal BTSP objective value using the BBSSP lower bound presented in Section 2.1.

2.2.1 Feasibility Oracle

The key part of our heuristic algorithm for the BTSP relies on answering the following question: *for an integer δ, does there exist a Hamiltonian cycle H in G such that* $\max\{c_{ij} : (i, j) \in H\} \leq \delta$? Let us call this the *$\delta$-feasibility question*.

If we could correctly answer this question for any δ, then we could solve the BTSP by performing a binary search over the distinct costs of C, much like how we solved the BBSSP. Simply arrange the distinct costs of C in ascending order $z_1 < z_2 < \ldots z_k$ and choose the smallest index $i \in \{1, 2, \ldots, k\}$ such that the z_i-feasibility question has a 'yes' answer. This is the well-known threshold approach for bottleneck problems specialized for the BTSP [29].

Determing if a graph is Hamiltonian or not is a NP-Hard problem, so instead we examine how to answer the δ-feasibility question in a heuristic manner. We start by considering how the TSP, an equivalently hard problem, can help answer our δ-feasibility question. Let $C^\delta = (c^\delta_{ij})_{n \times n}$ be an $n \times n$ cost matrix where

$$c^\delta_{ij} = \begin{cases} 0 & \text{if } c_{ij} \leq \delta, \\ c_{ij} & \text{otherwise.} \end{cases} \tag{2.5}$$

Suppose the optimal TSP tour on cost matrix C^δ is of zero-length. This means there exists a tour in the original cost matrix C whose largest cost is at most δ, i.e. a 'yes' answer to the δ-feasibility question. Likewise, it is clear if the optimal TSP tour on cost matrix C^δ is of nonzero-length that we consequently have a 'no' answer to the δ-feasibility question.

As already mentioned, the TSP is NP-Hard like the Hamiltonian cycle problem, so in theory it is no easier a question to solve. Although as of 2009 researchers have solved TSP instances of 85,900 vertices to optimality [23], solving the BTSP with a threshold approach would require in the worst case solving $O(\log m)$ TSPs. This is likely computationally infeasible for large problems, so a better approach is required.

One approach is to use a TSP heuristic instead of an exact algorithm. Excellent heuristic algorithms exist to find optimal or near optimal TSP solutions. These heuristics have played an important part in tackling large TSP instances as they provide good quality initial feasible solutions from which to start a search from. As TSP instances are generally proven optimal through branch-and-cut procedures, starting with a near optimal solution can drastically reduce the size of the search tree necessary to prove optimality. For a state-of-the-art discussion of TSP heuristics, interested readers should consult the book chapter by Johnson and McGeoch [51].

Suppose we use a TSP heuristic α instead of an exact TSP algorithm to solve the δ-feasibility question. If a TSP heuristic α can find a zero-length tour in C^δ, Equation (2.5), we can provably answer 'yes' to our question. The opposite is not true however. If our TSP heuristic α finds a nonzero-length tour in C^δ, we cannot conclude a 'no' answer as it is possible our heuristic was unable to find the optimal zero-length solution.

Let us assume our TSP heuristic is of a 'local search' type, such as a simulated annealing search, a tabu search, or a genetic algorithm. As per their name, local search heuristics can become trapped at non-optimal local minimum if there is no better solution to move towards in the current 'neighbourhood'. A popular way to deal with the problem of local minima is to perform a random restart: randomly generate a new solution and restart the local search from that point.

The solution space for the δ-feasibility problem is unique in that the non-zero costs in C^δ can take on any positive, nonzero value without affecting where optimal solutions exist (i.e. any zero-length tours). Changing these values might allow our TSP heuristic to avoid getting consistently trapped at a local minima. One way to do this is to add random values to the non-zero values of C^δ. We refer to introducing randomness to C^δ as a 'shake'

operation. 'Shaking' the cost matrix may help the TSP heuristic to discover a 'yes' answer to the δ-feasibility question.

Below we describe three cost matrices that perform shake operations.

Uncontrolled Shake Operation

For each $(i, j) \in E$ generate a random, positive integer r_{ij} uniformly on some interval $[a, b]$. Define an *uncontrolled shake operation* as the cost matrix $C_0^\delta = (c_{ij}^{\delta,0})_{n\times n}$ where

$$c_{ij}^{\delta,0} = \begin{cases} 0 & \text{if } c_{ij} \leq \delta, \\ r_{ij} & \text{otherwise.} \end{cases} \tag{2.6}$$

Type I Controlled Shake Operation

Let $z_1 < z_2 < \ldots z_k$ be the list of distinct costs in C arranged in ascending order. Generate a random, positive integer r_i for $i = 1, \ldots, k$ uniformly on some interval $[a, b]$ where $r_1 < r_2 < \cdots < r_k$. Define an *'Type I' shake operation* as the cost matrix $C_1^\delta = (c_{ij}^{\delta,1})_{n\times n}$ where

$$c_{ij}^{\delta,1} = \begin{cases} 0 & \text{if } c_{ij} \leq \delta, \\ z_k + r_k & \text{otherwise, where } z_k = c_{ij}. \end{cases} \tag{2.7}$$

We can see the difference between consecutive non-zero values of entries of C_1^δ is larger compared to corresponding values of C^δ, thus encouraging a TSP heuristic to pick smaller non-zero values to produce a better quality solution (in terms of BTSP objective value).

Type II Controlled Shake Operation

Let $z_1 < z_2 < \ldots z_k$ be a list of distinct costs in C arranged in ascending order. Define a random, positive integer r_i in the interval $[r_{i-1} + \theta, r_{i-1} + \tau]$, where $r_0 = 0$ and θ and τ are prescribed parameters, $\theta < \tau$. Further, define U to be a prescribed integer where $U > \delta$ and an integer M where $M \geq z_k$. Define an *'Type II' shake operation* as the cost matrix

$C_2^\delta = (c_{ij}^{\delta,2})_{n\times n}$ where

$$c_{ij}^{\delta,2} = \begin{cases} 0 & \text{if } c_{ij} \le \delta, \\ z_k + r_k & \text{if } c_{ij} \le U, \text{ and where } z_k = c_{ij}, \\ M & \text{otherwise.} \end{cases} \tag{2.8}$$

Cost matrix C_2^δ has similar properties as C_1^δ but with more control over the random numbers. Forcing costs greater than U to be a very large number might encourage a TSP heuristic to focus on finding tours with edge costs closer to δ.

It is easy to see if a tour of zero length is found by a TSP heuristic with either C_0^δ, C_1^δ, or C_2^δ that it corresponds to a 'yes' answer to the δ-feasibility problem. Further, if a nonzero-length tour is found by any of the shake cost matrices, one can make further attempts with different random integers. If after a prescribed number of attempts with any combination of these shake matrices and C^δ we cannot find a TSP tour of zero length, then we conclude with high probability that the answer to the δ-feasibility question is 'no'.

When answering the δ-feasibility question we can use any combination of cost matrices C^δ, C_0^δ, C_1^δ, or C_2^δ in our search for a tour using only edge costs less than or equal to δ. Let p, q, r, and s be four nonnegative integers that correspond to the number of attempts to answer the δ-feasibility question with cost matrices C^δ, C_0^δ, C_1^δ, and C_2^δ, respectively. Algorithm 2.1, *IsFeasible*, outlines our framework for solving the δ-feasibility question.

Let us now examine how we can use the *IsFeasible* method to extract a good quality heuristic solution for the BTSP.

2.2.2 Threshold Heuristics for the Symmetric BTSP

Let L be any lower bound on the optimal objective value for the BTSP, such as by the two schemes discussed in Section 2.1. In concert with the *IsFeasible* method, we present the *Simple Threshold Heuristic* in Algorithm 2.2. The choice of values for p, q, r, and s will be determined through computational experimentation. We use Concorde's implementation of the Lin-Kernighan algorithm as our TSP heuristic α [4]. The Lin-Kernighan algorithm is a local-search heuristic that experimentally has an average computational complexity of $O(n^{2.2})$ [47, 51].

For clarity, we do not include parameters for the shake matrices (a, b, θ, τ, and M) in

Algorithm 2.1: IsFeasible$(n, C, \delta, \alpha, p, q, r, s)$

Input: A problem on n nodes with cost matrix C, integer δ, TSP solver/heuristic α, and integers p, q, r, and s which represent the number of iterations with cost matrices C^δ, C_0^δ, C_1^δ, and C_2^δ, respectively.

Output: The 3-tuple $(feasible, tour, max\_cost)$ where $feasible$ is a Boolean value that indicates if a Hamiltonian cycle was found using only costs less than or equal to δ, $tour$ is the feasible/best tour found, and max_cost is largest cost in $tour$.

$minmax\_cost \leftarrow \infty$;
$best\_tour \leftarrow \emptyset$;
Let $z_1 < z_2 < \cdots < z_k$ be the distinct costs in C arranged in ascending order;
for $i = 1...p$ **do**
  Let C^δ be constructed as per Equation (2.5);
  $(length, tour) \leftarrow \alpha(n, C^\delta)$; `/* solve TSP on` C^δ `with` α `*/`
  $max\_cost \leftarrow \max\{c_{ij} : (i,j) \in tour\}$;
  if $length = 0$ **then**
    return $(TRUE, tour, max\_cost)$;
  else
    if $max\_cost < minmax\_cost$ **then**
      $minmax\_cost \leftarrow max\_cost$;
      $best\_tour \leftarrow tour$;
    end
  end
end
for $i = 1...q$ **do**
  Let $r_1, r_2, ..., r_k$ be a list of random integers where $1 \leq r_1 < r_2 < ... < r_k \leq n^2$;
  Let C_0^δ be constructed as per Equation (2.6);
  `/* Repeat check for a tour of zero-length as above for` C_0^δ`, ending the search if a zero length tour can be found; otherwise, save found tour if it is better in terms of BTSP objective value */`
end
for $i = 1...r$ **do**
  `/* Repeat as above for` C_1^δ `constructed as per Equation (2.7) */`
end
for $i = 1...s$ **do**
  `/* Repeat as above for` C_2^δ `constructed as per Equation (2.9) */`
end
return $(FALSE, best\_tour, minmax\_cost)$;

Algorithm 2.2: SimpleThreshold$(n, C, L, \alpha, p, q, r, s)$

Input: A problem on n nodes with cost matrix C, a lower bound L, TSP solver/heuristic α, and integers p, q, r, and s which represent the number of iterations with cost matrices C^δ, C_0^δ, C_1^δ, and C_2^δ, respectively, in the *IsFeasible* method.

Output: The (optimal/heuristic conclusion) on the BTSP objective value and tour.

$(feasible, best\_tour, max\_cost) \leftarrow$ IsFeasible$(n, C, L, \alpha, p, q, r, s)$;
return $(max\_cost, best\_tour)$;

the algorithm description. We will see $a = 1$, $b = n^2$, $\theta = 5$, $\tau = 50$ are good choices. No claim is made that these are the best parameter values. One might find for particularly difficult problem instances that different values for these parameters may produce improved solutions.

We can also use the *IsFeasible* method in a binary search over distinct costs of C. Let $z_1 < z_2 < \cdots < z_k$ be the distinct costs of C arranged in ascending order. Given a lower bound L on the BTSP objective value, let $z_l = L$. Perform one iteration of the *IsFeasible* method to get a heuristically good (perhaps optimal) tour whose largest cost is z_u. If $z_l = z_u$, then the tour found is an optimal BTSP tour; otherwise, the optimal solution lies in the interval $[z_l, z_u]$.

Let $m = (u - l)/2 + l$ be the median value in the range $[z_l, z_u]$. If the δ-feasibility question is true for $\delta = z_m$, then we know the optimal solution lies in the interval $[z_l, z_m]$; otherwise, we make a *heuristic* conclusion that the optimal solution lies in the interval $(z_m, z_u]$. We can repeat this procedure until we iterate to the point that $z_l = z_u$, heuristically concluding the optimal BTSP objective value for our problem instance is z_l. Algorithm 2.3, the *BinarySearchTreshold* method, formally illustrates this procedure.

We will see the *SimpleThreshold* method will often be enough to find an optimal BTSP tour for a given instance. For harder problems the *BinarySearchThreshold* method will be used to find good quality tours. As we've already stated, if we can find a tour whose largest cost equals a lower bound value, then we have a provably optimal solution. If we cannot find a tour whose largest cost is equal to a lower bound, then we can instead replace our TSP heuristic α with an exact TSP algorithm, such as Concorde's TSP branch-and-cut method [4], to create an exact BTSP algorithm.

It should be noted that Ramakrishnan et al. [91] explored solving the BTSP heuristically in much the same manner as our *BinarySearchThreshold* method without making

Algorithm 2.3: BinarySearchThreshold$(n, C, L, \alpha, p, q, r, s)$

Input: A problem on n nodes with cost matrix C, a lower bound L, TSP solver/heuristic α, and integers p, q, r, and s which represent the number of iterations with cost matrices C^δ, C_0^δ, C_1^δ, and C_2^δ, respectively, in the *IsFeasible* method.

Output: The (optimal/heuristic conclusion) on the BTSP objective value and tour.

$(feasible, best\_tour, max\_cost) \leftarrow$ IsFeasible$(n, C, L, \alpha, p, q, r, s)$;
if $feasible$ **then return** $(L, best\_tour)$;

Let $z_1 < z_2 < ... < z_k$ be a list of the distinct costs from C in ascending order;
Let l be the integer such that $z_l = L$;
Let u be the integer such that $z_u = max\_cost$;
while $l \neq u$ **do**
    $m \leftarrow ((u - l)/2) + l$;
    $(feasible, tour, max\_cost) \leftarrow$ IsFeasible$(n, C, z_m, \alpha, p, q, r, s)$;
    if $feasible$ **then**
        Let u be the integer such that $z_u = max\_cost$;
        $best\_tour \leftarrow tour$;
    else
        $l \leftarrow m + 1$;
    end
end
return $(z_l, best\_tour)$;

any 'controlled' shake attempts. Their choice of TSP heuristic was Helsgaun's implementation [47] of the Lin-Kernighan heuristic. Further, instead of calculating a BBSSP lower bound, they used the 2-max bound. Finally, instead of finding an initial tour with a TSP heuristic, they used a *nearest neighbour* heuristic to compute an initial upper bound for a binary search. Their heuristic reported good experimental results on TSPLIB problems of up to 783 cities [95]. Of the 72 problems they considered, they found provably optimal solutions to 37 problems. Our computational results show their solutions to the remaining 35 problems were also optimal.

2.3 Computational Results

We coded the algorithms discussed in Sections 2.1 and 2.2 in ANSI C, compiled under the GNU C Compiler (GCC) version 3.2. All experiments conducted in this chapter were carried out on a 164-processor Sun V60 clustered computer at the University of New Brunswick in Fredericton, New Brunswick. The clustered computer consisted of 60 slave nodes of dual 2.8GHz Intel Xeon processors with 2 to 3 GB of RAM.

To test the effectiveness of the lower bounding schemes of Section 2.1 and the heuristics of Section 2.2, we tested these algorithms on the following sets of problems.

1. 108 TSPLIB instances of sizes up to 18,512 vertices [95]. TSPLIB is generally considered as the standard set of benchmark instances for the TSP.

2. 46 random instances due to Johnson and McGeoch of sizes up to 31,623 vertices [50]. These instances were created for the DIMACS TSP Challenge [52], the results of which Johnson and McGeoch summarized in a 2001 book chapter [51]. These instances can be classified into three different groups: uniform point, clustered points, and random distance matrices.

3. 87 VLSI instances of sizes up to 29,514 vertices [22]. These instances are based upon research into the design of integrated circuits, such as microprocessors.

4. 25 National TSP instances of sizes up to 24,978 vertices [24]. These problems offer TSP instances of sets of cities from a number of different countries around the world.

5. 81 specially structured 'hard' random instances of 100, 500, and 2,500 vertices. These instances were designed to challenge our heuristic by ensuring the BBSSP lower bound

value would not be equal to the optimal BTSP objective value. We describe the construction of these instances later in the section.

We did not test our heuristics on larger problems due to memory limitations.

Please note all reported running-times are in seconds.

Lower bound results

Our first set of experiments tested 101 TSPLIB problems of size up to 7,397 vertices to assess the relative strength of the 2-max bound and the BBSSP bound. The BBSSP lower bound was optimal for all problems except `ts225`; the 2-max bound was optimal for 52 of the 101 problems. Of course, the $O(m)$ 2-max bound is significantly cheaper to calculate than the more complex $O(m \log m)$ BBSSP bound. We present results for select instances in Table 2.1. By 'Gap' we mean the relative error between the lower bound's objective value z and the optimal BTSP objective value z^*, i.e.

$$\text{Gap} = \frac{|z^* - z|}{z^*}.$$

Due to its superior quality and reasonable computational time we used the BBSSP lower bound in our further experiments. Recall the BBSSP implementation used in our computational experiments is the $O(m \log m)$ algorithm of Parker and Rardin [79]. Larger problems would obviously benefit from more efficient implementations of the BBSSP lower bound, such as Punnen and Nair's $O(m + n \log n)$ implementation [86], or Manku's linear $O(m)$ algorithm [73].

Results with the *SimpleThreshold* heuristic

We first consider the *SimpleThreshold* heuristic, Algorithm 2.2, on the same set of TSPLIB problems used in our lower bound experiments. For our initial experiments, we consider the effect using 'uncontrolled' shake operations, that is to say cost matrices defined by Equation (2.6), has on solution quality when used in the *IsFeasible* method. To that end, we set parameters p, r, and s as 1, 0, and 0, respectively, and vary the value of q. As mentioned previously, the random numbers generated in Equation (2.6) are selected from the interval $[1, n^2]$.

The results were surprisingly good as the SimpleThreshold heuristic found provably

	2-max bound			BBSSP bound		
Problem	Obj. Val.	Gap (%)	Time	Obj. Val.	Gap (%)	Time
ts225	500	50.00	0.00	500	50.00	0.01
att532	227	0.87	0.00	229	0.00	0.10
ali535	3,889	0.00	0.00	3,889	0.00	0.12
si535	186	18.06	0.00	227	0.00	0.06
pa561	16	0.00	0.00	16	0.00	0.08
u574	345	0.00	0.00	345	0.00	0.12
rat575	23	0.00	0.00	23	0.00	0.11
p654	934	23.63	0.00	1,223	0.00	0.17
d657	1,368	0.00	0.00	1,368	0.00	0.13
gr666	4,264	0.00	0.00	4,264	0.00	0.20
u724	152	10.59	0.00	170	0.00	0.14
rat783	26	0.00	0.01	26	0.00	0.14
dsj1000	92,501	68.74	0.01	295,939	0.00	0.84
pr1002	1,662	21.94	0.01	2,129	0.00	0.43
si1032	107	70.44	0.01	362	0.00	0.27
u1060	2,378	0.00	0.01	2,378	0.00	0.73
pcb1173	243	0.00	0.01	243	0.00	0.61
d1291	1,289	0.00	0.01	1,289	0.00	0.78
rl1304	789	48.60	0.01	1,535	0.00	0.62
rl1323	1,905	23.46	0.01	2,489	0.00	0.66
nrw1379	105	0.00	0.02	105	0.00	0.56
fl1400	530	0.00	0.01	530	0.00	0.52
u1432	300	0.00	0.02	300	0.00	1.06
fl1577	137	68.21	0.02	431	0.00	1.58
d1655	1,476	0.00	0.02	1,476	0.00	1.38
u1817	234	0.00	0.02	234	0.00	1.54
rl1889	752	16.07	0.02	896	0.00	1.11
d2103	1,133	0.00	0.03	1,133	0.00	1.52
u2152	105	0.00	0.03	105	0.00	2.66
u2319	224	0.00	0.04	224	0.00	2.26
pr2392	401	16.63	0.04	481	0.00	4.10
pcb3038	181	8.59	0.07	198	0.00	5.68
fl3795	528	0.00	0.10	528	0.00	3.57
fnl4461	132	0.00	0.15	132	0.00	7.68
rl5915	580	3.65	0.24	602	0.00	12.24
rl5934	533	40.51	0.24	896	0.00	10.47
pla7397	69,772	14.33	0.37	81,438	0.00	48.39

Table 2.1: Lower bound comparison on select TSPLIB instances

	$q = 0$		$q = 1$		$q = 3$		$q = 5$	
Problem	% Gap	Time	% Gap	Time	% Gap	Time	% Gap	Time
ts225	253.60	0.12	253.60	0.50	253.60	1.30	253.60	2.20
u2152	2.10	23.11	0.00	21.27	0.00	55.23	0.00	18.73
pr2392	3.37	21.62	0.60	27.00	0.00	21.50	3.01	203.56
rl5915	12.51	46.24	0.50	95.15	0.50	194.51	3.80	433.90
rl5934	8.78	48.03	0.00	44.14	0.00	43.99	0.00	47.15
pla7397	40.21	64.72	35.51	101.50	67.02	277.02	53.61	347.75

Table 2.2: *SimpleThreshold* heuristic results on select TSPLIB instances for $(p, r, s) = (1, 0, 0)$ and varying values of q, corresponding to 'uncontrolled' shake operations with Equation (2.6). Results are averages from 10 trials.

optimal solutions of problems up to 2,103 vertices without performing *any* shake operations, with the exception of the `ts225` problem. Table 2.2 presents results for the remaining five problems which did not produce optimal solutions with this setup, as well as results for problem `ts225`.

Performing shake operations seem to help the *SimpleThreshold* heuristic produce optimal solutions in some cases. However, it seems spending additional time performing uncontrolled shake operations does not necessarily improve solution quality. There is a large amount of randomness inherent in our algorithm, in both the shake operations as well as the Lin-Kernighan TSP heuristic. This might explain why sometimes the heuristic will perform poorly.

To try and minimize the variance in solution quality, we performed a similar experiment using Equation (2.7) to perform "Type I" controlled shake operations instead. In this set of experiments with the *SimpleThreshold* heuristic, we set p, q, and s equal to 1, 0, and 0, respectively, while varying values for r. As before, the random numbers generated in Equation (2.7) are selected from the interval $[1, n^2]$. As in the previous experiment, optimal solutions were found for most problems with no shake operations necessary. Table 2.3 presents results on the same five problems as Table 2.2. We see the solution quality is much better across the board.

Finally, we tested the effect of 'Type II' controlled shaking, Equation (2.9) on solution quality for varying values of s, with p, q, and s set to 1, 0, and 0, respectively. We also tested different values of θ and τ to see if some combinations worked significantly better than others. Table 2.4 presents results for $s = 3$ (i.e. 3 Type II controlled shakes) and varying values for θ and τ. No significant difference in solution quality is observed for differing

	$r = 1$		$r = 3$		$r = 5$	
Problem	% Gap	Time	% Gap	Time	% Gap	Time
ts225	182.80	0.30	164.41	0.64	190.32	0.95
u2152	0.00	23.68	0.00	30.08	0.00	28.08
pr2392	0.00	23.15	0.00	28.60	0.00	34.04
rl5915	9.70	113.51	3.21	153.45	0.00	162.96
rl5934	0.00	52.08	0.00	55.25	0.00	54.10
pla7397	26.81	211.62	20.10	254.21	33.51	451.67

Table 2.3: *SimpleThreshold* heuristic results on select TSPLIB instances for $(p, q, s) = (1, 0, 0)$ and varying values of r, corresponding to Type I 'controlled' shake operations with Equation (2.7). Results are averages from 10 trials.

	$\theta = 5, \tau = 10$		$\theta = 10, \tau = 25$		$\theta = 5, \tau = 100$		$\theta = 50, \tau = 100$	
Problem	% Gap	Time	% Gap	Time	% Gap	Time	% Gap	Time
ts225	253.60	0.40	253.60	0.40	253.60	0.45	253.60	0.46
u2152	0.00	42.74	0.00	47.12	2.10	34.91	0.00	32.69
pr2392	0.00	33.23	0.00	39.01	0.00	34.32	0.00	49.34
rl5915	9.82	196.63	9.70	232.78	9.65	145.64	49.45	214.76
rl5934	0.00	58.11	0.00	51.22	0.00	56.33	8.78	75.81
pla7397	33.51	229.96	40.21	238.55	46.91	257.63	40.21	265.27

Table 2.4: *SimpleThreshold* heuristic results on select TSPLIB instances for $(p, q, r, s) = (1, 0, 0, 3)$ and varying values for θ and τ, to Type II 'controlled' shake operations with Equation (2.9). Results are averages from 10 trials.

values of θ and τ, so we used $\theta = 10$, $\tau = 25$ in future experiments.

The results using 'Type II' controlled shakes appears to be not as good as those for 'Type I' controlled shakes. We believe this is due to the usage of the large integer M for corresponding elements in the original matrix that have cost $c_{ij} > U$. For future experiments, we removed this feature and instead used

$$c_{ij}^{\delta,2} = \begin{cases} 0 & \text{if } c_{ij} \leq \delta, \\ z_k + r_k & \text{otherwise, and where } z_k = c_{ij} \end{cases} \tag{2.9}$$

for future constructions of Type II shake matrices (i.e. with the use of the large constant M is removed).

Finally, we conducted experiments with Algorithm 2.3, the *BinarySearchTreshold* heuristic. We conducted two experiments: one with Type I controlled shaking, and one with Type

Parameter	Type I Exp.	Type II Exp.
p	1	1
q	0	0
r	3	0
s	0	3
a	1	-
b	n^2	-
θ	-	10
τ	-	25

Table 2.5: Parameters for the 'Type I' and 'Type II' *BinarySearchThreshold* heuristic experiments.

	Type I Exp.		Type II Exp.	
Problem	Gap (%)	Time	Gap (%)	Time
ts225	0.00	8.67	0.00	11.62
d2103	0.00	42.29	0.00	42.29
u2152	0.00	44.06	0.00	45.35
u2319	0.00	24.89	0.00	24.80
pr2392	0.00	30.10	0.00	51.18
pcb3038	0.00	32.91	0.00	32.83
fl3795	0.00	52.45	0.00	52.22
fnl4461	0.00	46.85	0.00	46.68
rl5915	0.08	197.15	0.00	163.96
rl5934	0.00	66.52	0.00	70.59
pla7397	0.55	737.69	0.00	418.32

Table 2.6: *BinarySearchThreshold* heuristic results on select TSPLIB instances. Results are averages from 10 trials. Running times include calculation of the BBSSP lower bound.

II controlled shaking. The parameters for these experiments are given in Table 2.5. We obtained optimal solutions for all but two problems with a minimal gap from the optimum for the others. The results for select problems are given in Table 2.6, and a summary of running-times for problems of up to 1,889 vertices is given in Table 2.7.

We also tested this algorithm on random instances due to Johnson and McGeoch [50]. These results are presented in Tables 2.15 and 2.16. All solutions obtained are optimal, and the running-times almost uniform with the exception of problem `C10K.1`, which is an outlier. The *BinarySearchThreshold* heuristic is clearly effective in obtaining good quality solutions in a reasonable running-time.

Problem size	Num. of Problems	Type I Exp. Avg. Time	Type II Exp. Avg. Time
14–99	22	0.25	0.25
100–198	25	1.39	1.41
200–318	11	3.27	3.33
400–493	6	9.92	10.06
532–575	6	11.06	10.97
654–783	5	16.33	16.03
1,000–1,291	6	29.79	30.14
1,304–1,432	5	25.45	25.41
1,577–1,889	4	31.65	30.62

Table 2.7: Average running times for the *BinarySearchThreshold* heuristic from 10 trials on each TSPLIB problem of size up to 1,889 vertices. Running times include calculation of the BBSSP lower bound.

		Type I Exp.		Type II Exp.	
Problem	Size	Gap (%)	Time	Gap (%)	Time
C1k.0	1,000	0.00	32.95	0.00	31.86
C1k.1	1,000	0.00	34.56	0.00	34.23
C1k.2	1,000	0.00	29.68	0.00	30.60
C1k.3	1,000	0.00	41.93	0.00	37.19
C1k.4	1,000	0.00	34.09	0.00	31.95
C1k.5	1,000	0.00	47.43	0.00	57.74
C1k.6	1,000	0.00	32.18	0.00	32.15
C1k.7	1,000	0.00	36.01	0.00	35.94
C1k.8	1,000	0.00	32.84	0.00	31.61
C1k.9	1,000	0.00	30.10	0.00	30.22
C3k.0	3,162	0.00	69.17	0.00	81.19
C3k.1	3,162	0.00	102.30	0.00	68.28
C3k.2	3,162	0.00	67.67	0.00	67.75
C3k.3	3,162	0.00	72.52	0.00	79.93
C3k.4	3,162	0.00	67.29	0.00	67.58
C10k.0	10,000	0.00	376.47	0.00	325.26
C10k.1	10,000	6.60	33,354.23	6.58	5,945.74
C10k.2	10,000	0.00	430.84	0.00	343.23

Table 2.8: *BinarySearchThreshold* heuristic results from 10 trials on Johnson-McGeoch clustered-point random instances sizes up to 10,000 vertices. Running times include calculation of the BBSSP lower bound.

		Type I Exp.		Type II Exp.	
Problem	Size	Gap (%)	Time	Gap (%)	Time
E1k.0	1,000	0.00	16.87	0.00	16.90
E1k.1	1,000	0.00	18.05	0.00	17.98
E1k.2	1,000	0.00	22.67	0.00	22.36
E1k.3	1,000	0.00	15.86	0.00	15.84
E1k.4	1,000	0.00	17.97	0.00	17.91
E1k.5	1,000	0.00	16.07	0.00	15.99
E1k.6	1,000	0.00	18.81	0.00	18.65
E1k.7	1,000	0.00	18.71	0.00	18.60
E1k.8	1,000	0.00	15.20	0.00	15.13
E1k.9	1,000	0.00	18.30	0.00	18.26
E3k.0	3,162	0.00	41.96	0.00	41.93
E3k.1	3,162	0.00	39.37	0.00	39.59
E3k.2	3,162	0.00	38.39	0.00	38.56
E3k.3	3,162	0.00	43.92	0.00	43.98
E3k.4	3,162	0.00	39.36	0.00	39.15
E10k.0	10,000	0.00	239.55	0.00	239.07
E10k.1	10,000	0.00	243.48	0.00	243.08
E10k.2	10,000	0.00	237.07	0.00	237.03
M1k.0	1,000	0.00	52.21	0.00	52.39
M1k.1	1,000	0.00	51.71	0.00	51.46
M1k.2	1,000	0.00	55.36	0.00	54.92
M1k.3	1,000	0.00	53.94	0.00	53.91
M3k.0	3,162	0.00	129.47	0.00	130.00
M3k.1	3,162	0.00	128.01	0.00	127.84
M10k.0	10,000	0.00	536.46	0.00	534.94

Table 2.9: *BinarySearchThreshold* heuristic results from 10 trials on Johnson-McGeoch uniform-point and random distance instances sizes up to 10,000 vertices. Running times include calculation of the BBSSP lower bound.

Results on 'hard' problems

In the computational results we have seen so far, the optimal BBSSP objective value often equals the optimal BTSP objective value. While it is desirable to have such a situation, as it provides a proof of optimality, it does not challenge our heuristic. Thus, we have generated the following special class of problems which do not have this property.

Let β and γ be two positive integers where $\beta < \gamma$. To construct 'hard' random instances we first define three matrices: a $r \times r$ matrix A with entries in the range $[\beta + 1, \gamma]$; a $r \times s$ matrix B with entries in the range $[0, \beta]$; and, a $s \times s$ matrix D with entries in the range $[\beta + 1, \gamma]$. Using these three cost matrices, we construct an $n \times n$ cost matrix C where $n = r + s$ and

$$C = \begin{pmatrix} A & B \\ B^T & D \end{pmatrix}. \tag{2.10}$$

Claim 1. *Cost matrix C of Equation* (2.10) *is guaranteed to have an optimal BTSP objective value greater than or equal to $\beta+1$, but an optimal BBSSP objective value less than or equal to β whenever $r \neq s$ for $r, s \geq 2$.*

Proof. Consider the complete graph G induced by C constructed by Equation (2.10) for some β and γ, $\beta < \gamma$, and cost matrices A, B, and D. By construction, we note the following.

1. For nodes $i = 1, 2, \ldots, r$ and:
   (a) for nodes $j = 1, 2, \ldots, r$, $i \neq j$ that $c_{ij} \geq \beta + 1$; likewise,
   (b) for nodes $j = (r + 1), (r + 2), \ldots, (r + s)$, $i \neq j$, that $c_{ij} \leq \beta$.
2. For nodes $i = (r + 1), (r + 2), \ldots, (r + s)$ and:
   (a) for nodes $j = 1, 2, \ldots, r$, $i \neq j$, that $c_{ij} \leq \beta$; likewise,
   (b) for nodes $j = (r + 1), (r + 2), \ldots, (r + s)$, $i \neq j$, that $c_{ij} \geq \beta + 1$.

Ideally a tour could be found that only uses cost less than or equal to β. This means every node $i = 1, 2, \ldots, r$ would have to be paired with a node $j = (r+1), (r+2), \ldots, (r+s)$, $i \neq j$. Likewise, every node $i = (r + 1), (r + 2), \ldots, (r + s)$ would have to be paried with a node $j = 1, 2, \ldots, r$, $i \neq j$. This is clearly not possible unless $r = s$, therefore any Hamiltonian cycle in C must include at least one edge of cost greater than or equal to $\beta + 1$.

However, because $r, s \geq 2$, for any node i we can find at least two vertex-disjoint paths between it and any other node j using only edge costs less than or equal to β. For example, for any $i = 1, 2, \ldots, r$, and $j = 1, 2, \ldots, r$, $i \neq j$, the paths $\{i, (r+1), j\}$ and $\{i, (r+2), j\}$ exist. This means the graph G is biconnected even if we only consider edge costs less than or equal to β, and that the optimal BBSSP objective value cannot be greater than β. □

To test the *BinarySearchTreshold* heuristic on these specially-structured hard instances, we use the following parameters: $p = 1$, $q = 0$, $r = 3$, $s = 3$, $a = 1$, $b = n^2$, $\theta = 10$, and $\tau = 25$. When $r = n/2$ the optimal BBSSP objective value may coincide with the optimal BTSP objective value; for all other values of r the optimal BBSSP objective value is guaranteed to be strictly lower than the optimal BTSP objective value.

We attempted to verify the optimality of these instances using an exact TSP solver as follows. Let z be the BTSP objective value of the solution our *BinarySearchThreshold* algorithm computes. Solve the δ-feasibility question, Algorithm 2.1, for $\delta = z - 1$ and an *exact* TSP solver α, such as Concorde's TSP solver [4]. If the δ-feasibility answer is 'no', then z must clearly be the optimal BTSP objective value; if the answer is 'yes', then z is clearly not the optimal BTSP objective value.

Table 2.10 presents results on these specially-structured hard instances with 500 vertices. Likewise, Table 2.11 presents results on these specially-structured hard instances with 2,500 vertices. The parameters β, γ, and r used to generate the test instances are given, along with the random number generator's seed value. We obtain optimal solutions for all 'hard' problems with 500 vertices, and guarantee optimality for 20 of 27 problems with 2,500 vertices. We also tested 'hard' instances with 100 vertices and obtained optimal solutions for all, but we do not present these results. The running-time of these instances is much higher compared to the other classes of problems, as expected.

β	γ	r	Seed	BTSP Obj.	Optimal?	BBSSP Obj.	BBSSP Time	LK Calls	Avg. LK Time	Total Time
500	5,000	125	6,125	525	Yes	29	0.07	57	108.05	6,190.58
500	5,000	250	6,250	17	Yes	17	0.08	1	5.86	5.94
500	5,000	375	6,375	524	Yes	35	0.08	50	109.19	5,491.03
500	7,500	125	8,625	538	Yes	31	0.08	58	97.84	5,709.70
500	7,500	250	8,750	22	Yes	22	0.08	1	5.68	5.77
500	7,500	375	8,875	537	Yes	36	0.08	58	90.37	5,272.00
500	10,000	125	11,125	547	Yes	53	0.09	52	90.77	4,749.93
500	10,000	250	11,250	20	Yes	20	0.08	1	5.88	5.98
500	10,000	375	11,375	551	Yes	30	0.10	60	90.71	5,471.06
1,000	5,000	125	6,625	1,022	Yes	56	0.08	65	100.55	6,573.59
1,000	5,000	250	6,750	26	Yes	26	0.08	1	5.50	5.60
1,000	5,000	375	6,875	1,019	Yes	73	0.08	56	107.78	6,061.48
1,000	7,500	125	9,125	1,031	Yes	59	0.08	65	102.46	6,697.39
1,000	7,500	250	9,250	40	Yes	40	0.08	1	5.91	5.99
1,000	7,500	375	9,375	1,034	Yes	55	0.08	65	97.96	6,396.32
1,000	10,000	125	11,625	1,052	Yes	70	0.08	64	88.71	5,721.51
1,000	10,000	250	11,750	34	Yes	34	0.09	1	5.88	5.98
1,000	10,000	375	11,875	1,047	Yes	99	0.08	64	82.80	5,336.26
2,500	5,000	125	8,125	2,514	Yes	175	0.08	72	95.30	6,897.77
2,500	5,000	250	8,250	88	Yes	88	0.07	1	6.10	6.18
2,500	5,000	375	8,375	2,513	Yes	156	0.06	78	90.44	7,088.92
2,500	7,500	125	10,625	2,530	Yes	174	0.08	63	101.75	6,446.26
2,500	7,500	250	10,750	106	Yes	106	0.08	1	5.69	5.79
2,500	7,500	375	10,875	2,528	Yes	170	0.08	71	91.07	6,493.24
2,500	10,000	125	13,125	2,534	Yes	171	0.08	79	86.61	6,874.99
2,500	10,000	250	13,250	83	Yes	83	0.08	1	6.02	6.11
2,500	10,000	375	13,375	2,536	Yes	161	0.08	64	101.73	6,539.55

Table 2.10: *BinarySearchThreshold* heuristic results from 10 trials on 'Hard' random instances of 500 vertices. Running times include calculation of the BBSSP lower bound.

β	γ	r	Seed	BTSP Obj.	Optimal?	BBSSP Obj.	BBSSP Time	LK Calls	Avg. LK Time	Total Time
500	5,000	625	8,625	505	???	7	4.16	64	377.10	24,305.76
500	5,000	1,250	9,250	4	Yes	4	3.80	1	27.82	32.01
500	5,000	1,875	9,875	505	Yes	9	3.90	54	400.87	21,989.61
500	7,500	625	11,125	508	Yes	8	4.35	64	361.42	23,334.92
500	7,500	1,250	11,750	4	Yes	4	3.97	1	28.18	32.55
500	7,500	1,875	12,375	507	???	8	4.20	58	389.32	22,762.23
500	10,000	625	13,625	510	???	6	4.91	58	408.11	23,854.55
500	10,000	1,250	14,250	4	Yes	4	4.13	1	27.94	32.44
500	10,000	1,875	14,875	510	Yes	7	4.25	64	367.58	23,708.87
1,000	5,000	625	9,125	1,005	Yes	16	4.63	71	352.41	25,242.82
1,000	5,000	1,250	9,750	6	Yes	6	4.27	1	31.18	35.85
1,000	5,000	1,875	10,375	1,005	Yes	16	4.37	71	351.78	25,255.14
1,000	7,500	625	11,625	1,007	Yes	14	4.08	70	362.80	25,724.98
1,000	7,500	1,250	12,250	8	Yes	8	4.03	1	30.41	34.84
1,000	7,500	1,875	12,875	1,007	???	16	4.13	65	375.00	24,531.80
1,000	10,000	625	14,125	1,009	???	17	4.54	65	364.93	23,909.13
1,000	10,000	1,250	14,750	6	Yes	6	4.58	1	30.31	35.28
1,000	10,000	1,875	15,375	1,009	???	17	4.30	65	369.04	24,152.90
2,500	5,000	625	10,625	2,503	Yes	39	4.31	79	340.14	27,607.44
2,500	5,000	1,250	11,250	21	Yes	21	4.26	1	28.52	33.17
2,500	5,000	1,875	11,875	2,503	Yes	40	4.34	78	339.14	26,995.48
2,500	7,500	625	13,125	2,506	Yes	46	4.61	72	356.50	25,895.66
2,500	7,500	1,250	13,750	22	Yes	22	4.24	1	31.88	36.52
2,500	7,500	1,875	14,375	2,506	Yes	35	4.35	78	352.15	27,706.35
2,500	10,000	625	15,625	2,509	Yes	38	4.59	72	367.34	26,657.82
2,500	10,000	1,250	16,250	17	Yes	17	4.18	1	32.69	37.25
2,500	10,000	1,875	16,875	2,508	???	37	4.28	72	368.48	26,699.86

Table 2.11: *BinarySearchThreshold* heuristic results from 10 trials on 'Hard' random instances of 2,500 vertices. Running times include calculation of the BBSSP lower bound.

Concluding results

For our final experiment, we tested all available standard problems, subject to memory limitations, with the *BinarySearchThreshold* heuristic given the following parameters: $p = 1$, $q = 0$, $r = 3$, $s = 3$, $a = 1$, $b = n^2$, $\theta = 10$, and $\tau = 25$. The outcome of these results are summarized in the following tables:

Problem Set	# Vertices	Table	Page
TSPLIB [95]	14–136	2.12	45
	137–654	2.13	46
	657–18,512	2.14	47
Random Instances: [50]			
- Clustered-Point	1,000–31,623	2.15	48
- Uniform-Point	1,000–31,623	2.16	49
- Random Matrices	1,000–10,000	2.17	49
VLSI [22]	131–2,086	2.18	50
	2,144–3,954	2.19	51
	4,355–29,514	2.20	52
National [24]	29–24,978	2.21	53

Our algorithm found optimal solutions for all but the problem `sw24978` with 24,978 vertices.

Using the large amount of data generated in the experiments we present a plot of problem-size versus average running-time in Figure 2.1. We also fit a quadratic function to the data points which approximates the running-time quite well. This is consistent with experimental results on the Lin-Kernighan algorithm which estimates its average complexity to be $O(n^{2.2})$ [47, 51].

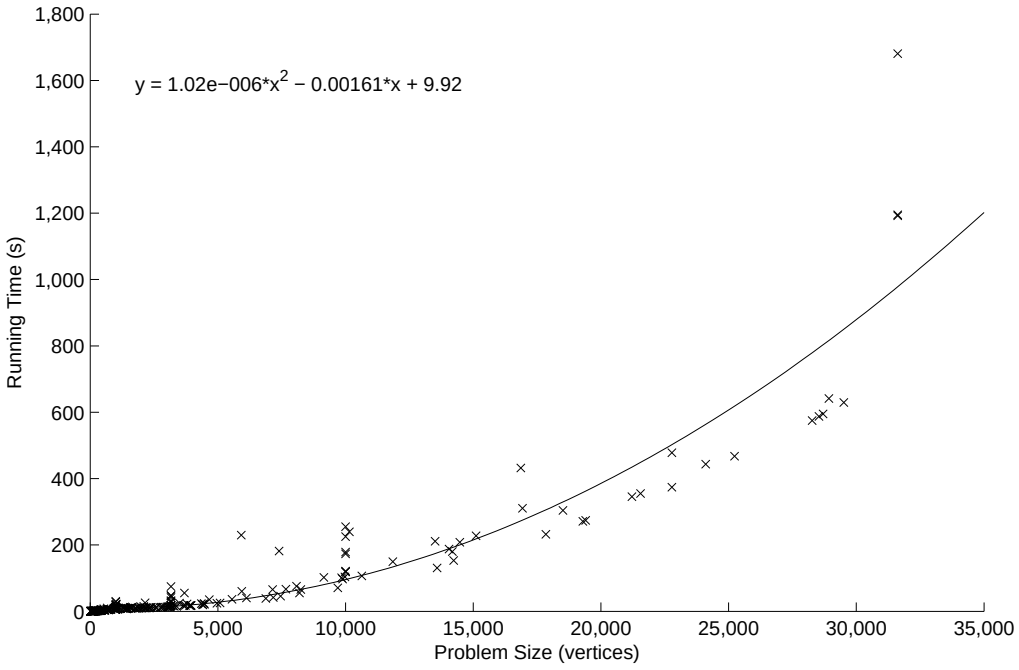


Figure 2.1: Problem-size versus average running-time

Problem	Size	BTSP Obj.	BBSSP Obj.	BBSSP Avg. Time	Avg. LK Calls	Avg. LK Time	Total Time
burma14	14	418	418	0.00	1.00	0.00	0.01
ulysses16	16	1,504	1,504	0.00	1.00	0.01	0.01
gr17	17	282	282	0.00	1.00	0.01	0.01
gr21	21	355	355	0.00	1.00	0.02	0.02
ulysses22	22	1,504	1,504	0.00	1.00	0.02	0.03
gr24	24	108	108	0.00	1.00	0.02	0.02
fri26	26	93	93	0.00	1.00	0.03	0.03
bayg29	29	111	111	0.00	1.00	0.03	0.03
bays29	29	154	154	0.00	1.00	0.03	0.03
dantzig42	42	35	35	0.00	1.00	0.05	0.05
swiss42	42	67	67	0.00	1.00	0.06	0.06
att48	48	519	519	0.00	1.00	0.08	0.08
gr48	48	227	227	0.00	1.00	0.07	0.07
hk48	48	534	534	0.00	1.00	0.06	0.07
eil51	51	13	13	0.00	1.00	0.05	0.05
berlin52	52	475	475	0.00	1.00	0.12	0.13
brazil58	58	2,149	2,149	0.00	1.00	0.18	0.18
st70	70	24	24	0.00	1.00	0.15	0.15
eil76	76	16	16	0.00	1.00	0.22	0.22
pr76	76	3,946	3,946	0.00	1.00	0.20	0.20
gr96	96	2,807	2,807	0.07	1.00	0.35	0.43
rat99	99	20	20	0.00	1.00	0.09	0.10
kroA100	100	475	475	0.01	1.00	0.19	0.20
kroB100	100	530	530	0.00	1.00	0.24	0.25
kroC100	100	498	498	0.00	1.00	0.18	0.19
kroD100	100	491	491	0.01	1.00	0.18	0.19
kroE100	100	490	490	0.00	1.00	0.18	0.19
rd100	100	221	221	0.00	1.00	0.29	0.30
eil101	101	13	13	0.00	1.00	0.25	0.25
lin105	105	487	487	0.01	1.00	0.28	0.28
pr107	107	7,050	7,050	0.00	1.00	0.41	0.42
gr120	120	220	220	0.00	1.00	0.48	0.49
pr124	124	3,302	3,302	0.01	1.00	0.47	0.48
bier127	127	7,486	7,486	0.01	1.00	0.77	0.78
ch130	130	142	142	0.01	1.00	0.41	0.42
pr136	136	2,976	2,976	0.01	1.00	0.53	0.54

Table 2.12: TSPLIB instances (14–136 vertices)

Problem	Size	BTSP Obj.	BBSSP Obj.	BBSSP Avg. Time	Avg. LK Calls	Avg. LK Time	Total Time
gr137	137	2,132	2,132	0.16	1.00	0.51	0.69
pr144	144	2,570	2,570	0.01	1.00	0.50	0.51
ch150	150	93	93	0.01	1.00	0.37	0.38
kroA150	150	392	392	0.01	1.00	0.32	0.34
kroB150	150	436	436	0.01	1.00	0.39	0.40
pr152	152	5,553	5,553	0.01	1.00	0.66	0.67
u159	159	800	800	0.01	1.00	0.51	0.52
si175	175	177	177	0.00	1.00	0.67	0.68
brg180	180	30	30	0.00	1.00	0.99	1.00
rat195	195	21	21	0.01	1.00	0.50	0.51
d198	198	1,380	1,380	0.02	1.00	1.40	1.42
kroA200	200	408	408	0.02	1.00	0.65	0.68
kroB200	200	344	344	0.02	1.00	0.52	0.55
gr202	202	2,230	2,230	0.36	1.00	1.57	1.97
ts225	225	1,000	500	0.02	10.40	2.92	2.99
tsp225	225	36	36	0.02	1.00	0.72	0.75
pr226	226	3,250	3,250	0.02	1.00	0.91	0.94
gr229	229	4,027	4,027	0.47	1.00	1.31	1.85
gil262	262	23	23	0.02	1.00	0.93	0.96
pr264	264	4,701	4,701	0.03	1.00	1.79	1.83
a280	280	20	20	0.02	1.00	0.89	0.92
pr299	299	498	498	0.05	1.00	2.35	2.41
lin318	318	487	487	0.04	1.00	1.36	1.41
rd400	400	104	104	0.07	1.00	1.86	1.94
fl417	417	472	472	0.07	1.10	5.58	5.67
gr431	431	4,027	4,027	1.81	1.00	3.44	5.46
pr439	439	2,384	2,384	0.10	1.00	2.95	3.07
pcb442	442	500	500	0.10	1.00	2.83	2.94
d493	493	2,008	2,008	0.11	1.00	5.22	5.34
att532	532	229	229	0.27	1.00	3.53	3.85
ali535	535	3,889	3,889	2.84	1.00	4.37	7.55
si535	535	227	227	0.04	1.00	4.30	4.35
pa561	561	16	16	0.05	1.00	3.38	3.44
u574	574	345	345	0.16	1.00	3.54	3.73
rat575	575	23	23	0.12	1.00	2.67	2.82
p654	654	1,223	1,223	0.20	1.00	5.17	5.40

Table 2.13: TSPLIB instances (137–654 vertices)

Problem	Size	BTSP Obj.	BBSSP Obj.	BBSSP Avg. Time	Avg. LK Calls	Avg. LK Time	Total Time
d657	657	1,368	1,368	0.21	1.00	6.68	6.92
gr666	666	4,264	4,264	4.62	1.00	5.17	10.33
u724	724	170	170	0.24	1.00	3.72	4.00
rat783	783	26	26	0.25	1.00	5.16	5.46
dsj1000	1,000	295,939	295,939	1.01	1.00	10.85	11.96
pr1002	1,002	2,129	2,129	0.41	1.00	6.57	7.07
si1032	1,032	362	362	0.13	1.00	21.67	21.84
u1060	1,060	2,378	2,378	0.59	1.00	7.72	8.39
vm1084	1,084	998	998	0.73	1.00	6.08	6.91
pcb1173	1,173	243	243	0.67	1.00	7.52	8.30
d1291	1,291	1,289	1,289	0.70	1.00	11.68	12.49
rl1304	1,304	1,535	1,535	1.05	1.00	8.15	9.35
rl1323	1,323	2,489	2,489	1.03	1.00	9.70	10.87
nrw1379	1,379	105	105	0.94	1.00	7.34	8.44
fl1400	1,400	530	530	0.86	1.00	9.57	10.58
u1432	1,432	300	300	0.84	1.00	6.20	7.21
fl1577	1,577	431	431	1.08	1.00	9.66	10.94
d1655	1,655	1,476	1,476	0.84	1.00	12.42	13.44
vm1748	1,748	1,017	1,017	1.76	1.00	7.45	9.47
u1817	1,817	234	234	1.46	1.00	6.88	8.61
rl1889	1,889	896	896	1.91	1.00	7.52	9.72
d2103	2,103	1,133	1,133	2.02	1.00	12.30	14.65
u2152	2,152	105	105	2.05	1.00	23.22	25.66
u2319	2,319	224	224	2.20	1.00	7.38	10.03
pr2392	2,392	481	481	2.71	1.00	8.84	12.03
pcb3038	3,038	198	198	4.44	1.00	8.39	13.58
fl3795	3,795	528	528	6.99	1.00	13.91	22.01
fnl4461	4,461	132	132	9.84	1.00	12.09	23.59
rl5915	5,915	602	602	21.19	4.20	188.47	229.82
rl5934	5,934	896	896	20.09	1.00	37.30	60.29
pla7397	7,397	81,438	81,438	55.21	2.18	113.82	181.79
rl11849	11,849	842	842	81.20	1.00	56.66	149.59
usa13509	13,509	16,754	16,754	142.76	1.00	53.03	211.14
brd14051	14,051	1,306	1,306	113.44	1.00	57.98	187.73
d15112	15,112	1,370	1,370	133.18	1.00	75.59	227.86
d18512	18,512	476	476	184.11	1.00	91.34	304.16

Table 2.14: TSPLIB instances (657–18,512 vertices)

Problem	Size	BTSP Obj.	BBSSP Obj.	BBSSP Avg. Time	Avg. LK Calls	Avg. LK Time	Total Time
C1k.0	1,000	290,552	290,552	0.80	1.00	12.56	13.44
C1k.1	1,000	335,184	335,184	0.83	1.00	11.20	12.12
C1k.2	1,000	225,295	225,295	0.71	1.00	9.39	10.18
C1k.3	1,000	416,768	416,768	0.76	1.20	26.80	27.66
C1k.4	1,000	318,930	318,930	0.90	1.00	9.88	10.86
C1k.5	1,000	260,389	260,389	0.89	1.10	29.26	30.25
C1k.6	1,000	175,740	175,740	0.80	1.00	10.06	10.94
C1k.7	1,000	301,366	301,366	0.87	1.00	11.32	12.28
C1k.8	1,000	246,519	246,519	0.84	1.00	9.83	10.75
C1k.9	1,000	208,091	208,091	0.82	1.00	9.32	10.22
C3k.0	3,162	252,245	252,245	8.55	1.00	22.83	32.21
C3k.1	3,162	167,466	167,466	8.56	1.50	64.86	74.67
C3k.2	3,162	194,007	194,007	8.32	1.00	18.28	27.43
C3k.3	3,162	180,852	180,852	8.48	1.10	32.54	41.94
C3k.4	3,162	180,583	180,583	8.60	1.00	20.19	29.63
C10k.0	10,000	161,062	161,062	86.80	1.60	125.16	225.32
C10k.1	10,000	94,139	94,139	80.19	1.00	90.09	178.63
C10k.2	10,000	121,209	121,209	87.36	1.10	76.53	173.08
C31k.0	31,623	84,302	84,302	888.35	1.67	652.06	1,680.74

Table 2.15: Johnson-McGeoch clustered-point random instances

Problem	Size	BTSP Obj.	BBSSP Obj.	BBSSP Avg. Time	Avg. LK Calls	Avg. LK Time	Total Time
E1k.0	1,000	64,739	64,739	0.74	1.00	5.26	6.09
E1k.1	1,000	67,476	67,476	0.78	1.00	5.64	6.50
E1k.2	1,000	88,522	88,522	0.76	1.00	7.24	8.08
E1k.3	1,000	59,220	59,220	0.78	1.00	4.87	5.73
E1k.4	1,000	68,259	68,259	0.78	1.00	5.68	6.54
E1k.5	1,000	61,406	61,406	0.78	1.00	4.98	5.85
E1k.6	1,000	68,777	68,777	0.74	1.00	5.87	6.70
E1k.7	1,000	70,389	70,389	0.76	1.00	5.93	6.77
E1k.8	1,000	57,597	57,597	0.83	1.00	5.00	5.92
E1k.9	1,000	68,420	68,420	0.75	1.00	5.75	6.58
E3k.0	3,162	39,854	39,854	8.13	1.00	8.71	17.68
E3k.1	3,162	37,500	37,500	8.15	1.00	7.99	16.97
E3k.2	3,162	35,145	35,145	8.14	1.00	7.59	16.57
E3k.3	3,162	44,428	44,428	8.15	1.00	9.43	18.42
E3k.4	3,162	36,621	36,621	8.14	1.00	7.98	16.96
E10k.0	10,000	20,174	20,174	81.09	1.00	29.60	119.03
E10k.1	10,000	22,883	22,883	81.23	1.00	31.47	121.10
E10k.2	10,000	20,208	20,208	80.94	1.00	29.72	119.04
E31k.0	31,623	12,419	12,419	863.27	1.00	243.78	1,192.12
E31k.1	31,623	15,169	15,169	862.53	1.00	248.05	1,194.92

Table 2.16: Johnson-McGeoch uniform-point instances

Problem	Size	BTSP Obj.	BBSSP Obj.	BBSSP Avg. Time	Avg. LK Calls	Avg. LK Time	Total Time
M1k.0	1,000	9,328	9,328	0.60	1.00	17.13	17.77
M1k.1	1,000	8,856	8,856	0.64	1.00	16.74	17.42
M1k.2	1,000	11,282	11,282	0.62	1.00	19.46	20.11
M1k.3	1,000	11,617	11,617	0.59	1.00	19.28	19.91
M3k.0	3,162	3,289	3,289	9.31	1.00	37.77	47.56
M3k.1	3,162	3,034	3,034	9.27	1.00	36.82	46.57
M10k.0	10,000	1,189	1,189	115.83	1.00	129.45	255.15

Table 2.17: Johnson-McGeoch random matrix instances

Problem	Size	BTSP Obj.	BBSSP Obj.	BBSSP Avg. Time	Avg. LK Calls	Avg. LK Time	Total Time
xqf131	131	23	23	0.00	1.00	0.57	0.57
xqg237	237	13	13	0.02	1.00	1.02	1.05
pma343	343	15	15	0.04	1.00	1.72	1.77
pka379	379	11	11	0.05	1.00	1.69	1.75
bcl380	380	16	16	0.05	1.00	2.09	2.15
pbl395	395	13	13	0.04	1.00	2.26	2.31
pbk411	411	14	14	0.05	1.00	2.48	2.54
pbn423	423	14	14	0.05	1.00	2.40	2.46
pbm436	436	15	15	0.06	1.00	2.63	2.70
xql662	662	26	26	0.15	1.00	4.40	4.59
rbx711	711	15	15	0.13	1.00	4.18	4.35
rbu737	737	13	13	0.18	1.00	4.39	4.62
dkg813	813	18	18	0.22	1.00	5.19	5.46
lim963	963	13	13	0.26	1.00	6.65	6.98
pbd984	984	13	13	0.32	1.00	6.77	7.17
xit1083	1,083	11	11	0.39	1.00	5.98	6.46
dka1376	1,376	15	15	0.63	1.00	7.38	8.16
dca1389	1,389	18	18	0.72	1.00	7.51	8.39
dja1436	1,436	15	15	0.51	1.00	7.03	7.72
icw1483	1,483	21	21	0.74	1.00	8.15	9.07
fra1488	1,488	13	13	0.81	1.00	7.23	8.23
rbv1583	1,583	18	18	0.72	1.00	7.75	8.68
rby1599	1,599	18	18	0.74	1.00	8.63	9.58
fnb1615	1,615	45	45	0.86	1.00	10.38	11.46
djc1785	1,785	24	24	1.06	1.00	8.31	9.63
dcc1911	1,911	16	16	1.34	1.00	7.80	9.45
dkd1973	1,973	14	14	1.42	1.00	7.86	9.61
djb2036	2,036	15	15	1.36	1.00	7.84	9.54
dcb2086	2,086	21	21	1.42	1.00	8.22	10.00

Table 2.18: VLSI instances (131–2,086 vertices)

Problem	Size	BTSP Obj.	BBSSP Obj.	BBSSP Avg. Time	Avg. LK Calls	Avg. LK Time	Total Time
bva2144	2,144	15	15	1.69	1.00	7.86	9.93
xqc2175	2,175	15	15	1.54	1.00	8.25	10.18
bck2217	2,217	16	16	1.61	1.00	8.25	10.27
xpr2308	2,308	14	14	1.77	1.00	7.88	10.09
ley2323	2,323	27	27	2.24	1.00	8.77	11.47
dea2382	2,382	25	25	2.35	1.00	8.93	11.75
rbw2481	2,481	27	27	2.27	1.00	9.18	11.96
pds2566	2,566	15	15	2.14	1.00	8.45	11.14
mlt2597	2,597	21	21	2.48	1.00	8.94	11.98
bch2762	2,762	15	15	2.50	1.00	8.69	11.83
irw2802	2,802	15	15	2.57	1.00	9.30	12.52
lsm2854	2,854	19	19	2.66	1.00	9.50	12.85
dbj2924	2,924	16	16	3.16	1.00	9.11	12.99
xva2993	2,993	12	12	3.29	1.00	8.67	12.71
pia3056	3,056	15	15	3.35	1.00	9.44	13.57
dke3097	3,097	19	19	3.52	1.00	9.44	13.75
lsn3119	3,119	16	16	3.58	1.00	9.50	13.89
lta3140	3,140	17	17	3.64	1.00	9.49	13.95
fdp3256	3,256	20	20	3.88	1.00	11.70	16.46
beg3293	3,293	24	24	4.01	1.00	10.08	15.00
dhb3386	3,386	17	17	4.23	1.00	9.69	14.87
fjs3649	3,649	21	21	5.48	1.00	10.42	17.01
fjr3672	3,672	21	21	6.09	1.00	10.45	17.65
dlb3694	3,694	19	19	5.02	1.60	48.48	55.45
ltb3729	3,729	13	13	5.06	1.00	11.75	17.97
xqe3891	3,891	20	20	5.54	1.00	10.34	17.15
xua3937	3,937	15	15	5.71	1.00	9.91	16.91
dkc3938	3,938	16	16	6.28	1.00	9.96	17.53
dkf3954	3,954	20	20	5.72	1.00	10.54	17.57

Table 2.19: VLSI instances (2,144–3,954 vertices)

Problem	Size	BTSP Obj.	BBSSP Obj.	BBSSP Avg. Time	Avg. LK Calls	Avg. LK Time	Total Time
bgb4355	4,355	13	13	6.90	1.00	14.52	23.01
bgd4396	4,396	28	28	7.17	1.00	12.49	21.27
frv4410	4,410	15	15	6.33	1.00	11.05	19.00
bgf4475	4,475	25	25	7.35	1.00	12.26	21.28
xqd4966	4,966	16	16	9.89	1.00	12.05	23.99
fqm5087	5,087	16	16	10.44	1.00	12.60	25.19
fea5557	5,557	15	15	11.18	1.00	22.56	36.32
xsc6880	6,880	18	18	17.24	1.00	17.31	38.49
bnd7168	7,168	15	15	18.86	1.00	18.72	41.90
lap7454	7,454	16	16	20.44	1.00	20.47	45.54
ida8197	8,197	14	14	26.95	1.00	22.03	55.20
dga9698	9,698	15	15	34.51	1.00	28.36	70.74
xmc10150	10,150	17	17	41.64	3.10	165.65	239.83
xvb13584	13,584	15	15	67.32	1.00	48.01	130.84
xrb14233	14,233	14	14	82.20	1.00	53.88	153.11
xia16928	16,928	15	15	115.89	1.70	153.71	310.45
pjh17845	17,845	14	14	128.50	1.00	76.89	231.96
frh19289	19,289	28	28	151.02	1.00	89.00	271.25
fnc19402	19,402	18	18	152.32	1.00	90.24	274.10
ido21215	21,215	16	16	182.31	1.10	121.90	345.75
fma21553	21,553	16	16	206.98	1.00	108.81	354.83
lsb22777	22,777	23	23	210.71	1.00	119.88	374.12
xrh24104	24,104	17	17	258.39	1.00	136.06	443.29
bbz25234	25,234	15	15	259.39	1.00	154.36	467.37
irx28268	28,268	15	15	323.91	1.00	183.93	575.07
fyg28534	28,534	15	15	330.94	1.00	187.94	587.37
icx28698	28,698	18	18	335.63	1.00	190.00	594.93
boa28924	28,924	18	18	377.03	1.00	193.97	641.48
ird29514	29,514	16	16	354.49	1.00	201.55	629.35

Table 2.20: VLSI instances (4,355–29,514 vertices)

Problem	Size	BTSP Obj.	BBSSP Obj.	BBSSP Avg. Time	Avg. LK Calls	Avg. LK Time	Total Time
wi29	29	2,250	2,250	0.00	1.00	0.02	0.02
dj89	89	437	437	0.00	1.00	0.28	0.29
qa194	194	370	370	0.02	1.00	1.11	1.13
uy734	734	389	389	0.27	1.00	4.73	5.04
zi929	929	887	887	0.37	1.00	7.68	8.12
lu980	980	44	44	0.33	1.00	5.57	5.97
rw1621	1,621	150	150	1.23	1.00	10.01	11.46
mu1979	1,979	1,153	1,153	2.22	1.00	11.58	14.10
nu3496	3,496	650	650	6.68	1.00	17.14	24.84
ca4663	4,663	13,768	13,768	16.27	1.00	17.22	35.18
tz6117	6,117	486	486	21.28	1.00	15.89	40.26
eg7146	7,146	2,150	2,150	28.43	1.00	34.21	65.86
ym7663	7,663	3,113	3,113	37.08	1.00	25.83	66.67
pm8079	8,079	331	331	36.15	1.00	33.93	75.56
ei8246	8,246	124	124	36.16	1.00	23.86	65.69
ar9152	9,152	4,871	4,871	59.16	1.00	35.93	102.02
ja9847	9,847	6,413	6,413	61.65	1.00	32.95	101.17
gr9882	9,882	1,399	1,399	56.73	1.00	32.92	97.15
kz9976	9,976	1,602	1,602	61.80	1.00	33.84	103.94
fi10639	10,639	768	768	59.86	1.00	37.33	106.56
mo14185	14,185	939	939	108.14	1.00	55.67	180.20
ho14473	14,473	440	440	118.28	1.00	72.54	208.38
it16862	16,862	1,499	1,499	163.35	1.70	229.81	431.88
vm22775	22,775	388	388	293.00	1.00	141.35	477.76
sw24978	24,978	1,068	1,044	357.33	27.40	211.81	8,270.68

Table 2.21: National TSP instances

Chapter 3

The Asymmetric Bottleneck TSP

In this chapter we consider the bottleneck traveling salesman problem (BTSP) on n vertices where the cost matrix C is asymmetric (i.e. $c_{ij} \neq c_{ji}$ for some $i, j \in V$). We refer the reader to Chapters 1 and 2 for background on the BTSP. Section 3.1 describes how an asymmetric BTSP instance may be solved as a symmetric BTSP instance. A discussion of lower bounds on the optimal BTSP objective value for asymmetric instances is found in Section 3.2, and includes extensive theoretical analysis. Sections 3.3 and 3.4 present an approximation algorithm and a heuristic algorithm, respectively, for the asymmetric BTSP. Computational results using our heuristic algorithm is presented in Section 3.5. Finally, the maximum scatter TSP (MSTSP) and the constrained BTSP are considered in Sections 3.7 and 3.8, respectively.

3.1 The Asymmetric BTSP as a Symmetric BTSP

In Chapter 2 we discuss heuristic algorithms and lower bounds on the optimal objective value for the symmetric BTSP (i.e. $c_{ij} = c_{ji}$ for all $i, j \in V$). In this section we will show how to construct equivalent symmetric BTSP instances for any asymmetric BTSP instance. Symmetric BTSP lower bound algorithms and heuristics may then be used on the symmetric instance to solve the original asymmetric instance.

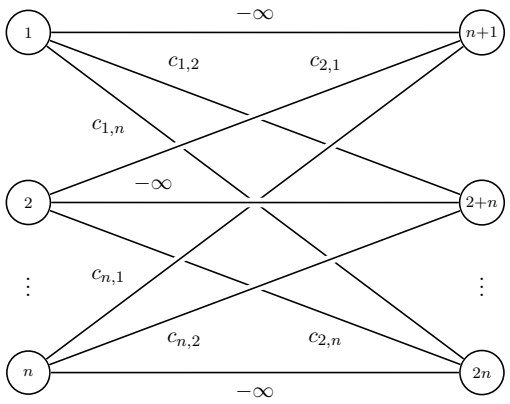


Figure 3.1: Graph of $2n$-vertex symmetric instance from n-vertex asymmetric instance as given by Equation (3.1). All arcs (i, j) not pictured have cost $\bar{c}_{ij} = \infty$.

3.1.1 A $2n$-Vertex Transformation

This formulation is similar to the one Ramakrishnan et al. proposed [91]. Given an asymmetric BTSP instance on n vertices and asymmetric cost matrix C, an equivalent symmetric instance on $2n$ vertices can be constructed with cost matrix $\bar{C} = (\bar{c}_{ij})_{2n \times 2n}$ where

$$\bar{c}_{ij} = \begin{cases} -\infty \text{ (fixed)} & \text{if } i = j + n \text{ or } j = i + n \\ c_{i,j-n} & \text{if } i \leq n \text{ and } j > n \\ c_{i-n,j} & \text{if } i > n \text{ and } j \leq n \\ \infty & \text{otherwise.} \end{cases} \tag{3.1}$$

The edges of cost $-\infty$ are fixed edges and must be included in any symmetric BTSP solution to make the transformation valid. We discuss how we force these edges into the solution later, but note that Ramakrishnan et al. use an additional constraint to enforce this [91]. Figures 3.1 and 3.2 depict the transformation graphically. Any tour $\pi = (\pi_1, \pi_2, \pi_3, ..., \pi_n)$ in the asymmetric instance C corresponds to the tour $\bar{\pi} = (\pi_1, \pi_2 + n, \pi_2, \pi_3 + n, \pi_3, ..., \pi_n, \pi_1 + n)$ in the symmetric instance, and vice-versa. Since $c_{ij} > -\infty$ for all i, j in C, the optimal BTSP objective value on $\bar{C}$ will be identical to the optimal BTSP objective value on C.

Claim 2. *If $\pi = (\pi_1, \pi_2, \pi_3, ..., \pi_n)$ is an optimal asymmetric BTSP tour in C, then*

$$\bar{\pi} = (\pi_1, \pi_2 + n, \pi_2, \pi_3 + n, \pi_3, ..., \pi_n, \pi_1 + n)$$

$\bar{c}_{ij}$	1	2	$\cdots$	n	$n+1$	$n+2$	$\cdots$	$2n$
1	$-$	∞	$\cdots$	∞	$-\infty$	c_{12}	$\cdots$	c_{1n}
2	∞	$-$	$\cdots$	∞	c_{21}	$-\infty$	$\cdots$	c_{2n}
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
n	∞	∞	$\cdots$	$-$	c_{n1}	c_{n2}	$\cdots$	$-\infty$
$n+1$	$-\infty$	c_{21}	$\cdots$	c_{n1}	$-$	∞	$\cdots$	∞
$n+2$	c_{12}	$-\infty$	$\cdots$	c_{n2}	∞	$-$	$\cdots$	∞
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$2n$	c_{1n}	c_{2n}	$\cdots$	$-\infty$	∞	∞	$\cdots$	$-$

Figure 3.2: Cost matrix of $2n$-vertex symmetric instance from n-vertex asymmetric instance as given by Equation (3.1).

is an optimal symmetric BTSP tour in $\bar{C}$, and vice-versa.

Proof. First we show if π is an optimal asymmetric BTSP tour in C that $\bar{\pi}$ is an optimal symmetric BTSP tour in $\bar{C}$. Suppose not, and that another tour $\bar{\phi} = (\phi_1, \phi_2 + n, \phi_2, \phi_3 + n, \phi_3, ..., \phi_n, \phi_1 + n)$ exists in $\bar{C}$ such that

$$-\infty < \max_{(i,j)\in\bar{\phi}} c_{ij} < \max_{(i,j)\in\bar{\pi}} c_{ij}$$

(assume without loss of generality that $(\phi_2+n, \phi_2), (\phi_3+n, \phi_3), \ldots, (\phi_1+n, \phi_1)$ are the fixed edges in ϕ with a cost of $-\infty$). Clearly, if ϕ exists in $\bar{C}$, then a tour $\phi = (\phi_1, \phi_2, \ldots, \phi_n)$ exists in C where

$$\max_{(i,j)\in\phi} c_{ij} < \max_{(i,j)\in\pi} c_{ij}.$$

Therefore, the tour $\bar{\phi}$ cannot exist in $\bar{C}$ if π is optimal for C, so $\bar{\pi}$ is optimal for $\bar{C}$.

The proof to show if $\bar{\pi}$ is an optimal symmetric BTSP tour in $\bar{C}$ that π is an optimal asymmetric BTSP tour in C is similar, so we omit the details. □

3.1.2 A $3n$-vertex Transformation

The following transformation is nearly identical to the $2n$ transformation, except it does not include any fixed edges. Given an asymmetric BTSP instance on n vertices and asymmetric cost matrix C, an equivalent symmetric instance of $3n$ vertices can be constructed with cost

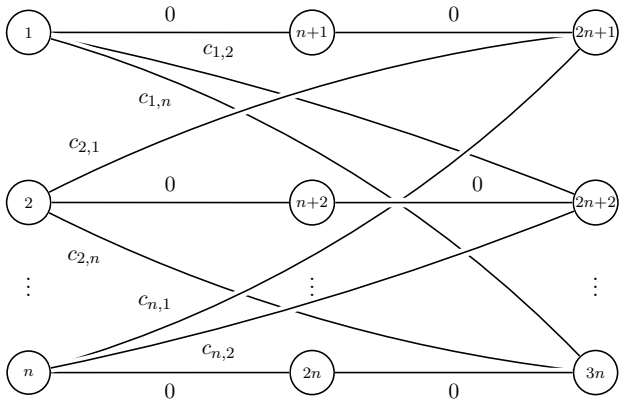


Figure 3.3: Graph of $3n$-vertex symmetric instance from n-vertex asymmetric instance as given by Equation (3.2). All arcs (i, j) not pictured have cost $\tilde{c}_{ij} = \infty$.

matrix $\tilde{C} = (\tilde{c}_{ij})_{3n \times 3n}$ where

$$\tilde{c}_{ij} = \begin{cases} 0 & \text{if } i = j + n \text{ or } j = i + n \\ c_{i,j-2n} & \text{if } i \leq n \text{ and } j > 2n \\ c_{i-2n,j} & \text{if } i > 2n \text{ and } j \leq n \\ \infty & \text{otherwise.} \end{cases} \tag{3.2}$$

Figures 3.3 and 3.4 depict the transformation graphically. Here, the pair of edges $(i, i + n)$ and $(i + n, i + 2n)$ act like the fixed edge in the $2n$-vertex transformation, but without the additional constraint. Any tour $\pi = (\pi_1, \pi_2, \pi_3, ..., \pi_n)$ in the asymmetric instance C corresponds to the tour $\tilde{\pi} = (\pi_1, \pi_2+2n, \pi_2+n, \pi_2, \pi_3+2n, \pi_3+n+\pi_3, ..., \pi_n, \pi_1+2n, \pi_1+n)$ in the symmetric instance, and vice-versa. As before, the optimal BTSP objective value on $\tilde{C}$ will clearly be identical to the optimal BTSP objective value on C.

Claim 3. *If $\pi = (\pi_1, \pi_2, \pi_3, ..., \pi_n)$ is an optimal tour in C, then*

$$\tilde{\pi} = (\pi_1, \pi_2 + 2n, \pi_2 + n, \pi_2, \pi_3 + 2n, \pi_3 + n + \pi_3, ..., \pi_n, \pi_1 + 2n, \pi_1 + n)$$

is an optimal tour in $\tilde{C}$, and vice-versa.

Proof. This proof is nearly identical to Claim 2, so we omit the details. □

$\tilde{c}_{ij}$	1	2	$\cdots$	n	$n+1$	$n+2$	$\cdots$	$2n$	$2n+1$	$2n+2$	$\cdots$	$3n$
1	$-$	∞	$\cdots$	∞	0	∞	$\cdots$	∞	$-\infty$	c_{12}	$\cdots$	c_{1n}
2	∞	$-$	$\cdots$	∞	∞	0	$\cdots$	∞	c_{21}	$-\infty$	$\cdots$	c_{2n}
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
n	∞	∞	$\cdots$	$-$	∞	∞	$\cdots$	0	c_{n1}	c_{n2}	$\cdots$	$-\infty$
$n+1$	0	∞	$\cdots$	∞	$-$	∞	$\cdots$	∞	0	∞	$\cdots$	∞
$n+2$	∞	0	$\cdots$	∞	∞	$-$	$\cdots$	∞	∞	0	$\cdots$	∞
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$2n$	∞	∞	$\cdots$	0	∞	∞	$\cdots$	$-$	∞	∞	$\cdots$	0
$2n+1$	$-\infty$	c_{21}	$\cdots$	c_{n1}	0	∞	$\cdots$	∞	$-$	∞	$\cdots$	∞
$2n+2$	c_{12}	$-\infty$	$\cdots$	c_{n2}	∞	0	$\cdots$	∞	∞	$-$	$\cdots$	∞
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$3n$	c_{1n}	c_{2n}	$\cdots$	$-\infty$	∞	∞	$\cdots$	0	∞	∞	$\cdots$	$-$

Figure 3.4: Cost matrix of $3n$-vertex symmetric instance from n-vertex asymmetric instance as given by Equation (3.2).

3.2 Lower Bounds for the Asymmetric BTSP

We now discuss some polynomial schemes to compute good quality lower bounds on the optimal objective function value of the BTSP. Ideally a tight lower bound exists such that, in concert with a good heuristic, optimality is established without the cost of an expensive exact BTSP solver. Some of the lower bounds we discuss here are well known, but we present them for the purpose of comparison. Two of these algorithms are also discussed in Chapter 2, but slightly different results present themselves here.

3.2.1 2-Max Bound (2MB)

This simple bound is described in [54, 91]. Simply find the smallest in-edge and out-edge cost incident on every node and select the largest of all these values over all vertices. It is clearly a lower bound for the BTSP, and is computed in $O(m)$ time.

3.2.2 Bottleneck Assignment Problem (BAP) bound

The BAP finds a permutation ϕ of $1, \ldots, n$ such that $\max\{c_{i\phi(i)} : i = 1, \ldots, n\}$ is minimized. i.e.

$$\begin{aligned} &\text{minimize } \max_{i=1,\ldots,n} \{c_{i\phi(i)}\} \\ &\text{subject to } \phi \in \Phi(n) \end{aligned}$$

where $\Phi(n)$ is the set of all permutations of the integers $1, \ldots, n$. The optimal objective function value of the bottleneck assignment problem with cost matrix C is a lower bound for the BTSP on C [17, 37].

One can solve the BAP in $O(m\sqrt{n}\log k)$-time where k is the number of unique costs in C using a unit-capacity max-flow algorithm [2]. The $O(m\sqrt{n}\log k)$-time algorithm is the one we used in our computational experiments. However, the best known algorithm for solving the BAP is by Punnen and Nair with a running time of $O(n^{2.5})$ [87].

3.2.3 Bottleneck Biconnected Spanning Subgraph Problem (BBSSP)

In Chapter 2 we observed that the bottleneck biconnected spanning subgraph problem (BBSSP) is experimentally a strong lower bound for the symmetric BTSP. We refer the reader to Chapter 2 for a full outline of the BBSSP algorithm. A simple implementation of this algorithm, used in our computational experiments, runs in $O(m \log n)$-time. An $O(m + n \log n)$ algorithm by Punnen and Nair [86] and an $O(m)$ algorithm by Manku [73] have better worst case complexity.

The proof of correctness for the BBSSP for the symmetric BTSP relies on the fact that any Hamiltonian cycle in an undirected graph is biconnected. However, a Hamiltonian cycle in a directed graph is *not* biconnected (it is only *strongly* connected). However, there are two ways one can use the BBSSP to generate a valid lower bound on the asymmetric BTSP.

The first way is to create either the $2n \times 2n$ symmetric cost matrix $\bar{C}$, as per Equation (3.1), or a $3n \times 3n$ symmetric cost matrix $\tilde{C}$, as per Equation (3.2), from the $n \times n$ asymmetric cost matrix C. It is easy to show that the BBSSP objective value on $\bar{C}$ and $\tilde{C}$ are identical, and both give a lower bound on the optimal objective function value for the BTSP on C. We denote this lower bound by BBSSP($\bar{C}$).

The second way is to compute the BBSSP on the symmetric relaxation $\hat{C}$ of the asymmetric cost matrix C where $\hat{C} = (\hat{c}_{ij})_{n\times n}$ is defined as

$$\hat{c}_{ij} = \min\{c_{ij}, c_{ji}\}. \tag{3.3}$$

We denote this lower bound by BBSSP($\hat{C}$).

Theorem 2. *The BBSSP($\hat{C}$) is a lower bound on the optimal BTSP objective value on C.*

Proof. Let H be any directed Hamiltonian cycle in C, and let $\hat{H}$ be its undirected version in $\hat{C}$. Clearly,

$$\max_{(i.j)\in H} c_{ij} \geq \max_{(i.j)\in H} \min\{c_{ij}, c_{ji}\} = \max_{(i.j)\in \hat{H}} \hat{c}_{ij} \geq \text{BBSSP}(\hat{C}),$$

so the bound is correct. □

3.2.4 Bottleneck Strongly Connected Spanning Subgraph Problem (BSCSSP) Bound

Since a directed Hamiltonian tour in $G = (V, E)$ is strongly connected, the bottleneck strongly connected spanning subgraph problem (BSCSSP) provides a lower bound on the optimal BTSP objective value on G with cost matrix C. For our experiments we use a simple implementation, discussed in [84], which may be described informally as follows. Given graph $G = (V, E)$ with cost matrix C:

1. Arrange the unique costs of C in non-ascending order $z_1 < z_2 < \cdots < z_k$;
2. Let $l \leftarrow 1$, $u \leftarrow k$;
3. Let $m \leftarrow (u - l)/2 + l$;
4. Let $G^m = (V, E^m)$ where $E^m = \{(i, j) \in E : c_{ij} \leq z_m\}$;
5. If G^m is strongly-connected, set $u \leftarrow m$, otherwise set $l \leftarrow m + 1$;
6. If $l < u$, go to step 3, otherwise terminate and return z_m.

Checking a graph for strong-connectivity may be done in $O(m)$-time using a depth-first search, so this simple algorithm runs in $O(m \log k)$, where k is the number of unique costs in

C. Punnen also presents an $O(\min\{m + n \log n, m \log^* n\})$ implementation for the BSCSSP, where $\log^* n$ is the iterative logarithm of n [84].

3.2.5 Bidirectional Bottleneck Path (BBP) Bound

Readers will likely be familiar with shortest-path algorithms for finding the shortest path between any pair of nodes, such as Dijkstra's $O(n^2)$ algorithm [2]. A shortest-path algorithm can be easily modified to solve the *bottleneck-path* problem, where we seek the path P between any pair of vertices where the largest edge cost in P is as small as possible.

Dijkstra's algorithm, in particular, produces a 'shortest-path tree" that gives the shortest path between any node $v \in V$ and every other node in V. Modifying it to solve the bottleneck-path problem produces an analogous 'bottleneck-path tree".

Using the bottleneck-path problem, Carpaneto, Martello and Toth [17] construct a lower bound for the asymmetric BTSP using two bottleneck-path computations as follows:

1. Pick any node $v \in V$;
2. Let T_1 be the bottleneck path tree *from* v to all other nodes in V;
3. Let T_2 be the bottleneck path tree *to* v from all other nodes in V;
4. Return $\max\{c_{ij} : (i,j) \in T_1 \cup T_2\}$.

This algorithm clearly runs in $O(n^2)$-time. It is easy to show that this bound is equivalent to the BSCSSP bound.

Lemma 1. *The BBP bound and the BSCSSP bounds are equivalent.*

Proof. Let $z_a = \text{BSCSSP}(G, C)$ and $z_b = \text{BBP}(G, C)$ be the optimal BSCSSP and BBP objective values, respectively, on a graph $G = (V, E)$ with cost matrix C. We will show $z_a = z_b$, else the optimality of one or the other is contradicted.

Suppose $z_a < z_b$. Let $\bar{G} = (V, \bar{E})$ be the strongly-connected subgraph found by the BSCSSP. For any vertex $u \in V$ let T_1 (T_2) be the bottleneck path tree from u (to u) to (from) every $v \in V \setminus \{u\}$ in $\bar{G}$. Clearly,

$$\max\{c_{ij} : (i,j) \in T_1 \cup T_2\} \leq max\{c_{ij} : (i,j) \in \bar{E}\} = z_a$$

and so the BBP has returned a suboptimal solution.

Similarly, if $z_b < z_a$, then let T_1 and T_2 be the pair of trees found in the BBP algorithm. Clearly, $T_1 \cup T_2$ is a spanning, strongly-connected subgraph of G. As $\max\{c_{ij} : (i,j) \in T_1 \cup T_2\} = z_b$, the BSCSSP has returned a suboptimal solution.

It follows that only when $z_a = z_b$ are these contradictions avoided. □

3.2.6 Strengthening the Lower Bounds

We now show how any of the lower bounds discussed so far can be strengthened by a simple polynomial scheme. For any vertex $i \in V$, pick any outedge $(i,j) \in E$ and any inedge $(k,i) \in E$. Now consider the graph G^i_{jk} obtained by deleting all arcs incident on i expect (i,j) and (k,i). For any lower bound discussed so far let β^i_{jk} be a lower bound for the BTSP on G^i_{jk}. Define

$$\beta^i = \min\{\beta^i_{jk} : j,k \in V \setminus \{i\}, j \neq k\}. \tag{3.4}$$

Theorem 3. $\beta = \max_{i \in V} \beta^i$ *is a lower bound for the BTSP.*

Proof. Recall that $\Pi(G)$ denotes the set of Hamiltonian cycles in a graph G. Similarly, we let $\Pi(G^i_{jk})$ denote the set of Hamiltonian cycles in the subgraph G^i_{jk}. The set $\Pi(G^i_{jk})$ can alternatively be thought of as the collection of Hamiltonian cycles in G that use the edges (i,j) and (k,i).

It is clear for any i if we collect all the Hamiltonian cycles in $\Pi(G^i_{jk})$ for all j,k that we end up with the entire collection of Hamiltonian cycles in G. That is to say for any $i \in V$,

$$\Pi(G) = \bigcup_{j,k \in V, i \neq j \neq k} \Pi(G^i_{jk}). \tag{3.5}$$

By definition,

$$\beta^i_{jk} \leq \max\{c_{pq} : (p,q) \in H\} \text{ for all } H \in \Pi(G^i_{jk}) \tag{3.6}$$

In view of (3.5) and (3.6) we have

$$\beta^i \leq \max\{c_{pq} : (p,q) \in H\} \text{ for all } H \in \Pi(G). \tag{3.7}$$

Since (3.7) is true for all $i \in V$, we conclude $\beta = \max_{i \in V} \beta^i$ is a valid lower bound on the optimal BTSP objective value. □

Theorem 3 may be used to strengthen any of the lower bounds discussed so far. A naïve implementation of this theorem may be as described by Algorithm 3.1.

Algorithm 3.1: Naïve application of Theorem 3

Input: A graph $G = (V, E)$ with cost matrix C, and a lower bound algorithm α.
Output: A lower bound on the optimal BTSP objective value.
for $i \in V$ **do**
  for $(i, j) \in E$ **do**
    for $(k, i) \in E$ **do**
      `/* Remove all arcs incident on` i `except` (i, j) `and` (k, i) `*/`
      $E^i_{jk} \leftarrow E \setminus \{(i, p) \in E : p \neq j\} \setminus \{(q, i) \in E : q \neq k\}$;
      $G^i_{jk} \leftarrow (V, E^i_{jk})$;
      $\beta^i_{jk} \leftarrow \alpha(G^i_{jk}, C)$; `/* Calculate lower bound` α `on` G^i_{jk} `and` C `*/`
    end
  end
  $\beta^i \leftarrow \min\{\beta^i_{jk} : j, k \in V \setminus \{i\}\}$;
end
$\beta \leftarrow \max\{\beta^i : i \in V\}$;
return β;

Algorithm 3.1 requires $O(n^3)$ calls to the lower bound algorithm α when G is a complete graph. If α is the BAP lower bound, which has a running time of $O(n^{2.5})$ for each BAP call, then this implementation requires $O(n^{5.5})$ operations. Similarly, it is easy to see if α is the BSCSSP or BBSSP lower bound algorithm that, even with their best known implementations, $O(n^5)$ calculations are necessary.

However, a careful implementation that exploits special properties of the lower bound algorithm may allow one to reduce the complexity significantly. We illustrate this by showing we can apply Theorem 3 with the BBP lower bound (or, equivalently, the BSCSSP lower bound) and identify β with $O(n^{3.792})$ calculations. We call the resulting, strengthened lower bound the *enhanced bidirectional bottleneck path* (EBBP) bound.

The EBBP bound may be described informally as follows: For each node $i \in V$ construct the graph $G^i = G \setminus \{i\}$ (i.e. the graph G with node i removed, along with any arcs incident on i). Solve the *all-pairs bottleneck path problem* on G^i and let P^i be the resulting matrix, where P^i_{jk} is the bottleneck path distance in G^i from vertex j to vertex k. Matrix P^i may be used to easily calculate β^i_{jk}. The largest bottleneck path distance from i to every other vertex is the larger of c_{ij} and the largest entry in row j of P^i. Likewise, the largest bottleneck path distance to i from every other vertex is the larger of c_{ki} and the largest entry of column k in P^i. Thus, β^i can be identified by constructing matrix P^i followed by $O(n^2)$ lookups in

matrix P^i. The value β is set as the largest value of β^i obtained. A formal description of the EBBP algorithm is given in Algorithm 3.2.

Algorithm 3.2: $EBBP(G, C)$

Input: A graph $G = (V, E)$ with cost matrix C.
Output: A lower bound on the BTSP objective value.
for $i \in V$ **do**
  $G^i \leftarrow (V \setminus \{i\}, E \setminus \{(u, v) \in E : u = i \text{ or } v = i\})$; `/* remove i from G */`
  $P^i \leftarrow$ all-pairs-bottleneck-paths(G^i, C)
  for $(i, j) \in E$ **do**
    $\alpha^i_j \leftarrow \max\{P^i_{jl} : l \in V \setminus \{i, j\}\}$; `/* max bottleneck edge from` j `*/`
    for $(k, i) \in E$ **do**
      $\gamma^i_k \leftarrow \max\{P^i_{lk} : l \in V \setminus \{i, k\}\}$; `/* max bottleneck edge to` k `*/`
      $\beta^i_{jk} \leftarrow \max\{\alpha^i_j, \gamma^i_k, c_{ij}, c_{ki}\}$;
    end
  end
  $\beta^i \leftarrow \min\{\beta^i_{jk} : j, k \in V \setminus \{i\}\}$;
end
$\beta \leftarrow \max\{\beta^i : i \in V\}$;
return β;

Theorem 4. *The EBBP algorithm correctly identifies the lower bound β in $O(n^{3.792})$-time when β^i_{jk} is the BBP lower bound in G^i_{jk}.*

Proof. Let T_1 be the tree of bottleneck paths from node i to all other nodes in G^i_{jk}. Similarly, let T_2 be the tree of bottleneck paths from all nodes in G^i_{jk} to node i. Clearly,

$$\beta^i_{jk} = \max\{c_{pq} : (p, q) \in T_1 \cup T_2\}. \tag{3.8}$$

Note that P^i is the all-pairs bottleneck path matrix on $G^i = G \setminus \{i\}$. The jth row of P^i gives the bottleneck distances from node j to all other nodes in G^i. Define α^i_j as

$$\begin{aligned} \alpha^i_j &= \max\{P^i_{jl} : l \in V \setminus \{i, j\}\} \\ &= \max\{c_{pq} : (p, q) \in T_1 \setminus \{(i, j)\}\} \end{aligned}$$

Similarly, the kth column of P^i gives the bottleneck distances from each node l of G^i to

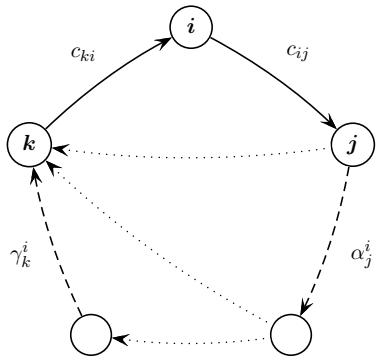


Figure 3.5: Lower bound construction with EBBP bound

node k. Define γ_k^i as

$$\begin{aligned}\gamma_k^i &= \max\{P_{lk}^i : l \in V \setminus \{i,k\}\} \\ &= \max\{c_{pq} : (p,q) \in T_2 \setminus \{(i,k)\}\}.\end{aligned}$$

Hence, from Equation (3.8) we have

$$\beta_{jk}^i = \max\{\alpha_j^i, \gamma_k^i, c_{ij}, c_{ki}\}.$$

Refer to Figure 3.5 for a diagram of this construction. We conclude the algorithm correctly computes β_{jk}^i, β^i and hence β.

To analyze the complexity, note that P^i can be identified in $O(n^{2.792})$-time for each $i \in V$ using Vassilevska et al.'s algorithm for solving the all-pairs bottleneck paths problem [109]. All other computations for fixed i takes $O(n^2)$ time. Since these computations are repeated for each $i \in V$, the overall complexity of the algorithm is $O(n^{3.792})$. □

Vassilevska et al.'s $O(n^{2.792})$-time algorithm for solving the all-pairs bottleneck path problem is complex to implement, so instead we use a modified Floyd-Warshal all-pairs shortest path algorithm instead (see Ahuja et al. for a formal description [2]). The Floyd-Warshal algorithm has a worst case complexity of $O(n^3)$ to compute the matrix P^i. Using this in Algorithm 3.2 results in a moderately higher complexity of $O(n^4)$, but with the advantage of easy implementation.

3.2.7 Analysis of the Lower Bounds

To analyze the relative strengths of the lower bounds discussed so far we introduce some definitions. Let A and B be any two BTSP lower bound algorithms. If A always computes an objective value greater than or equal to the one produced by B, then A *dominates* B, which is to say algorithm B will never produce a superior lower bound value than algorithm A. Likewise, if there exists instances where A produces a superior lower bound value compared to B and vice versa, then A and B are *non-dominated* with respect to each other. Finally, two lower bounds A and B are equivalent if they always return the same objective value for any instance. We noted in Lemma 1 that the BSCSSP and BBP bounds are equivalent.

Theorem 5. *The BSCSSP lower bound (equivalently, the BBP lower bound) dominates the BBSSP($\bar{C}$) bound.*

Proof. Let δ be the optimal objective value of the BSCSSP bound on a graph $G = (V, E)$ with cost matrix C. Construct a directed graph $\vec{G}^{\delta} = (V, E^{\delta})$ with $E^{\delta} = \{(i, j) \in E : c_{ij} \leq \delta\}$. By construction, $\vec{G}^{\delta}$ is strongly-connected.

Now consider the undirected graph constructed from $\vec{G}^{\delta}$ using the $2n$-node symmetric transformation of Equation (3.1). Discard all edges of infinite cost from this graph and call the resulting graph $\bar{G}$. Note that $\bar{G}$ is bipartite with vertex sets $V_1 = \{1, 2, \ldots, n\}$ and $V_2 = \{n + 1, n + 2, \ldots, 2n\}$, and has no edges of cost more than δ. It suffices to show that $\bar{G}$ is biconnected.

Suppose $\bar{G}$ is not biconnected. If it is not biconnected, then it must contain a cut-vertex r. Suppose $r \in V_1$. Let G_1 and G_2 be two connected components of $\bar{G} \setminus \{r\}$. Then there exist vertices $(u + n) \in G_1$ and $(v + n) \in G_2$ such that edges $(r, u + n)$ and $(r, v + n)$ are in $\bar{G}$. By construction of $\bar{G}$, $\vec{G}^{\delta}$ must contain edges (r, u) and (r, v). Since $\vec{G}^{\delta}$ is strongly connected, it must contain a path from u to r and a path from v to r. Thus, there must exist a path in $\bar{G}$ from u to $(r+n)$ that does not contain r and a path from v to $(r+n)$ that does not contain r. Since r is a cut-vertex, both u and v must be in the same component. Since $u+n$ and $v+n$ are in different components, u and v are in the same component, and $(u, u + n)$ and $(v, v + n)$ are edges of $\bar{G}$, this shows that a cut vertex cannot belong to the set V_1.

The same logic shows that a cut vertex cannot belong to the set V_2, thus $\bar{G}$ must be biconnected. Since the optimal objective value of the BBSSP($\bar{C}$) bound cannot be greater than δ, this proves the BSCSSP dominates. □

Theorem 6. *The BAP, BBSSP($\bar{C}$), BSCSSP, BBP, and EBBP bounds dominate the 2MB.*

Proof. The 2MB selects the in-edge and out-edge of every vertex of *minimum cost*. If we can show that at least one in-edge and one out-edge is selected by the other lower bound algorithms for every vertex, then we prove the 2MB is dominated.

Recall that the BAP generates a cycle cover S of G. Since S contains all vertices of G, and each vertex has an incoming edge and an outgoing edge, the 2MB is dominated.

For a strongly connected graph, each vertex have at least one incoming edge and at least one outgoing edge. Since the solutions returned by the BSCSSP, BBP, and EBBP bounds are strongly connected, they all dominate the 2MB.

Finally, consider the optimal solution $\bar{G}$ on $2n$ vertices to the BBSSP($\bar{C}$) bound. As $\bar{G}$ is biconnected, each vertex in $\bar{G}$ will have a degree of at least 2. One of the edges incident on every vertex will be the edge of cost $-\infty$, but the other will represent either an out-edge (for $i = 1, 2, \ldots, n$) or an in-edge (for $i = n+1, n+2, \ldots, 2n$) for each of the n vertices in the original asymmetric instance. Therefore, the BBSSP($\bar{C}$) bound includes at least one in-edge and one out-edge for every vertex, and thus dominates the 2MB. □

Theorem 7. *The EBBP lower bound dominates the 2MB, BSCSSP, BBP, BBSSP($\bar{C}$), and BBSSP($\hat{C}$) lower bounds.*

Proof. As the EBBP lower bound is strongly-connected and obtained by additional restrictions to the BBP bound, it will dominate the BBP bound, and thus the BSCSSP, BBSSP($\bar{C}$), and 2MB bounds in light of Lemma 1 and Theorems 5 and 6.

We now show the BBSSP($\hat{C}$) bound is dominated by contradiction. Let $G = (V, E)$ with cost matrix C be an instance where the EBBP objective value is strictly less than the BBSSP($\hat{C}$) bound.

Let β be the optimal EBBP objective value on a graph G with cost matrix C. Construct the directed graph $G^\beta = (V, E^\beta)$ where $E^\beta = \{(i,j) \in E : c_{ij} \leq \beta\}$. G^β is clearly strongly connected, but must have a cut-vertex i that would disconnect the graph (otherwise, the EBBP objective value would be at least equal to the BBSSP($\hat{C}$) objective value, and i would not exist).

For $(i,j), (k,i) \in E$, let G^i_{jk} be the subgraph of G with all edges incident on i removed except for (i,j) and (k,i). Recall that the EBBP algorithm calculates the value β^i_{jk}, which is the BBP objective value on G^i_{jk}. Let $\beta^i = \min\{\beta^i_{jk} : j, k \in V \setminus \{i\}\} = \beta^i_{uv}$. By definition, $\beta = \max\{\beta^i : i \in V\} \geq \beta^i_{uv}$, and thus G^i_{uv} is strongly connected. Let u and v be two

Bound	C_1	C_2	C_3	C_4
2MB	1	1	1	3
BAP	3	1	2	3
BBSSP($\hat{C}$)	1	2	2	1
BBSSP($\bar{C}$)	1	2	1	3
BSCSSP	1	2	1	3
EBBP	2	3	2	3
BTSP	3	3	3	3

Table 3.1: Lower bound values for cost matrices given in Theorem 8.

vertices in each of the connected components of $G^i \setminus \{i\}$. It must follow that there is no path between u and v in G^i_{uv}, contradicting its strong connectivity.

Therefore, no cut vertex in G^β exists, and β must be at least equal to the objective value of the BBSSP($\hat{C}$) bound. □

Theorem 8. *The following pairs of bounds are non-dominated:*

(a) The BAP bound and any of the bounds BBSSP($\hat{C}$), BBSSP($\bar{C}$), BSCSSP, BBP, or EBBP;

(b) The BSCSSP or BBP bound and the BBSSP($\hat{C}$) bound;

(c) The BBSSP($\hat{C}$) bound and the BBSSP($\bar{C}$) bound; and

(d) The 2MB and the BBSSP($\hat{C}$) bounds.

Proof. Consider the follow cost matrices:

$$C_1 = \begin{bmatrix} - & 3 & 9 & 1 & 1 \\ 2 & - & 1 & 9 & 9 \\ 9 & 9 & - & 1 & 9 \\ 9 & 9 & 9 & - & 1 \\ 1 & 1 & 9 & 9 & - \end{bmatrix}, C_2 = \begin{bmatrix} - & 2 & 9 & 9 & 1 \\ 2 & - & 1 & 9 & 9 \\ 9 & 9 & - & 1 & 9 \\ 9 & 1 & 9 & - & 3 \\ 1 & 9 & 9 & 9 & - \end{bmatrix}, C_3 = \begin{bmatrix} - & 1 & 3 & 3 & 3 \\ 3 & - & 1 & 2 & 3 \\ 1 & 3 & - & 3 & 1 \\ 3 & 3 & 1 & - & 3 \\ 2 & 3 & 3 & 1 & - \end{bmatrix}, C_4 = \begin{bmatrix} - & 3 & 3 & 1 & 1 \\ 3 & - & 3 & 1 & 3 \\ 2 & 1 & - & 1 & 3 \\ 3 & 3 & 3 & - & 3 \\ 3 & 3 & 1 & 1 & - \end{bmatrix}.$$

Table 3.1 provides optimal objective values for each lower bound algorithm on each of these four cost matrices. Statement (a) is true by cost matrices C_1 and C_2. The remaining statements are true by cost matrices C_3 and C_4. □

	2MB	BAP	BBSSP($\bar{C}$)	BBSSP($\hat{C}$)	BSCSSP	BBP	EBBP	Complexity
2MB	=	▲	▲	✓	▲	▲	▲	$O(n^2)$
BAP	◀	=	✓	✓	✓	✓	✓	$O(n^{2.5})$
BBSSP($\bar{C}$)	◀	✓	=	✓	▲	▲	▲	$O(n^2)$
BBSSP($\hat{C}$)	✓	✓	✓	=	✓	✓	▲	$O(n^2)$
BSCSSP	◀	✓	◀	✓	=	=	▲	$O(n^2)$
BBP	◀	✓	◀	✓	=	=	▲	$O(n^2)$
EBBP	◀	✓	◀	◀	◀	◀	=	$O(n^{3.792})$

Table 3.2: Comparison of lower bounds

Table 3.2 summarizes our discussion. A '◀' in the table indicates the bound representing the row dominates the bound representing the column. A '▲' in the table indicates the bound representing the column dominates the bound representing the row. An '=' sign indicates the two bounds are equivalent, and a '✓' indicates they are non-dominated.

3.3 Approximation Algorithm for the Asymmetric BTSP

Recall that an ϵ-approximation is a polynomial-time algorithm that is guaranteed to return a solution with objective value z that is at most ϵ-times larger than the optimal objective value z^*, which is to say

$$z \leq \epsilon \cdot z^*. \tag{3.9}$$

In Chapter 1, we discuss progress on approximation algorithms for the TSP, noting, unless P=NP, no ϵ-approximation exists for any ϵ, $1 \leq \epsilon < \infty$ [96]. Progress on approximation algorithms for the TSP can only be made when we restrict our attention to cost matrices satisfying specific conditions.

For example, let us restrict our attention to cost matrices that satisfy the *triangle inequality*, i.e.:

$$c_{ij} \leq c_{ik} + c_{kj} \text{ for all } i, j, k. \tag{3.10}$$

The best known ϵ-approximation algorithm for symmetric cost matrices satisfying (3.10) is Christofides' $\frac{3}{2}$-approximation [20]. Likewise, $O(\log n)$-approximations exist for asymmetric

cost matrices satisfying (3.10) [33, 12, 56]. While it is unknown if these are the best ϵ-approximations possible for the symmetric and asymmetric TSP, respectively, the results suggest the asymmetric TSP is much harder to approximate.

The existence of an ϵ-approximation algorithm for the BTSP on a general cost matrix is as difficult as that for the TSP as Parker and Rardin proved in 1982 [79].

Theorem 9. *[79] Unless $P = NP$, there is no polynomial time ϵ-approximation algorithm for the BTSP for any constant ϵ, $1 \leq \epsilon < \infty$.*

As with the TSP we can instead restrict our attention to cost matrices that satisfy the triangle inequality.

A 2-approximation for the symmetric BTSP satisfying the triangle inequality

For symmetric cost matrices, a simple, well-known 2-approximation for the BTSP exists when the cost matrix satisfies the triangle inequality, Equation (3.10). This 2-approximation relies on the concept of *powers of graphs*.

Definition 2. The t^{th} power of a (not necessarily complete) graph G is the graph $G^t = (V, E^t)$, where $(u, v) \in E^t$ whenever a path from u to v exists in G with at most t edges.

The 2-approximation for the symmetric BTSP may be informally described as follows:

1. Let δ be the optimal BBSSP objective value on $G = (V, E)$ with cost matrix C.
2. Construct $\bar{G} = (V, \bar{E})$ where $\bar{E} = \{(i, j) \in E : c_{ij} \leq \delta\}$.
3. Output any Hamiltonian cycle in $\bar{G}^2$.

Fleischner showed the 'square' of any biconnected graph (i.e. $t = 2$) is Hamiltonian [32], and Lau gave an $O(n^2)$-algorithm for finding a Hamiltonian cycle in the square of a biconnected graph [66, 67] (Rardin and Parker independently proved the same result in [93]). By the triangle inequality (3.10) one can easily show the largest cost in $\bar{G}^2$ is at most twice the optimal BTSP objective value, hence a 2-approximation for symmetric BTSP (we provide a proof of this in a moment).

We can generalize the triangle inequality as follows: let τ be any constant where $\tau \geq \frac{1}{2}$, and define the *τ-triangle inequality* to be

$$c_{ij} \leq \tau(c_{ik} + c_{kj}) \text{ for all } i, j, k. \tag{3.11}$$

Clearly, when $\tau = 1$, Equation (3.11) equivalent to Equation (3.10). When $\tau = \frac{1}{2}$, Equation (3.11) forces all costs c_{ij} to assume the same value. One can view the τ-triangle inequality as a relaxation of the triangle inequality when $\tau > 1$, and as a restriction of the triangle inequality when $\frac{1}{2} < \tau < 1$.

Theorem 10. *Let C be the cost matrix associated with a complete, directed graph G satisfying the τ-triangle inequality for some $\tau > \frac{1}{2}$, and let H^* be an optimal solution to the BTSP on G such that $z^* = \max\{c_{ij} : (i,j) \in H^*\}$. Let S be a spanning subgraph of G such that $z = \max\{c_{ij} : (i,j) \in S\}$, and $z \leq z^*$. If the graph S^t, $t \in 1, \ldots, n$, contains a Hamiltonian cycle H, then*

$$\frac{z}{z^*} = \begin{cases} t & \text{if } \tau = 1, \\ \frac{\tau}{\tau-1}(2\tau^{t-1} - \tau^{t-2} - 1) & \text{if } \tau > 1, \\ \frac{\tau}{\tau-1}(\tau^{t-1} + \tau - 2) & \text{if } \tau < 1. \end{cases}$$

Kabadi and Punnen provide a proof of Theorem 10 when the graph is undirected [54]. The proof is almost identical when the graph is directed, hence we skip a detailed proof.

Theorem 10 shows the 2-approximation for symmetric BTSP is correct when $\tau = 1$. Indeed, the algorithm generalizes to a 2τ-approximation for the symmetric BTSP for any $\tau \geq \frac{1}{2}$. We finish with the following theorem.

Theorem 11. *Unless P=NP, there is no polynomial $2\tau - \delta$ approximation algorithm for the symmetric BTSP on a complete graph with edge costs satisfying the τ-triangle inequality for any $\delta > 0$ and $\tau > \frac{1}{2}$.*

Doroshko and Sarvanov [27], Parker and Rardin [79], and Hochbaum and Shmoys [48] all provide a proof of Theorem 11 when $\tau = 1$. Kabadi and Punnen prove the theorem for a general $\tau > \frac{1}{2}$ [54]. Theorem 11 shows, unless P=NP, this 2τ-approximation for the symmetric BTSP is the best possible.

Table 3.3 outlines the best-known ϵ-approximation algorithms for the TSP, the Max-TSP, the BTSP, and the MSTSP for problems with complete cost matrices satisfying the triangle inequality. In addition, the table shows the best known ϵ-approximation algorithms for the Max-TSP on general complete cost matrices. There are currently no known ϵ-approximation algorithms for the asymmetric BTSP and asymmetric MSTSP. For more detail on the progress into approximation algorithms for the TSP and Max-TSP, please refer to Table 1.1 in Chapter 1.

Problem	Symmetric		Asymmetric	
△-TSP	3/2	[20]	$0.841 \log n$	[56]
Max-TSP	7/9	[77]	2/3	[56]
△-Max-TSP	7/8	[64]	31/40	[13]
△-BTSP	2	[93]	(Open)	
△-MSTSP	2	[6]	(Open)	

Table 3.3: Best-known ϵ-approximation algorithms for the TSP, Max-TSP, BTSP, and MSTSP, where "△" denotes the problem on complete cost matrices satisfying the triangle inequality. Please note the 2-approximation for the symmetric BTSP was also proved independently by [32, 66, 67].

Approximation for the asymmetric BTSP satisfying the triangle inequality

The approximation algorithm for the asymmetric BTSP we now present is inspired by the 2τ-approximation for the symmetric BTSP. As before, let us assume our cost matrix C satisfies the τ-triangle inequality (3.11) for some $\tau > \frac{1}{2}$. Instead of solving the BBSSP, we solve the bottleneck strongly connected spanning subgraph problem (BSCSSP). We call the algorithm Approx-BTSP, and describe it in Algorithm 3.3.

Algorithm 3.3: Approx-BTSP(G, C)

Input: A graph $G = (V, E)$ with cost matrix C.
Output: a Hamiltonian Cycle in G.
$S \leftarrow$ BSCSSP(G, C) ; `/* See Section 3.2.4 */`
Compute S^t for $t = \lceil \frac{n}{2} \rceil$;
Let H be any Hamiltonian cycle in S^t;
return H;

We establish the complexity and performance ratio of algorithm Approx-BTSP using the following well known theorem of Ghouilà-Houri [38].

Theorem 12. *[38] If G is a directed graph on n vertices and $\min\{\delta^+(v), \delta^-(v)\} \geq \frac{n}{2}$ for every vertex $v \in G$, then G is Hamiltonian.*

Theorem 13. *Algorithm Approx-BTSP runs in polynomial time and guarantees an ϵ-approximate solution for the asymmetric BTSP whenever the edge-costs satisfy the τ-triangle*

inequality, where $t = \lceil \frac{n}{2} \rceil$ *and*

$$\epsilon = \begin{cases} t & \text{if } \tau = 1, \\ \frac{\tau}{\tau - 1}(2\tau^{t-1} - \tau^{t-2} - 1) & \text{if } \tau > 1, \\ \frac{\tau}{\tau - 1}(\tau^{t-1} + \tau - 2) & \text{if } \tau < 1. \end{cases} \tag{3.12}$$

Proof. Let H^* be an optimal solution to the BTSP on G where $z^* = \max\{c_{ij} : (i,j) \in H^*\}$. Since S is a bottleneck strongly connected spanning subgraph of G, we have $z = \max\{c_{ij} : (i,j) \in S\} \leq z^*$. Thus, by Theorem 10, the performance ratio holds.

We now show that the algorithm is polynomially bounded. The BBSSP can be computed in $O(n^2)$-time [84]. $S^{\lceil \frac{n}{2} \rceil}$ can be obtained in $O(n^3)$ time using an all-pairs shortest path algorithm with unit edge costs. Schaar and Wojda [98] guarantee that $S^{\lceil \frac{n}{2} \rceil}$ satisfies the conditions of Theorem 12, and Bondy and Thomassen [14] give an $O(n^4)$ algorithm for finding a Hamiltonian cycle in graphs satisfying Theorem 12. Thus a Hamiltonian cycle in $S^{\lceil \frac{n}{2} \rceil}$ is obtained in polynomial time. □

3.4 Heuristic Algorithm for the Asymmetric BTSP

In Chapter 2 we discuss a heuristic algorithm for the symmetric BTSP. This algorithm can appropriately modified to solve the asymmetric BTSP using the symmetric transformation of Section 3.1 and the lowering bounding schemes of Section 3.2. The notable difference between theoretical approximability of the symmetic and asymmetric versions of the BTSP necessitates a systematic experimental analysis to understand the practical level of difficulty in solving the asymmetric BTSP in comparison to the symmetric BTSP.

For clarity, let us reintroduce the BTSP heuristic algorithm of Chapter 2 in the context of the asymmetric BTSP. The key ingredient of this algorithm is a feasibility test: '*Given an integer* δ *and a complete directed graph* G*, determine if* G *has a Hamiltonian tour whose largest cost is no more than* δ *and, if yes, produce such a Hamiltonian tour in* G." As this is an NP-hard problem, we explore ways to solve this heuristically.

Consider the cost matrix $C^\delta = (c^\delta_{ij})_{n \times n}$ where

$$c^\delta_{ij} = \begin{cases} 0 & \text{if } c_{ij} \leq \delta, \\ c_{ij} & \text{otherwise.} \end{cases} \tag{3.13}$$

Suppose we solve the asymmetric TSP on G with cost matrix C^δ. The feasibility test has an 'yes' answer if and only if the optimal objective function value of the TSP is zero. Solving the TSP using a heuristic lets us answer the feasibility test in an approximate way. There are several ways to improve the accuracy of this approximation. One could employ different TSP heuristics on cost matrix C^δ. A more reasonable way is to apply the best known TSP heuristic on different, but equivalent, cost matrices.

One way we can generate equivalent cost matrices is to introduce some randomness as follows. Let $z_1 < z_2 < ... < z_k$ be an ascending arrangement of distinct costs in C. Generate positive, random integers $r_1 < r_2 < ... < r_k$ in an interval $[a, b]$. Now define the cost matrix $C^{\delta,r} = (c_{ij}^{\delta,r})_{n\times n}$ where

$$c_{ij}^{\delta,r} = \begin{cases} 0 & \text{if } c_{ij} \leq \delta, \\ z_l + r_l & \text{otherwise, where } c_{ij} = z_l. \end{cases} \tag{3.14}$$

The TSP with cost matrix C^δ has an optimal tour of zero cost if and only if the TSP with cost matrix $C^{\delta,r}$ has an optimal tour of zero cost. If a non-zero cost tour is constructed by the TSP heuristic, a new matrix $C^{\delta,r}$ can be generated with a new set of random numbers. The TSP heuristic can then be applied on the new cost matrix (we call this a 'shake' operation). The process can be repeated a prescribed number of times, or 'shakes". If no zero-length tour can be found after a prescribed number of shake operations, we conclude with high probability that the answer to the feasibility test is 'no'.

The construction of $C^{\delta,r}$ is designed to discourage using edges of large cost in the solution produced by the TSP heuristic. Each time we solve a TSP using a heuristic, the bottleneck value of the resulting tour is also noted and, upon termination, the best such tour is returned. A formal description of this 'feasibility test' procedure is summarized in Algorithm 3.4. For our experiments we selected the interval $[a, b]$ as $[1, n^2]$.

Let H^* be any heuristic solution for an asymmetric BTSP instance where $U = \max\{c_{ij} : (i, j) \in H^*\}$. Further, let L be the objective value of any lower bound discussed in Section 3.2. We can attempt to find a better BTSP tour by performing a binary search over edge costs in the range $[L, U]$ using the feasibility test of Algorithm 3.4. The resulting heuristic algorithm is formally described in Algorithm 3.5.

Any asymmetric TSP heuristic α may be used in algorithms 3.4 and 3.5. Further, any symmetric TSP heuristic may also be used after applying one of the two transformations

Algorithm 3.4: IsFeasible$(n, C, \delta, \alpha, p, q)$

Input: A problem on n nodes with cost matrix C, integer δ, TSP solver/heuristic α, and integers p and q, which represent the number of iterations with cost matrix C^{δ} and $C^{\delta,r}$, respectfully.

Output: The 3-tuple $(feasible, tour, max\_cost)$ where $feasible$ is a Boolean value that indicates if a Hamiltonian cycle was found using only costs less than or equal to δ, $tour$ is the feasible/best tour found, and max_cost is largest cost in $tour$.

$minmax\_cost \leftarrow \infty$;
$best\_tour \leftarrow \emptyset$;
for i = *1...p* **do**
  $(length, tour) \leftarrow \alpha(n, C^{\delta})$;
  $max\_cost \leftarrow \max\{c_{ij} : (i,j) \in tour\}$;
  if $length = 0$ **then**
    return $(TRUE, tour, max\_cost)$;
  else
    if $max\_cost < minmax\_cost$ **then**
      $minmax\_cost \leftarrow max\_cost$;
      $best\_tour \leftarrow tour$;
    end
  end
end
for i = *1...q* **do**
  Let $r \leftarrow \{r_1, r_2, ..., r_k\}$ be a list of random integers such that $1 \leq r_1 < r_2 < ... < r_k \leq n^2$;
  $(length, tour) \leftarrow \alpha(n, C^{\delta})$;
  $max\_cost \leftarrow \max\{c_{ij} : (i,j) \in tour\}$;
  if $length = 0$ **then**
    return $(TRUE, tour, max\_cost)$;
  else
    if $max\_cost < minmax\_cost$ **then**
      $minmax\_cost \leftarrow max\_cost$;
      $best\_tour \leftarrow tour$;
    end
  end
end
return $(FALSE, best\_tour, minmax\_cost)$;

Algorithm 3.5: BTSPThreshold(n, C, l, α, p, q)

Input: A problem on n nodes with cost matrix C, a lower bound l, TSP solver/heuristic α, and integers p and q, which represent the number of iterations with cost matrix C^{δ} and $C^{\delta,r}$, respectfully.

Output: The (optimal/heuristic conclusion) on the BTSP objective value and tour.

$(feasible, best\_tour, max\_cost) \leftarrow IsFeasible(n, C, l, \alpha, p, q)$;
if $feasible$ **then return** $(l, best\_tour)$;

Let $z_1 < z_2 < ... < z_k$ be a list of the unique ordered costs from C in non-increasing order;
$low \leftarrow l$; $high \leftarrow k$;
while $low \neq high$ **do**
  $med \leftarrow ((high - low)/2) + low$;
  $(feasible, tour, max\_cost) \leftarrow IsFeasible(n, C, z_{med}, \alpha, p, q)$;
  if $feasible$ **then**
    $high \leftarrow median$;
    $best\_tour \leftarrow tour$;
  else
    $low \leftarrow median + 1$;
  end
end
return $(z_{low}, best\_tour)$;

described in Section 3.1. For our computational study, we used Concorde's implementation of the Lin-Kernighan algorithm [4] after applying the $2n$-vertex transformation of Equation (3.1). The Lin-Kernighan heuristic should naturally force the fixed edges of cost $-\infty$ into the tour with no additional constraint necessary, but we make sure these edges are present to ensure any tour found in the symmetric instance is valid for the asymmetric instance.

3.5 Computational Results

We implemented the lower bounding schemes discussed in Section 3.2 and the heuristic algorithm described in Section 3.4 in C, compiled with the GNU C compiler. We tested this code on 86 benchmark asymmetric TSP instances commonly studied in literature. These instances are as follows:

(a) 42 instances by Cirasella, Johnson, McGeoch, and Zhang that simulate real-world applications in various fields, as described in [21]. There are seven groups of problems, each including five instances of 100 nodes and a single instance of 316 nodes:

- coin: Pay phone coin collection instances
- crane: Random Euclidean stacker crane instances
- disk: Disk drive instances
- rtilt: Tilted drilling machine instances with additive norm
- shop: No-wait flow shop instances
- stilt: Tilted drilling machine instances with sup norm
- super: Approximate shortest common superstring instances

(b) 5 scheduling instances generated by Balas that simulate an application in a Dupont chemical plant. There are five problems in total of sizes 84, 108, 120, 160, and 200 vertices [9];

(c) all 27 TSPLIB instances maintained by Reinelt [95]. We subdivide them into the problems labelled 'ftv" and those that are not, as well as subdivide them into problems with 100 nodes or less and problems with more than 100 vertices.

Problem Set (# of Problems)	2MB	BAP	BBSSP($\bar{C}$)	BBSSP($\hat{C}$)	BSCSSP	EBBP
coin (6)	0	0	6	1	1	6
crane (6)	4	4	0	4	6	6
disk (6)	6	6	0	5	6	6
rtilt (6)	2	2	0	2	6	6
shop (6)	5	5	0	5	6	6
stilt (6)	2	3	0	2	5	6
super (6)	6	6	5	6	6	6
balas (5)	0	0	0	0	5	5
tsplib: no ftv, ≤ 100 nodes (6)	1	1	2	2	5	6
tsplib: no ftv, > 100 nodes (4)	0	4	0	0	0	0
tsplib: ftv, ≤ 100 nodes (9)	7	7	1	7	8	9
tsplib: ftv, > 100 nodes (8)	2	8	0	2	2	3
ftv180 (1)	0	1	0	0	0	0
uk66 (1)	0	0	1	0	0	1
ran500 (5)	5	5	0	5	5	5
ran1000 (5)	5	5	0	5	5	5
Total (86)	45	57	15	46	66	76

Table 3.4: Asymmetric BTSP lower bound summary on problem groups. The number listed under each bound name is the number of problems that bound gave the tightest lower bound in that problem group.

(d) 2 real-world instances (ftv180 and uk66) and 5 random instances with integer costs uniformly generated in the range $[1, 1000]$ of 500 nodes and 1000 vertices each. These were created by Fischetti et al. citefischetti2002atsp.

All computational experiments in this section were carried out on PC with a 3.40GHz Pentium 4 CPU and 2GB of RAM running Microsoft Windows XP SP2 operating system and Cygwin NT 5.1. All reported running times are in CPU seconds rounded to two decimal places and include input and output times.

Results on lower bounding schemes

Each of the 86 problems are grouped into 16 groups for a compact presentation of the results in Table 3.4. The number given for each bound and problem group indicates the number of problems in that group for which the lower bound achieved the tightest result.

With the exception of the EBBP bound, all other bounds generally take less than one second to run, even on problems of 1000 vertices. As expected, the EBBP bound, being an $O(n^4)$ algorithm, is expensive to calculate, and did not provide a tighter lower bound than any of the other lower bounds we tested on our problem set.

Results on heuristic

Algorithm 3.5 was tested on the same problem set in two experiments to observe the effects of 'shaking" (i.e. using cost matrix $C^{\delta,r}$ versus cost matrix C^{δ}). For one experiment, we set $p = 10$ and $q = 0$, corresponding to 10 attempts with cost matrix C^{δ} and 0 attempts with cost matrix $C^{\delta,r}$, respectively; the second we set $p = 5$ and $q = 5$.

The goal is to determine if splitting the effort between both cost matrix formulations produces superior results as opposed to simply using cost matrix C^{δ}. We make no claim that these values are the optimal choices for p and q, but these choices appear to be reasonable for the problems in our test set.

In both experiments we set l equal to the strongest lower bound computed (i.e. the largest optimal objective value of any lower bound algorithm discussed in Section 3.2). We also use Concorde's implementation of the Lin-Kernighan algorithm [4] with the $2n$-vertex symmetric instance constructed using Equation (3.1) from Section 3.1 for our asymmetric TSP heuristic α. We call Concorde's Lin-Kernighan algorithm with the default parameters and five random restarts.

We can easily verify optimality for a given instance if our heuristic returns a solution equal to a valid lower bound for the instance. For problems where the best found solution is not equal to an asymmetric BTSP lower bound value we instead replace the Lin-Kernighan heuristic in Algorithm 3.5 with Concorde's exact TSP solver. In one instance (`stilt316.10`) we are unable to verify optimality due to an integer overflow error (this problem contains particularly large costs), so we compare our heuristic results against the best lower bound computed.

Our heuristic, Algorithm 3.5, appears to produce good quality solutions in a reasonable running time. Out of the 86 instances attempted, our heuristic consistently found the optimal solution to 60 of the instances. We focus on 18 select problems in Table 3.5. These are problems where a consistent solution was not found for each of the 10 trials. We present the best, average, and worst solution gaps found from the optimal solution value over 10 trials, i.e. if b is the (best/average/worst) bottleneck solution found by our algorithm and

b^* is the optimal solution value then

$$\text{gap } \% = (b - b^*)/b^* \times 100.$$

The average time reported includes output times, which were negligible (generally much less than 0.10 seconds).

We notice that performing shake operations ($p = 5$, $q = 5$) seem to produce solutions with a lower average and lower worst-gaps from the optimal solution. This indicates that shake operations are a promising idea for helping the Lin-Kernighan algorithm find good BTSP tours. Although the results are generally excellent, the heuristic performs poorly on the 'rtilt' and 'stilt' class of problems. These problems have many distinct, large, integer costs, and seem structurally quite difficult for the Lin-Kernighan algorithm.

We present in detail the results for all 86 problems in Tables 3.6, 3.7, and 3.8 for $p = 5$ and $q = 5$. We also present the lower bounds that found a tight optimal solution, as well as the best lower bound for each problem (ties are broken by shortest running time). The columns 'Avg. Bin. Steps" and 'Avg. # of LK Calls" give the average number of binary search steps and calls to Concorde's Lin-Kernighan algorithm, respectively. The results are generally quite good and computational times reasonable.

3.6 Nozzel Guide Vane Assembly in Gas Turbine Engines: an Application

Many TSP applications have a natural and useful interpretation as a BTSP model. In this section we adapt an interesting application in aircraft engine maintenance from using a TSP model to a BTSP model, as well as present some computation results.

The gas turbine engines that power military and commercial aircrafts require extensive maintenance to ensure efficiency and reliability. One such stage of maintenance, the nozzle guide vane assembly, is formulated by Plante, Lowe and Chandrasekaran [81] as a TSP with a product type cost matrix. Interested readers may consult [49] or [81] for additional details on this application.

One of the main sections of the engine is the turbine, which consists of a series of nozzle assemblies and rotor pairings. The nozzle assembly consists of a sequence of nozzle guide vanes (hereafter referred to simply as 'vanes') affixed along the inner circumference of the

Problem	Size	Opt. Sol.	p	q	Best Gap %	Avg. Gap %	Worst Gap %	Avg. Time (s)
coin100.2	100	207	10	0	0.00	0.97	2.42	15.58
			5	5	0.00	0.48	1.45	16.95
rtilt100.2	100	227248	10	0	20.70	28.57	36.99	51.01
			5	5	22.29	28.76	33.63	48.99
rtilt100.3	100	236920	10	0	16.30	30.93	44.80	52.03
			5	5	22.79	29.48	44.78	54.63
rtilt316.10	100	152510	10	0	71.68	104.74	127.82	384.90
			5	5	96.33	101.68	110.68	490.10
shop100.0	100	2232	10	0	0.00	0.16	1.16	8.06
			5	5	0.00	0.06	0.58	5.57
shop316.10	316	2311	10	0	0.00	0.66	2.29	47.75
			5	5	0.00	1.11	2.34	92.42
stilt100.0	100	382208	10	0	3.72	11.20	23.61	43.25
			5	5	7.43	11.20	20.96	45.81
stilt100.2	100	377720	10	0	6.71	9.78	14.50	48.29
			5	5	5.56	11.80	19.12	48.93
stilt100.3	100	401976	10	0	1.88	9.81	22.74	44.22
			5	5	2.24	7.04	19.50	42.77
stilt100.4	100	347440	10	0	19.19	29.28	38.72	49.34
			5	5	24.40	29.43	38.10	52.61
stilt316.10	316	226504*	10	0	77.64	92.67	110.02	333.25
			5	5	83.18	91.39	99.89	352.84
ftv120	121	39	10	0	10.26	10.26	10.26	25.65
			5	5	0.00	5.38	10.26	16.85
ftv130	131	39	10	0	0.00	77.69	141.03	34.21
			5	5	0.00	27.18	135.90	21.42
ftv140	141	41	10	0	109.76	124.88	129.27	65.83
			5	5	0.00	86.10	129.27	53.34
ftv150	151	37	10	0	2.70	59.73	151.35	48.11
			5	5	0.00	57.30	140.54	51.01
rbg323	323	12	10	0	16.67	26.67	50.00	86.75
			5	5	16.67	20.00	50.00	94.78
rbg358	358	14	10	0	0.00	5.71	7.14	71.85
			5	5	0.00	4.29	7.14	75.06
rbg403	403	20	10	0	5.00	7.00	15.00	94.72
			5	5	5.00	5.50	10.00	110.60

Table 3.5: Asymmetric BTSP initial experimental results on 18 select problems. *Note that the 'optimal solution" reported for problem stilt316.10 is the best known lower bound for the problem.

Problem	Size	Tight LBs	Best LB	Best LB Time	Best LB Sol.	Opt. Sol.	Best Gap %	Avg. Gap %	Worst Gap %	Avg. Bin. Steps	Avg. # LK Calls	Avg. Time (s)
coin100.0	100	c f	BBSSP($\hat{C}$)	0.00	253	253	0.00	0.00	0.00	0.00	2.50	0.97
coin100.1	100	c f	BBSSP($\hat{C}$)	0.00	219	219	0.00	0.00	0.00	0.00	1.60	0.42
coin100.2	100	c f	BBSSP($\hat{C}$)	0.00	203	207	0.00	0.48	1.45	10.80	46.20	16.95
coin100.3	100	c f	BBSSP($\hat{C}$)	0.02	232	232	0.00	0.00	0.00	0.00	2.40	0.85
coin100.4	100	c d e f	BBSSP($\hat{C}$)	0.00	214	214	0.00	0.00	0.00	0.00	1.00	0.11
coin316.10	316	c f	BBSSP($\hat{C}$)	0.05	227	227	0.00	0.00	0.00	0.00	3.60	6.51
crane100.0	100	a b d e f	2MB	0.00	173390	173390	0.00	0.00	0.00	0.00	1.00	0.12
crane100.1	100	a b d e f	2MB	0.00	152923	152923	17.80	17.80	17.80	14.00	60.00	25.34
crane100.2	100	a b d e f	2MB	0.00	214843	214843	0.00	0.00	0.00	0.00	1.00	0.11
crane100.3	100	e f	BSCSSP	0.00	145622	145622	0.00	0.00	0.00	0.00	1.90	0.65
crane100.4	100	e f	BSCSSP	0.02	171484	171484	0.00	0.00	0.00	0.00	1.00	0.20
crane316.10	316	a b d e f	2MB	0.00	119345	120333	0.00	0.00	0.00	16.00	62.00	128.23
disk100.0	100	a b d e f	2MB	0.00	508034	508034	0.00	0.00	0.00	0.00	1.00	0.12
disk100.1	100	a b d e f	2MB	0.00	473495	473495	0.00	0.00	0.00	0.00	1.00	0.11
disk100.2	100	a b d e f	2MB	0.00	382677	382677	1.08	1.08	1.08	13.00	32.00	10.78
disk100.3	100	a b d e f	2MB	0.00	453657	453657	0.00	0.00	0.00	0.00	1.00	0.13
disk100.4	100	a b d e f	2MB	0.00	415696	415696	0.00	0.00	0.00	0.00	1.00	0.20
disk316.10	316	a b e f	2MB	0.01	309801	309801	0.00	0.00	0.00	0.00	1.00	0.85
rtilt100.0	100	e f	BSCSSP	0.00	260342	260342	9.43	12.88	18.16	12.80	79.30	47.55
rtilt100.1	100	a b d e f	2MB	0.00	291040	291040	0.00	9.65	17.33	11.50	68.60	41.22
rtilt100.2	100	e f	BSCSSP	0.00	227248	227248	22.29	28.76	33.63	13.00	79.30	48.99
rtilt100.3	100	e f	BSCSSP	0.00	236920	236920	22.79	29.48	44.78	12.80	81.80	54.63
rtilt100.4	100	e f	BSCSSP	0.00	294367	294367	7.99	9.19	11.96	13.00	83.40	58.21
rtilt316.10	316	a b d e f	2MB	0.01	152510	152510	96.33	101.68	110.68	16.00	131.50	490.10
shop100.0	100	a b d e f	2MB	0.00	2232	2232	0.00	0.06	0.58	1.10	9.00	5.57
shop100.1	100	e f	BSCSSP	0.01	2608	2608	0.00	0.00	0.00	0.00	1.00	0.11
shop100.2	100	a b d e f	2MB	0.00	3620	3620	0.00	0.00	0.00	0.00	1.00	0.10
shop100.3	100	a b d e f	2MB	0.00	2526	2526	0.00	0.00	0.00	0.00	1.00	0.39
shop100.4	100	a b d e f	2MB	0.00	2792	2792	0.00	0.00	0.00	0.00	1.00	0.10
shop316.10	316	a b d e f	2MB	0.00	2311	2311	0.00	1.11	2.34	6.50	30.70	92.42

Table 3.6: Complete asymmetric BTSP results (Part 1/3). Lower Bound (LB) algorithms: a) 2MB, b) BAP, c) BBSSP($\hat{C}$), d) BBSSP($\bar{C}$), e) BSCSSP, f) EBBP. Best/average/worst results reported from 10 trials for each problem.

Problem	Size	Tight LBs	Best LB	Best LB Time	Best LB Sol.	Opt. Sol.	Best Gap %	Avg. Gap %	Worst Gap %	Avg. Bin. Steps	Avg. # LK Calls	Avg. Time (s)
stilt100.0	100	e f	BSCSSP	0.00	382208	382208	7.43	11.20	20.96	12.80	75.20	45.81
stilt100.1	100	a b d e f	2MB	0.00	491416	491416	0.00	0.00	0.00	0.00	4.10	2.57
stilt100.2	100	e f	BSCSSP	0.00	377720	377720	5.56	11.80	19.12	12.80	80.10	48.93
stilt100.3	100	a b d e f	2MB	0.00	401976	401976	2.24	7.04	19.50	12.90	69.80	42.77
stilt100.4	100	e f	BSCSSP	0.00	347440	347440	24.40	29.43	38.10	12.80	85.00	52.61
stilt316.10	316	b f	BAP	0.80	226504	?	83.18	91.39	99.89	16.40	102.10	352.84
super100.0	100	a b c d e f	2MB	0.00	10	10	0.00	0.00	0.00	0.00	1.00	0.09
super100.1	100	a b d e f	2MB	0.00	11	11	0.00	0.00	0.00	0.00	1.00	0.09
super100.2	100	a b c d e f	2MB	0.00	10	10	0.00	0.00	0.00	0.00	1.00	0.09
super100.3	100	a b c d e f	2MB	0.00	10	10	0.00	0.00	0.00	0.00	1.00	0.09
super100.4	100	a b c d e f	2MB	0.00	10	10	0.00	0.00	0.00	0.00	1.00	0.09
super316.10	316	a b c d e f	2MB	0.00	9	9	0.00	0.00	0.00	0.00	1.00	0.48
ftv180	181	b	BAP	0.03	35	37	0.00	0.00	0.00	8.00	27.50	25.08
uk66	66	c f	BBSSP($\hat{C}$)	0.00	170	170	0.00	0.00	0.00	0.00	1.00	0.05
balas84	84	e f	BSCSSP	0.00	18	18	0.00	0.00	0.00	0.00	1.00	0.07
balas108	108	e f	BSCSSP	0.00	13	13	0.00	0.00	0.00	0.00	1.00	0.09
balas120	120	e f	BSCSSP	0.00	22	22	0.00	0.00	0.00	0.00	1.00	0.11
balas160	160	e f	BSCSSP	0.00	13	13	0.00	0.00	0.00	0.00	1.00	0.17
balas200	200	e f	BSCSSP	0.02	13	13	0.00	0.00	0.00	0.00	1.00	0.39
ran500.0	500	a b d e f	2MB	0.02	24	24	0.00	0.00	0.00	0.00	1.00	1.23
ran500.1	500	a b d e f	2MB	0.02	22	22	0.00	0.00	0.00	0.00	1.10	3.73
ran500.2	500	a b d e f	2MB	0.02	23	23	0.00	0.00	0.00	0.00	1.00	2.73
ran500.3	500	a b d e f	2MB	0.02	23	23	0.00	0.00	0.00	0.00	1.00	2.43
ran500.4	500	a b d e f	2MB	0.01	28	28	0.00	0.00	0.00	0.00	1.00	1.89
ran1000.0	1000	a b d e f	2MB	0.03	18	18	0.00	0.00	0.00	0.00	1.00	4.61
ran1000.1	1000	a b d e f	2MB	0.03	17	17	0.00	0.00	0.00	0.00	1.40	7.52
ran1000.2	1000	a b d e f	2MB	0.03	17	17	0.00	0.00	0.00	0.00	1.30	9.46
ran1000.3	1000	a b d e f	2MB	0.03	19	19	0.00	0.00	0.00	0.00	1.30	8.25
ran1000.4	1000	a b d e f	2MB	0.03	19	19	0.00	0.00	0.00	0.00	1.10	7.61

Table 3.7: Complete asymmetric BTSP results (Part 2/3). Lower Bound (LB) algorithms: a) 2MB, b) BAP, c) BBSSP($\hat{C}$), d) BBSSP($\bar{C}$), e) BSCSSP, f) EBBP. Best/average/worst results reported from 10 trials for each problem.

Problem	Size	Tight LBs	Best LB	Best LB Time	Best LB Sol.	Opt. Sol.	Best Gap %	Avg. Gap %	Worst Gap %	Avg. Bin. Steps	Avg. # LK Calls	Avg. Time (s)
br17	17	c d e f	BBSSP($\hat{C}$)	0.00	8	8	0.00	0.00	0.00	0.00	1.00	0.01
ft53	53	e f	BSCSSP	0.02	977	977	0.00	0.00	0.00	0.00	1.00	0.04
ft70	70	e f	BSCSSP	0.00	1398	1398	0.00	0.00	0.00	0.00	1.00	0.05
ftv33	34	a b d e f	2MB	0.00	113	113	0.00	0.00	0.00	0.00	1.00	0.02
ftv35	36	a b d e f	2MB	0.00	113	113	0.00	0.00	0.00	0.00	1.00	0.02
ftv38	39	a b d e f	2MB	0.00	113	113	0.00	0.00	0.00	0.00	1.00	0.03
ftv44	45	a b d e f	2MB	0.00	113	113	0.00	0.00	0.00	0.00	1.00	0.03
ftv47	48	a b d e f	2MB	0.00	104	104	0.00	0.00	0.00	0.00	1.00	0.03
ftv55	56	c f	BBSSP($\hat{C}$)	0.02	64	64	0.00	0.63	3.13	2.40	11.20	2.35
ftv64	65	a b d e f	2MB	0.00	104	104	0.00	0.00	0.00	0.00	1.00	0.05
ftv70	71	a b d e f	2MB	0.00	104	104	0.00	0.00	0.00	0.00	1.00	0.06
ftv90	91	e f	BSCSSP	0.00	48	48	0.00	0.00	0.00	0.00	2.70	1.21
ftv100	101	a b d e f	2MB	0.00	53	53	0.00	0.00	0.00	0.00	1.20	0.66
ftv110	111	b	BAP	0.02	39	39	0.00	7.95	10.26	7.20	33.40	19.03
ftv120	121	b	BAP	0.02	39	39	0.00	5.38	10.26	4.80	26.00	16.85
ftv130	131	b f	BAP	0.03	39	39	0.00	27.18	135.90	3.20	24.40	21.42
ftv140	141	a b d e f	2MB	0.00	41	41	0.00	86.10	129.27	5.60	44.20	53.34
ftv150	151	b	BAP	0.02	35	37	0.00	57.30	140.54	8.10	48.30	51.01
ftv160	161	b	BAP	0.02	35	37	0.00	0.00	0.00	8.00	31.90	26.14
ftv170	171	b	BAP	0.03	35	37	0.00	0.00	0.00	8.00	29.40	24.24
kro124p	100	a b d e f	2MB	0.00	607	607	0.00	0.00	0.00	0.00	1.20	0.24
p43	43	e f	BSCSSP	0.00	5008	5008	0.00	0.00	0.00	0.00	1.00	0.02
rbg323	323	b	BAP	0.25	12	12	16.67	20.00	50.00	4.10	26.00	94.78
rbg358	358	b	BAP	0.13	14	14	0.00	4.29	7.14	4.00	20.10	75.06
rbg403	403	b	BAP	0.73	20	20	5.00	5.50	10.00	4.00	27.80	110.60
rbg443	443	b	BAP	0.94	20	20	15.00	15.00	15.00	4.00	32.00	138.43
ry48p	48	c f	BBSSP($\hat{C}$)	0.00	550	577	0.00	0.00	0.00	10.00	38.00	5.17

Table 3.8: Complete asymmetric BTSP results (Part 3/3). Lower Bound (LB) algorithms: a) 2MB, b) BAP, c) BBSSP($\hat{C}$), d) BBSSP($\bar{C}$), e) BSCSSP, f) EBBP. Best/average/worst results reported from 10 trials for each problem.

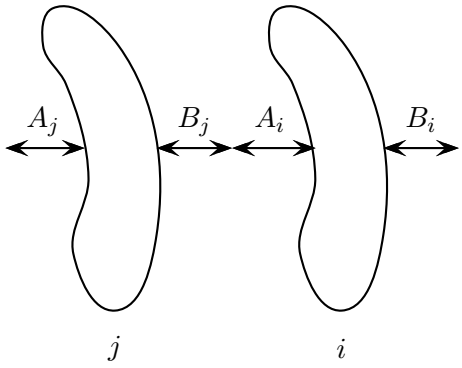


Figure 3.6: Area between vanes i and j

nozzle diaphragm. Typically, there are 46 to 100 vanes in a given nozzle assembly. The nozzle assembly accelerates, deflects, and distributes the gases that drive the engine. An engine is more efficient if the distribution of gas flow throughout the nozzle assembly is 'uniform'. The vanes experience wear and tear due to the high temperatures and velocities of the gases pushed through the nozzle assembly. During maintenance, a mechanic may choose to refurbish or discard some vanes, adding new or refurbished vanes to the collection as necessary. The mechanic then sequences the collection of vanes so as to obtain uniformity of flow areas between adjacent vanes.

Let n be the number of vanes in the nozzle assembly. Each vane has a convex and concave side which is measured against a master vane to derive two contribution values, A_i and B_i, for each vane i. Figure 3.6 illustrates this measurement. A positive value A_i or B_i indicates the vane contributes a larger area for gas flow than the master vane; likewise, a negative value indicates a small area contribution compared to the master vane. The measurement $(A_i + B_j)$ indicates how the area between vanes i and j differ from the area between two adjacent master vanes, with vane j placed clockwise-adjacent to vane i. In general, $(A_i + B_j) \neq (A_j + B_i)$.

For a set of vanes V, $n = |V|$, we can construct a complete graph $G = (V, E)$ where Hamiltonian cycles in G correspond to vane sequences in the nozzle assembly. For a Hamiltonian cycle H, let

$$x_{ij} = \begin{cases} 1 & \text{if } (i,j) \in H \\ 0 & \text{otherwise} \end{cases} \tag{3.15}$$

designate whether vane j is clockwise-adjacent to vane i in the nozzle assembly. One can

easily show the total nozzle flow area of any vane sequence will be

$$\sum_{i=1}^{n}\sum_{j=1}^{n}(A_i + B_j)x_{ij} = \sum_{i=1}^{n}(A_i + B_i) \tag{3.16}$$

as its value is constant since each vane is included exactly once. Suppose we calculate the mean nozzle flow area

$$\bar{d} = \frac{\sum_{i=1}^{n}(A_i + B_i)}{n}.$$

An assembly is perfectly uniform if $(A_i + B_j) = \bar{d}$ whenever $x_{ij} = 1$. However, it is unlikely such an assembly is possible due to imprecise manufacturing tolerances. Instead, one can determine vane placement by attempting to maximize 'uniformity" by some subjective criteria.

Let us cost matrices $C = (c_{ij})_{n\times n}$ and $D = (d_{ij})_{n\times n}$ where

$$c_{ij} = (\bar{d} - (A_i + B_j))^2, \tag{3.17}$$

and

$$d_{ij} = A_i B_j. \tag{3.18}$$

The entries of cost matrix C represent the square deviations of $(A_i + B_j)$ from $\bar{d}$, and the entries of D represent a product matrix. As noted previously, in general C and D are asymmetric matrices.

Plante et al.'s approach to maximize uniformity is to minimize the sum of square deviations of $(A_i + B_j)$ from $\bar{d}$ whenever $x_{ij} = 1$. The TSP heuristic they develop relies on the fact that an optimal TSP tour in C is also an optimal TSP tour in D. Our approach to maximizing uniformity is to minimize the maximum square deviation of $(A_i + B_j)$ from $\bar{d}$, that is to say modeling the problem as a BTSP instance with cost matrix C.

To test our asymmetric BTSP heuristic on nozzle guide vane problems we randomly generated five problems for each size $n = 50, 100, 250, 500$ with A_i and B_i values generated uniformly on the interval $[-1.0, 1.0]$. Because we prefer to work with a cost matrix C that is integer and nonnegative, we construct cost matrix $C^Z = (c^Z_{ij})_{n\times n}$ where

$$c^Z_{ij} = \lfloor 10^M c_{ij} + 0.5 \rfloor, \tag{3.19}$$

and M is a positive integer. Finding a BTSP tour with C^Z is equivalent to finding a BTSP tour with C if we rounded all c_{ij} to M decimal places. As measuring the values of A_i and B_i on an actual vane is limited to the accuracy of the measuring tool, rounding the values of c_{ij} in this manner is not unreasonable. We constructed our instances for $M = 4$ using a Python script to create problems in the TSPLIB format [95].

Table 3.9 reports the experimental results using Algorithm 3.5 for $p = 5$ and $q = 5$ on each problem over 10 trials. The table lists the 'seed" value Python's standard uniform random number generator function is initialized with [92]. The BAP bound is the strongest lower bound for each problem, and also tight to the optimal BTSP objective value for all problems. As before, if our algorithm cannot find a tour whose largest cost is equal to the best lower bound, we determine the optimal solution by running the exact version of our heuristic.

The results appear excellent for problems where $n = 50$ and $n = 100$, which from a practical point of view is excellent as typically there are 46 to 100 vanes in an actual nozzle assembly. For $n = 250$ and $n = 500$, the results are less impressive. We suspect generating instances with a larger value M, say $M = 6$ or $M = 8$, would produce integer cost matrices C^Z with more unique values, thus giving the Lin-Kernighan algorithm embedded in our heuristic more choices for finding an improving swap, thereby improving overall solution quality.

3.7 The Maximum Scatter TSP

The BTSP is a 'minimax" version of the traveling salesman problem (TSP), where the maximum cost in a Hamiltonian cycle (tour) is minimized. It is natural to ask the opposite: can we find a Hamiltonian cycle whose *minimum* cost is maximized? The 'maxmin" version of the TSP is called the *maximum scatter traveling salesman problem* (MSTSP). Given a (directed or undirected) graph $G = (V, E)$ with a nonnegative, integer cost c_{ij} assigned to each edge $(i, j) \in E$, the MSTSP is formally defined as

$$\begin{aligned} &\text{maximize } \min\{c_{ij} : (i,j) \in H\} \\ &\text{subject to } H \in \Pi(G) \end{aligned} \tag{3.20}$$

where $\Pi(G)$ is the collection of Hamiltonian cycles in G.

Problem	Size	Seed	Tight LBs	Best LB	Best LB Time	Best LB Sol.	Opt. Sol.	Best Gap %	Avg. Gap %	Worst Gap %	Avg. Bin. Steps	Avg. # LK Calls	Avg. Time (s)
vane50.1	50	501	b	BAP	0.00	359	359	0.00	0.00	0.00	0.00	1.00	0.05
vane50.2	50	502	b	BAP	0.00	617	617	0.00	0.00	0.00	0.00	1.00	0.06
vane50.3	50	503	b f	BAP	0.00	597	597	0.00	0.00	0.00	0.00	1.00	0.05
vane50.4	50	504	b	BAP	0.00	258	258	0.00	0.00	0.00	0.00	1.00	0.04
vane50.5	50	505	b	BAP	0.00	1056	1056	0.00	0.00	0.00	0.00	1.00	0.04
vane100.1	100	1001	b	BAP	0.03	528	528	0.00	2.23	6.63	6.30	35.20	24.76
vane100.2	100	1002	b	BAP	0.03	355	355	0.00	0.00	0.00	0.00	1.80	1.16
vane100.3	100	1003	b	BAP	0.02	292	292	0.00	0.00	0.00	0.00	1.20	0.57
vane100.4	100	1004	b	BAP	0.03	447	447	0.00	2.19	13.65	6.50	30.90	22.32
vane100.5	100	1005	b	BAP	0.02	209	209	0.00	0.00	0.00	0.00	1.00	0.37
vane250.1	250	2501	b f	BAP	0.50	377	377	0.00	0.00	0.00	0.00	1.00	2.91
vane250.2	250	2502	b	BAP	0.13	86	86	0.00	13.49	22.09	12.90	50.80	153.04
vane250.3	250	2503	b	BAP	0.19	134	134	41.79	50.00	58.21	14.30	71.80	223.24
vane250.4	250	2504	b	BAP	0.16	108	108	28.70	33.70	40.74	14.40	64.40	198.94
vane250.5	250	2505	b	BAP	0.16	106	106	35.85	46.60	51.89	14.10	68.30	207.06
vane500.1	500	5001	b	BAP	0.81	19	19	78.95	94.74	110.53	15.00	53.80	497.54
vane500.2	500	5002	b	BAP	0.97	49	49	20.41	31.22	46.94	15.00	48.50	387.04
vane500.3	500	5003	b	BAP	1.16	22	22	54.55	100.45	140.91	15.00	53.70	492.42
vane500.4	500	5004	b	BAP	0.89	65	65	184.62	210.46	229.23	15.00	69.70	614.13
vane500.5	500	5005	b	BAP	1.41	79	79	94.94	101.39	106.33	15.00	65.00	558.13

Table 3.9: Results on random nozzle guide vane problems. Lower Bound (LB) algorithms: a) 2MB, b) BAP, c) BBSSP($\hat{C}$), d) BBSSP($\bar{C}$), e) BSCSSP, f) EBBP. Best/Average/Worst results reported from 10 trials for each problem.

The MSTSP was first introduced by Arkin, Chiang, Mitchell, Skiena, and Yang in 1999 [6]. The MSTSP model finds use whenever one wishes to have significant separation between consecutive vertices in a tour. Arkin et al.'s motivation came from two applications: finding an optimal riveting sequencing for fastening aircraft body pieces together, and finding an optimal firing sequence for a dynamic spatial reconstructor. These two applications are discussed in Section 1.3.

It is well known from an optimization perspective that the MSTSP and the BTSP are equivalent. Given a (directed or undirected) graph $G = (V, E)$ with cost matrix C, Let

$$\bar{d} \geq \max\{c_{ij} : (i, j) \in E\} \tag{3.21}$$

and define an $n \times n$ cost matrix $D = (d_{ij})_{n \times n}$ where

$$d_{ij} = \bar{d} - c_{ij} \text{ for all } (i, j) \in E. \tag{3.22}$$

Solving the BTSP on D is equivalent to solving the MSTSP on C [54], therefore our BTSP heuristic can be used without modifications to solve the MSTSP. Further, any lower bounding scheme for the BTSP can be applied on cost matrix D to provide an upper bound on the optimal MSTSP objective value on C. We note that although this transformation preserves optimality, it does not preserve theoretical approximations, i.e. an ϵ-approximation for solving the MSTSP does not immediately provide an ϵ-approximation for the BTSP.

This transformation might prove to be difficult for our BTSP heuristic, so it warrants some computational study to explore its impact on the quality of our heuristic. We limit our computational study with the MSTSP to the same 86 asymmetric problem set used in Section 3.5.

Recall the lower bounding schemes presented for the asymmetric BTSP:

- the 2-max bound (2MB) [54, 91];
- the bottleneck assignment problem (BAP) [2, 87];
- the bottleneck biconnected spanning subgraph problem (BBSSP) [86, 73] on cost matrices $\bar{C}$, Equation (3.1), and $\hat{C}$, Equation (3.3);
- the bottleneck strongly connected spanning subgraph problem (BSCSSP) [84]; and
- the enhanced bidirectional bottleneck paths bound (EBBP).

Problem Set (# of Problems)	2MB	BAP	BBSSP($\check{C}$)	BBSSP($\hat{C}$)	BSCSSP	EBBP
coin (6)	0	5	1	0	0	2
crane (6)	1	6	0	1	1	2
disk (6)	1	6	1	1	1	2
rtilt (6)	5	6	0	5	5	6
shop (6)	5	6	0	5	5	5
stilt (6)	4	6	0	4	4	6
super (6)	6	6	5	6	6	6
balas (5)	5	5	0	5	5	5
tsplib: no ftv, ≤ 100 nodes (6)	1	6	1	1	1	2
tsplib: no ftv, > 100 nodes (4)	0	4	0	0	0	0
tsplib: ftv, ≤ 100 nodes (9)	1	9	0	1	1	1
tsplib: ftv, > 100 nodes (8)	0	8	0	0	0	0
ftv180 (1)	0	1	0	0	0	0
uk66 (1)	0	1	0	0	0	0
ran500 (5)	5	5	0	5	5	5
ran1000 (5)	5	5	0	5	5	5
Total (86)	39	85	8	39	39	47

Table 3.10: Asymmetric MSTSP upper bound summary on problem groups. The number listed under each bound name is the number of problems that bound gave the tightest upper bound in that problem group.

As before, we categorize each of the 86 problems into 16 groups for a compact presentation of the results in Table 3.10. The number given for each bound and problem group indicates the number of problems in that group for which the lower bound achieved the tightest result. These MSTSP upper bound results are very similar to the BTSP lower bound results we observed in Section 3.5 in that the BAP, BBSSP($\hat{C}$), and BSCSSP bounds generally provide cheap, tight upper bounds to the MSTSP objective value.

Tables 3.11, 3.12, and 3.13 report results of Algorithm 3.5 applied to cost matrix D, Equation (3.22), for $p = 5$ and $q = 5$ on each of the 86 problems over 10 trials. For 63 of the 86 problems, our algorithm had little trouble consistently finding an optimal tour. For the remaining 23 problems, our algorithm found a tour generally within 3% of optimality. As in previous cases, if our algorithm did not find a tour whose largest cost is equal to the best upper bound obtained, we confirm optimality using the exact version of Algorithm 3.5

where (heuristic) solver α is selected as Concorde's exact TSP solver.

3.8 The Constrained BTSP

As Figure 1.3 illustrates, a given tour can have a long total length while still being an optimal BTSP tour. For practical purposes, one may wish to minimize the total length as a secondary criteria. For a given graph $G = (V, E)$ with cost matrix C and optimal BTSP objective value z^*, one can find the shortest BTSP tour for G and C by solving the TSP on cost matrix $D = (d_{ij})_{n \times n}$ where

$$d_{ij} = \begin{cases} c_{ij} & \text{if } c_{ij} \leq z^*, \\ (z^* + 1)n & \text{otherwise.} \end{cases} \tag{3.23}$$

Alternatively, one may be satisfied with any BTSP tour whose length is less than a given value $\bar{l}$. Let us explore this idea in a more general way.

Given a graph G with cost matrix C, weight matrix W, and a given parameter $\bar{w}$, the *constrained bottleneck TSP* (constrained BTSP) is to

$$\begin{aligned} &\text{minimize } \max\{c_{ij} : (i,j) \in H\} \\ &\text{subject to } H \in \Pi(G), \\ &\qquad \sum_{(i,j) \in H} w_{ij} \leq \bar{w} \end{aligned} \tag{3.24}$$

where $\Pi(G)$ is the set of all Hamiltonian cycles in G. Let w^* be the TSP tour length on G with weight matrix W. It is obvious the constrained BTSP is only feasible for $\bar{w} \geq w^*$. One can let $W = C$ and $\bar{w} = \bar{l}$ to find the BTSP tour whose length is less than $\bar{l}$.

The constrained BTSP can be solved with minor modifications to the *IsFeasible* method, Algorithm 3.4. Start by modifying cost matrix C^δ, Equation (3.13), to be

$$c_{ij}^\delta = \begin{cases} w_{ij} & \text{if } c_{ij} \leq \delta, \\ \bar{w} + 1 & \text{otherwise.} \end{cases} \tag{3.25}$$

If we can find a tour of length less than or equal to $\bar{w}$, then we have a 'yes" answer to our feasibility test. The *BTSPThreshold* method, Algorithm 3.5, can be used as is.

Problem	Size	Tight LBs	Best LB	Best LB Time	Best LB Sol.	Opt. Sol.	Best Gap %	Avg. Gap %	Worst Gap %	Avg. Bin. Steps	Avg. # LK Calls	Avg. Time (s)
coin100.0	100	b	BAP	0.03	896	891	1.35	1.63	2.13	10.00	57.30	40.99
coin100.1	100	b	BAP	0.03	858	858	0.00	0.59	1.05	5.60	24.80	14.51
coin100.2	100	b f	BAP	0.02	974	974	1.85	2.46	2.77	10.00	56.30	34.58
coin100.3	100	b	BAP	0.03	890	890	2.92	3.31	3.93	9.90	52.20	34.12
coin100.4	100	c f	BBSSP($\hat{C}$)	0.00	903	903	0.00	0.12	0.78	6.00	19.40	10.90
coin316.10	316	b	BAP	0.59	1684	1684	2.61	3.05	3.56	10.80	63.60	293.33
crane100.0	100	b	BAP	0.03	647354	647354	0.00	0.00	0.00	0.00	1.10	0.30
crane100.1	100	a b d e f	2MB	0.00	627751	627751	0.00	0.00	0.00	0.00	1.00	0.11
crane100.2	100	b	BAP	0.00	645372	645372	0.00	0.00	0.00	0.00	1.00	0.22
crane100.3	100	b	BAP	0.02	629932	629932	0.07	0.32	0.48	12.90	55.50	36.57
crane100.4	100	b f	BAP	0.02	619054	619054	0.00	0.00	0.00	0.00	1.10	0.23
crane316.10	316	b	BAP	0.28	664713	664713	1.43	2.25	2.83	16.20	90.40	385.49
disk100.0	100	b	BAP	0.03	4767083	4767083	10.11	14.76	16.51	12.60	94.30	78.04
disk100.1	100	b	BAP	0.03	4882684	4882684	0.28	0.30	0.36	13.00	48.50	31.31
disk100.2	100	b c f	BBSSP($\hat{C}$)	0.00	4959789	4959789	0.00	0.00	0.00	0.00	1.80	1.24
disk100.3	100	b	BAP	0.03	4663663	4663663	0.95	2.03	2.52	12.90	74.70	57.06
disk100.4	100	a b d e f	2MB	0.00	4849971	4849971	1.61	4.32	17.20	12.70	72.70	57.94
disk316.10	316	b	BAP	0.91	4947068	4947068	7.66	11.57	19.41	16.20	107.00	416.43
rtilt100.0	100	a b d e f	2MB	0.00	529202	529202	0.00	0.00	0.00	0.00	1.00	0.24
rtilt100.1	100	a b d e f	2MB	0.00	546234	546234	0.00	0.00	0.00	0.00	1.00	0.13
rtilt100.2	100	b f	BAP	0.02	494714	494714	0.00	0.00	0.00	0.00	1.00	0.16
rtilt100.3	100	a b d e f	2MB	0.00	500897	500897	0.00	0.00	0.00	0.00	1.00	0.13
rtilt100.4	100	a b d e f	2MB	0.00	517383	517383	0.00	0.00	0.00	0.00	1.00	0.12
rtilt316.10	316	a b d e f	2MB	0.00	509367	509367	0.00	0.00	0.00	0.00	1.00	1.06
shop100.0	100	a b d e f	2MB	0.00	1939	1939	0.00	0.00	0.00	2.20	9.90	6.12
shop100.1	100	a b d e f	2MB	0.00	1786	1786	0.00	0.00	0.00	0.00	1.00	0.10
shop100.2	100	b	BAP	0.03	2466	2466	5.15	6.25	8.72	10.70	71.40	57.33
shop100.3	100	a b d e f	2MB	0.00	2044	2044	0.00	0.00	0.00	0.00	1.00	0.37
shop100.4	100	a b d e f	2MB	0.00	2104	2104	0.00	0.00	0.00	1.10	6.80	4.07
shop316.10	316	a b d e f	2MB	0.01	1966	1966	0.00	0.00	0.00	0.00	1.10	1.06

Table 3.11: Complete asymmetric MSTSP results (Part 1/3). Lower Bound (LB) algorithms: a) 2MB, b) BAP, c) BBSSP($\hat{C}$), d) BBSSP($\bar{C}$), e) BSCSSP, f) EBBP. Best/average/worst results reported from 10 trials for each problem.

Problem	Size	Tight LBs	Best LB	Best LB Time	Best LB Sol.	Opt. Sol.	Best Gap %	Avg. Gap %	Worst Gap %	Avg. Bin. Steps	Avg. # LK Calls	Avg. Time (s)
stilt100.0	100	a b d e f	2MB	0.00	1011282	1011282	0.00	0.00	0.00	0.00	1.00	0.11
stilt100.1	100	a b d e f	2MB	0.00	1071396	1071396	0.00	0.00	0.00	0.00	1.00	0.14
stilt100.2	100	a b d e f	2MB	0.00	957076	957076	0.00	0.00	0.00	0.00	1.00	0.11
stilt100.3	100	b f	BAP	0.03	997468	997468	0.00	0.00	0.00	0.00	1.00	0.12
stilt100.4	100	a b d e f	2MB	0.00	985154	985154	0.00	0.00	0.00	0.00	1.00	0.12
stilt316.10	316	b f	BAP	0.52	990472	990472	0.00	0.00	0.00	0.00	1.00	0.85
super100.0	100	a b c d e f	2MB	0.00	16	16	0.00	0.00	0.00	0.00	1.00	0.09
super100.1	100	a b c d e f	2MB	0.00	16	16	0.00	0.00	0.00	0.00	1.00	0.09
super100.2	100	a b d e f	2MB	0.00	16	16	0.00	0.00	0.00	0.00	1.00	0.09
super100.3	100	a b c d e f	2MB	0.00	16	16	0.00	0.00	0.00	0.00	1.00	0.09
super100.4	100	a b c d e f	2MB	0.00	16	16	0.00	0.00	0.00	0.00	1.00	0.10
super316.10	316	a b c d e f	2MB	0.00	17	17	0.00	0.00	0.00	0.00	1.00	0.49
ftv180	181	b	BAP	0.03	180	180	0.00	0.00	0.00	0.00	1.00	0.34
uk66	66	b	BAP	0.00	609	604	0.33	0.66	0.99	9.00	39.80	12.52
balas84	84	a b d e f	2MB	0.00	29	29	0.00	0.00	0.00	0.00	1.00	0.06
balas108	108	a b d e f	2MB	0.00	24	24	0.00	0.00	0.00	0.00	1.00	0.09
balas120	120	a b d e f	2MB	0.00	29	29	0.00	0.00	0.00	0.00	1.60	1.04
balas160	160	a b d e f	2MB	0.00	31	31	0.00	0.00	0.00	0.00	1.00	0.18
balas200	200	a b d e f	2MB	0.00	32	32	0.00	0.00	0.00	0.00	1.00	0.23
ran500.0	500	a b d e f	2MB	0.02	1000	1000	0.00	0.00	0.00	0.00	1.10	2.06
ran500.1	500	a b d e f	2MB	0.02	997	997	0.00	0.00	0.00	0.00	1.00	1.28
ran500.2	500	a b d e f	2MB	0.02	997	997	0.00	0.00	0.00	0.00	1.00	1.43
ran500.3	500	a b d e f	2MB	0.02	998	998	0.00	0.00	0.00	0.00	1.00	1.92
ran500.4	500	a b d e f	2MB	0.00	996	996	0.00	0.00	0.00	0.00	1.00	1.08
ran1000.0	1000	a b d e f	2MB	0.06	1000	1000	0.00	0.00	0.00	0.00	1.30	9.79
ran1000.1	1000	a b d e f	2MB	0.05	1005	1005	0.00	0.00	0.00	0.00	1.10	7.55
ran1000.2	1000	a b d e f	2MB	0.06	1004	1004	0.00	0.00	0.00	0.00	1.20	5.82
ran1000.3	1000	a b d e f	2MB	0.05	1004	1004	0.00	0.00	0.00	0.00	1.00	2.60
ran1000.4	1000	a b d e f	2MB	0.05	1006	1006	0.00	0.00	0.00	2.00	6.50	50.92

Table 3.12: Complete asymmetric MSTSP results (Part 2/3). Lower Bound (LB) algorithms: a) 2MB, b) BAP, c) BBSSP($\hat{C}$), d) BBSSP($\bar{C}$), e) BSCSSP, f) EBBP. Best/average/worst results reported from 10 trials for each problem.

Problem	Size	Tight LBs	Best LB	Best LB Time	Best LB Sol.	Opt. Sol.	Best Gap %	Avg. Gap %	Worst Gap %	Avg. Bin. Steps	Avg. # LK Calls	Avg. Time (s)
br17	17	b	BAP	0.00	5	5	0.00	0.00	0.00	0.00	1.00	0.02
ft53	53	b f	BAP	0.00	379	379	0.00	0.00	0.00	0.00	1.00	0.06
ft70	70	b	BAP	0.00	976	976	0.20	0.30	0.31	9.00	30.20	9.81
ftv33	34	b	BAP	0.00	143	143	0.00	0.00	0.00	0.00	1.00	0.02
ftv35	36	b	BAP	0.00	154	154	0.00	0.00	0.00	0.00	1.00	0.02
ftv38	39	b	BAP	0.02	154	154	0.00	0.00	0.00	0.00	1.00	0.03
ftv44	45	a b d e f	2MB	0.00	162	162	0.00	0.00	0.00	0.00	1.00	0.03
ftv47	48	b	BAP	0.00	168	168	0.00	0.00	0.00	0.00	1.00	0.03
ftv55	56	b	BAP	0.00	154	154	0.00	0.00	0.00	0.00	1.00	0.05
ftv64	65	b	BAP	0.00	160	160	0.00	0.00	0.00	0.00	1.00	0.05
ftv70	71	b	BAP	0.00	161	161	0.00	0.00	0.00	0.00	1.00	0.09
ftv90	91	b	BAP	0.02	148	148	2.70	3.04	4.05	7.00	38.30	22.67
ftv100	101	b	BAP	0.03	155	155	0.65	1.81	2.58	7.10	35.70	23.17
ftv110	111	b	BAP	0.02	165	165	0.00	1.03	1.82	6.90	27.40	21.22
ftv120	121	b	BAP	0.02	165	165	0.00	0.00	0.00	0.00	1.20	0.87
ftv130	131	b	BAP	0.02	172	172	0.00	0.00	0.00	0.00	1.00	0.17
ftv140	141	b	BAP	0.02	172	172	0.00	0.00	0.00	0.00	1.10	0.46
ftv150	151	b	BAP	0.03	178	178	0.00	0.00	0.00	0.00	1.00	0.23
ftv160	161	b	BAP	0.02	178	178	0.00	0.00	0.00	0.00	1.00	0.19
ftv170	171	b	BAP	0.02	180	180	0.00	0.00	0.00	0.00	1.00	0.94
kro124p	100	a b c d e f	2MB	0.00	2347	2347	0.00	0.00	0.00	0.00	1.00	0.15
p43	43	b	BAP	0.02	17	17	0.00	1.76	5.88	1.60	11.30	2.12
rbg323	323	b	BAP	0.17	23	23	8.70	8.70	8.70	4.00	24.20	78.19
rbg358	358	b	BAP	0.34	21	21	0.00	0.00	0.00	0.00	1.70	4.49
rbg403	403	b	BAP	0.41	19	19	0.00	0.53	5.26	2.00	11.90	43.47
rbg443	443	b	BAP	0.75	18	18	0.00	0.00	0.00	0.00	2.10	8.90
ry48p	48	b	BAP	0.00	1232	1232	2.27	3.20	4.30	9.80	49.90	11.43

Table 3.13: Complete asymmetric MSTSP results (Part 3/3). Lower Bound (LB) algorithms: a) 2MB, b) BAP, c) BBSSP($\hat{C}$), d) BBSSP($\bar{C}$), e) BSCSSP, f) EBBP. Best/average/worst results reported from 10 trials for each problem.

Similarly, we can modify cost matrix $C^{\delta,r}$, Equation (3.14), to create a 'shake" cost matrix. Recall that $z_1 < z_2 < \cdots < z_m$ are the distinct costs of C in ascending order. Let

$$c_{ij}^{\delta} = \begin{cases} w_{ij} + r_p & \text{if } c_{ij} \leq \delta \text{ where } c_{ij} = z_p, \\ \bar{w} + r_m + 1 & \text{otherwise.} \end{cases} \tag{3.26}$$

To answer 'yes" or 'no" to the feasibility question with cost matrix $C^{\delta,r}$ we have to compute the length of any tour using Equation (3.26) against the original weight matrix W, checking if the tour length is no more than $\bar{w}$ for a 'yes" answer.

Clearly, any BTSP lower bound on C is a valid lower bound for the constrained BTSP on C and W. Such lower bounds do not take into account the additional constraint, however, so we now explore a method to formulate a valid lower bound for the constrained BTSP that takes into account the additional constraint on total weight with weight matrix W.

Let γ be any TSP lower bound algorithm. Using γ and the constrained version of C^{δ}, (3.24), a constrained BTSP lower bound can be constructed informally as follows.

1. Let $z_1 < z_2 < \cdots < z_k$ be the distinct entries of C sorted in non-increasing order.
2. Let $l \leftarrow 1$, $u \leftarrow k$.
3. Let $k \leftarrow (u - l)/2 + l$.
4. Let $w \leftarrow \gamma(C^{z_k})$ (i.e. solve TSP lower bound γ on C^{δ} where $\delta = z_k$).
5. If $w \leq \bar{w}$, set $u = \delta$; otherwise, set $l = \delta + 1$.
6. If $l = u$ then return z_l; otherwise, go to step 3.

Let z be the result of this procedure. It is easy to se z is a valid lower bound for the constrained BTSP on C and W with parameter $\bar{w}$.

There are a variety of TSP lower bounds to choose as γ. For example, there is the minimum spanning tree bound [2], the Held-Karp bound [45, 46, 94], the assignment problem [2], and the degree-constrained minimum spanning tree bound [69]. If the running-time of a bound γ is $O(T_\gamma)$, the constrained bottleneck lower bound extension runs in $O(T_\gamma \log k)$-time. It is possible to significantly improve on this running-time by exploiting the problem's structure, such as Punnen and Nair's constrained bottleneck spanning tree bound [88].

Chapter 4

The Balanced TSP

A balanced optimization problem (BOP) finds a feasible solution that minimizes the range of dispersion among competing activities. It can be used to model problems where a uniform distribution of resources is important. The *balanced traveling salesman problem* (Balanced TSP) is an example of a BOP. Given a complete graph $G = (V, E)$, a nonnegative integer cost c_{ij} prescribed for each $(i, j) \in E$, and a collection $\Pi(G)$ of Hamiltonian cycles in G, the Balanced TSP is to

$$\begin{aligned} &\text{minimize } \max\{c_{ij} : (i,j) \in H\} - \min\{c_{ij} : (i,j) \in H\} \\ &\text{subject to } H \in \Pi(G). \end{aligned} \tag{4.1}$$

The nozzle guide vane assembly problem introduced in Chapter 1 is an example of an application where the Balanced TSP model is suitable for solving this problem. Recall that each vane i has values A_i and B_i that measure, respectively, the offset of the convex and concave sides of the vane from a master vane. The nozzle flow area between two vanes i and j placed clockwise-adjacent from one another is $(B_i + A_j)$. Refer to Figure 4.1 for a diagram of the vanes. The goal is to construct a vane ordering that is 'uniform'.

There are various ways to construct a 'uniform' assembly. The approach of Plante et al. [81] is to minimize the sum squared difference between the mean nozzle flow area $\bar{d}$ and the nozzle flow area $A_i + B_j$ between any pair of adjacent vanes. The mean total nozzle flow area $\bar{d}$ is given by

$$\bar{d} = \frac{\sum_{i=1}^{n}(A_i + B_i)}{n} \tag{4.2}$$

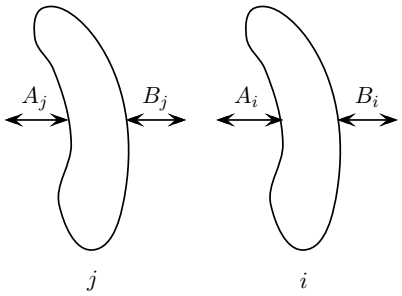


Figure 4.1: Area between vanes i and j

where n is the number of vanes. Note that $\bar{d}$ is constant, independent of sequencing. Construct a complete $n \times n$ cost matrix $C = (c_{ij})_{n \times n}$ where

$$c_{ij} = (\bar{d} - (A_i + B_j))^2.$$

Plante at al.'s approach is to solve the traveling salesman problem (TSP) on C.

In Chapter 3 we minimize the largest square difference between any pair of consecutive vanes by solving the bottleneck traveling salesman problem (BTSP) on C. Solving the Balanced TSP on C gives an assembly where the smallest and largest square difference between any pair of consecutive vanes is as small as possible. We believe the Balanced TSP model is very much suitable for this application.

Vairaktarakis' cycle workforce scheduling problem is another problem with a balanced objective [106]. BOP objective functions have also been studied by Katoh [58] and Martello et al. [74] in the context of single machine scheduling, and by Zeitlin [111] in the context of resource allocation. Further, many TSP and bottleneck TSP applications have an analogous meaning in the context of range minimization, thus the Balanced TSP is an interesting problem with practical implications.

Background on balanced optimization problems

The BOP class of problems was introduced by Martello, Pulleyblank, Toth, and de Werra in 1984 where they proposed a general purpose algorithm for solving BOPs of size n by invoking a feasibility-oracle $O(n)$ times [74]. Their motivation was an application in designing a tour for a travel agency. They also presented an improved algorithm for solving the balanced assignment problem.

Several authors have studied special cases, generalizations, and variations of BOPs since

Martello et al.'s paper. Camerini et al. [15] and later Galil and Schieber [35] considered BOP problems where the feasible solutions form a collection of all spanning-trees of a graph. Nemoto provided an efficient algorithm for finding an ideal of size k of a partially ordered set such that the range of the weights is at a minimum [76]. Improved algorithms for the special case where the set of feasible solutions consist of all cuts in a graph has been studied by Katoh and Iwano [59], Dai et al. [25], and Epstein [30].

Cappanera and Scutella studied an NP-hard BOP involving k-paths and presented polynomially solvable cases of the problem [16]. The balanced linear programming problem was introduced by Ahuja, where he proposed a parametric simplex algorithm for solving the problem [1]. Scutella [99] and Altenhöfer et al. [3] proposed strongly polynomial algorithms for special cases of network flow problems. Finally, Duin and Volgenant [28], Katoh [58], Punnen and Nair [89], Punnen and Aneja [85], and Tigan et al. [104] studied generalizations and variations on BOPs.

To the best of our knowledge no NP-hard BOP has been studied in literature, and no computational study has been performed with heuristics for any instance of a BOP.

Our contributions

We present heuristic algorithms for solving the Balanced TSP. The *Double-Threshold Algorithm* of Section 4.1 discusses an approach that has been used successfully in other balanced combinatorial optimization problems [89]. Section 4.2 introduces the *Double-Bottleneck* heuristic and the *Iterative-Bottleneck* algorithms, both which use the bottleneck TSP heuristics developed in Chapters 2 and 3. In Section 4.3 we show how to calculate a lower bound on the Balanced TSP objective value. Considerations for the Balanced TSP on an asymmetric cost matrix is presented in Section 4.4. Finally, computational results are given in Section 4.5.

Throughout this chapter we use the following notation. Let n be the number of vertices in the complete graph $G = (V, E)$, and let $z_1 < z_2 < \cdots < z_m$ be the distinct entries in a cost matrix C arranged in ascending order. We define $G[l, u]$ to be the subgraph containing only arcs with costs at least z_l and at most z_u, i.e. $G[l, u] = (V, E^{l,u})$ where

$$E^{l,u} = \{(i, j) \in E : z_l \leq c_{ij} \leq z_u\}. \tag{4.3}$$

Recall that $\Pi(G)$ denotes the set of Hamiltonian cycles in G. Finally, let BTSP(C) and

MSTSP(C) be the optimal objective values for the BTSP and MSTSP on cost matrix C, respectively. To develop algorithms for the Balanced TSP we initially assume C is a symmetric matrix. The case when C is asymmetric is treated separately in Section 4.4.

4.1 Double-Threshold Algorithm

Using the $G[l, u]$ notation we can restate the Balanced TSP problem for a graph G with cost matrix C as

$$\begin{aligned} &\text{minimize } z_u - z_l \\ &\text{subject to } G[l, u] \text{ is Hamiltonian,} \\ &1 \leq l \leq u \leq m. \end{aligned} \tag{4.4}$$

Assuming we could determine if $G[l, u]$ is Hamiltonian or not by a polynomial-time oracle, we could solve the problem through a systematic search of values of l and u. We can informally describe the Double-Threshold (DT) algorithm as follows:

1. Set $l \leftarrow 1$, $u \leftarrow 1$, $l^* \leftarrow 1$, $u^* \leftarrow m$.
2. If $G[l, u]$ is not Hamiltonian, set $u \leftarrow u + 1$. Go to step 4.
3. If $G[l, u]$ is Hamiltonian then:
   (a) If $z_u - z_l < z_{u^*} - z_{l^*}$, let $l^* \leftarrow l$, $u^* \leftarrow u$.
   (b) Set $l \leftarrow l + 1$.
4. If $l > u$ or $u > m$, return any tour in $G[l^*, u^*]$. Otherwise, go to step 2.

Note that the DT-algorithm is a specialization of Martello et al.'s algorithm [74]. Of course, detecting if a general graph is Hamiltonian or not is an NP-Hard problem. The above algorithm asks if a graph is Hamiltonian $O(m)$-times, so the approach is not practical when the problem size is very large. We instead look at how to convert this approach into an efficient heuristic. In previous chapters we formulate the Hamiltonian question as a TSP problem. Although solving the TSP is no easier, we can utilize powerful TSP heuristics to help answer if $G[l, u]$ is Hamiltonian or not (in an approximate way). Section 4.1.2 explores this idea for the Balanced TSP.

But first, in Section 4.1.1 we test the graph $G[l, u]$ to determine if it possess necessary conditions to support Hamiltonian cycles. Section 4.1.3 introduces some heuristic decisions

on values of l and u to help speed up the search considerably, and also presents the final algorithm.

4.1.1 Necessary Conditions for Hamiltonicity

Before attempting to find a Hamiltonian cycle in $G[l, u]$, we can examine the graph to see if passes some necessary conditions to support Hamiltonian cycles. In this section, we discuss how BTSP lower bounds and MSTSP upper bounds can be used to eliminate large portions of the solution space. We also present two simple checks one can perform in polynomial-time to conclude $G[l, u]$ is non-Hamiltonian, both derived from lower bounds for the Bottleneck TSP.

Information from BTSP and MSTSP solutions

Let H' and H'' be optimal BTSP and MSTSP tours in a graph G with cost matrix C. Find the integers L, U, and M such that $z_L = \min\{c_{ij} \in H'\}$, $z_U = \max\{c_{ij} \in H'\}$, and $z_M = \min\{c_{ij} \in H''\}$. From these values we know Hamiltonian cycles do not exist in $G[l, u]$ when $u < U$ and $l > M$. Further, as H' is a Hamiltonian cycle in $G[L, U]$, we can start our search in the DT-algorithm with $l = L + 1$, $u = U$, and stop whenever $l > M$.

The preceding chapters show how to heuristically find BTSP and MSTSP tours in a heuristic manner, but a provably optimal solution is often found. As such, we can use these heuristics for our Balanced TSP heuristics so long as the BTSP and MSTSP heuristics find provably optimal solutions. Alternatively, one can instead find a lower bound and upper bound on the BTSP and MSTSP objective values, respectively, and use those values to inform a starting value for u and a stopping criteria for l. We refer to Chapters 2 and 3 for more information on bounds for these problems.

Biconnectivity of Hamiltonian Cycles

A biconnected (2-connected) graph can be defined as a graph where either:

(a) two vertex-disjoint paths exists between any pair of vertices; or,

(b) the removal of any vertex will not disconnect the graph (i.e. the graph contains no *articulation points*).

As any Hamiltonian cycle in an undirected graph is biconnected, $G[l,u]$ must also be biconnected in order to support a Hamiltonian cycle. Tarjan presents a depth-first search algorithm for checking biconnectivity in $O(m)$-time [103].

Cycle-covers and Hamiltonian Cycles

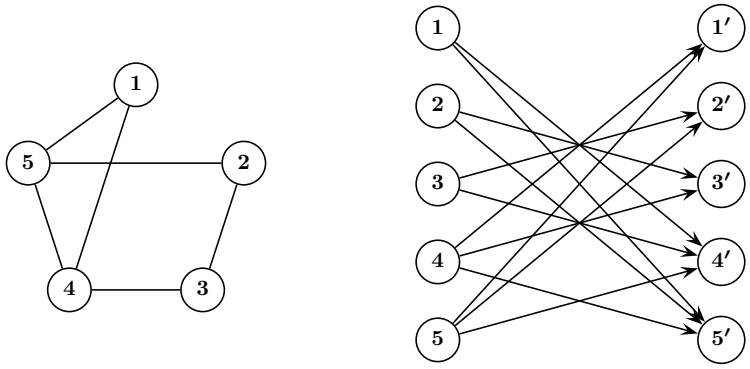
A k-cycle-cover is a spanning set of node-disjoint cycles where each cycle contains at least k vertices. A Hamiltonian cycle is therefore a n-cycle-cover. However, as finding Hamiltonian cycles is NP-Hard, we can instead attempt to determine if a k-cycle-cover exists for some $k < n$. If no k-cycle-cover exists for $k < n$ in $G[l,u]$ then it is clearly true that $G[l,u]$ is not Hamiltonian.

For undirected graphs Hartvigsen proved 4-cycle-covers can be found in $O(n^3)$-time [42]. This improves the result of Tutte, who reduced the 3-cycle-cover problem to the classic perfect matching problem in undirected graphs [105]. Papadimitriou and Yannakakis showed the k-cycle-cover problem is NP-Complete for $k \geq 6$ [78]. The complexity of the 5-cycle-cover problem is still open.

For directed graphs we will show how to solve the 2-cycle-cover problem in polynomial-time. This result obviously presents a polynomial-time 2-cycle-cover algorithm for undirected graphs as well. Valiant showed the k-cycle-cover problem is NP-Complete for $k \geq 3$ in directed graphs [107] (see also Garey and Johnson [36]).

Given a graph $G = (V, E)$ we can construct a directed graph $D = (N_1 \cup N_2, A)$ where $N_1 = \{1, 2, \ldots, n\}$, $N_2 = \{1', 2', \ldots, n'\}$ and $A = \{(i, j') \in N_1 \times N_2 \text{ if } (i, j) \in E\}$ (note: we assume E is a set of undirected edges, but the same construction works if E is a set of directed edges). Clearly, D is bipartite. Suppose we solve the *bipartite matching problem* (BMP) on D, which is to find a matching M in D of maximal cardinality. If G is Hamiltonian, then a matching M of cardinality $n = |V|$ must exist. Simply take any Hamiltonian cycle in G, orient its edges either clockwise or counter-clockwise, and let that set of edges be the matching M. Figure 4.2 gives an example of the construction of D, and shows how a Hamiltonian cycle in G corresponds to a matching of cardinality n in D. The BMP can be solved in $O(\sqrt{n}m)$-time [2].

Suppose then if we construct the directed graph $D[l,u]$ from $G[l,u]$ and solve the BMP on $D[l,u]$. For $G[l,u]$ to contain a Hamiltonian cycle, the maximal cardinality matching in $D[l,u]$ must be n. If a matching M of cardinality n exists, then M defines a 2-cycle cover $S = \{S_1, S_2, \ldots, S_k\}$ of $G[l,u]$. This check for the support of a 2-cycle-cover is the one we use in



(a) Undirected graph G (b) Bipartite, directed graph D

Figure 4.2: Example of constructing D from a given G. Note that G has a Hamiltonian cycle $H = \{(1,4),(4,3),(3,2),(2,5),(5,1)\}$, which corresponds to the matching $M = \{(1,4'),(4,3'),(3,2'),(2,5'),(5,1')\}$ in D.

our computational experiments for simplicity, but this check could obviously be strengthened by using Hartvigsen's 4-cycle-cover algorithm when the cost matrix is symmetric [42].

Further Comments

The idea for checking $G[l,u]$ for biconnectivity comes from the Bottleneck Biconnected Spanning Subgraph Problem (BBSSP) bound, and the idea of finding a cycle cover in $G[l,u]$ comes from the Bottleneck Assignment Problem (BAP) bound. Both these bounds are discussed in Chapter 2, where it is argued that the BAP and BBSSP are *non-dominated* lower bounds. This means instances exist where the BAP is the superior bound, just as instances exist where the BBSSP is the superior bound. On the other hand, the BBSSP bound dominates the 2-Max Bound (2MB), as no instance exists where the 2MB gives a superior bound to the BBSSP. A similar argument can be made for performing these two checks. Figure 4.3(a) is biconnected, but does not contain a cycle cover, while Figure 4.3(b) contains a cycle cover, but is not biconnected.

Other polynomial-time checks could be made to find structures in $G[l,u]$ that do not support Hamiltonian cycles, but one must be careful that the time spent checking for non-Hamiltonian structures does not outweigh the time spent searching for a Hamiltonian tour. Further, many of the various sufficient conditions of Hamiltonicity are satisfied either for dense graphs, or graphs that are specially structured [90]. The graph $G[l,u]$ is likely to be sparse when l and u are close, so we decided to not incorporate such sufficient conditions.

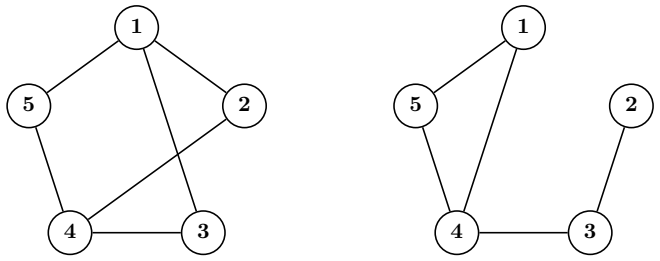


Figure 4.3: The graph depicted in (a) is biconnected, but does not contain a cycle cover, while the graph depicted in (b) contains a cycle cover, but is not biconnected.

4.1.2 Detecting Hamiltonian Cycles

As explored in previous chapters, we can use a TSP heuristic with a graph $G[l, u]$ to conclude with certainty that $G[l, u]$ is Hamiltonian, or heuristically conclude that it is non-Hamiltonian. There are many ways to formulate a cost matrix to help us answer this question as a TSP problem. Let us consider some desirable properties of such a cost matrix for the DT-algorithm.

The basic DT-algorithm only increases l by one whenever a Hamiltonian cycle is found in $G[l, u]$. If we can make costs closer to z_u more desirable for a TSP heuristic, then it is possible we can find a tour in $G[l, u]$ whose smallest cost is greater than z_l. This would let us increase l rapidly to achieve faster convergence without any additional effort.

Further, in previous chapters we have found good value in introducing randomness into the cost matrix (what we call 'shake" operations). As our TSP heuristic of choice, the Lin-Kernighan heuristic, is a local-search algorithm, this randomness permutes the local neighbourhoods without changing where global minima are located at. This process can help the Lin-Kernighan heuristic avoid getting consistently stuck in a local minimum. If the Lin-Kernighan heuristic fails to find a Hamiltonian cycle in $G[l, u]$, we can simply generate a new set of random numbers and ask the question again.

Finally, we obviously want to weight edges outside of the range $[z_l, z_u]$ heavily to exclude them from any solution. To this end, let $\delta_1 > \delta_2 > \cdots > \delta_m$ be a set of ordered random numbers on some interval $[a, b]$ and let r_{ij} be a random integer also on the interval $[a, b]$. Further, choose an integer $M \geq \delta_1 + z_u - z_l$. Let $\hat{C} = (\hat{c}_{ij})_{n \times n}$ be a $n \times n$ cost matrix for

the graph $G[l, u]$ where

$$\hat{c}_{ij} = \begin{cases} z_u - z_k + \delta_k & \text{if } z_l \leq c_{ij} \leq z_u \text{ and } c_{ij} = z_k \\ nM + r_{ij} & \text{otherwise.} \end{cases} \tag{4.5}$$

If a TSP heuristic with cost matrix $\hat{C}$ can find a tour with length less than or equal to nM, then we can provably conclude that $G[l, u]$ is Hamiltonian. However, a tour with length greater than nM allows us to only heuristically conclude $G[l, u]$ is non-Hamiltonian. We can see $\hat{C}$ satisfies the properties mentioned above: costs outside of $[z_l, z_u]$ are heavily weighted, costs closer to z_u are small, and random numbers allow us to shake this cost matrix to help a TSP heuristic find the global minimum.

Algorithm 4.1, *IsHamiltonian*, outlines our procedure for detecting if $G[l, u]$ is Hamiltonian or not, where α is a TSP heuristic, and μ is the number of attempts with cost matrix $\hat{C}$. This method also includes the necessary conditions for Hamiltonicity discussed in Section 4.1.1.

With Algorithm 4.1 defined, we formally describe the double threshold (DT) method in Algorithm 4.2. Algorithm 4.2 uses a Balanced TSP lower bound value LB which we show how to calculate in Section 4.3. Whenever l is increased we can also use the lower bound value LB to increase u to the point such that $z_u - z_l \geq LB$.

Theorem 14. *The DT-Algorithm computes an optimal solution to the Balanced TSP if α is an exact TSP solver. If α is a TSP heuristic, it terminates with a heuristic solution to the Balanced TSP.*

Proof. The proof of this correctness is not difficult when α is an exact TSP solver, so we omit it. If α is a TSP heuristic, it might falsely conclude $G[l, u]$ is non-Hamiltonian. Algorithm 4.2 therefore becomes a Balanced TSP heuristic when α is a TSP heuristic. □

Experimentally, the Lin-Kernighan TSP heuristic has a running time of approximately $O(n^{2.2})$ [47, 51]. When it used as the TSP heuristic α, the DT-Algorithm has a running time of approximately $O(n^{2.2}m)$.

4.1.3 Heuristic Decisions on Values of l and u

Incrementing either l or u by one at each iteration of the DT-algorithm is a slow crawl through the solution space of possible z_i values, particularly if m is large. As m could be

Algorithm 4.1: IsHamiltonian$(G, C, l, u, \alpha, \mu)$

Input: A complete graph G on n vertices with cost matrix C, indices l and u $(1 \leq l \leq u \leq m)$, TSP solver/heuristic α, and integer μ which represents the number of iterations with cost matrix $\hat{C}$.

Output: The 4-tuple $(feasible, tour, min\_cost, max\_cost)$ where $feasible$ is a Boolean value that indicates if a Hamiltonian cycle was found in $G[l, u]$, $tour$ is the feasible tour found, and min_cost and max_cost is smallest and largest cost in $tour$, respectively.

if *G is not biconnected* **then**
  return $(FALSE, \emptyset, -\infty, \infty)$
end
if *G does not contain a cycle cover* **then**
  return $(FALSE, \emptyset, -\infty, \infty)$
end
Let $z_1 < z_2 < ... < z_m$ be a list of the unique costs from C in ascending order;
for $i = 1, \ldots, \mu$ **do**
  Let $\delta_1 > \delta_2 > ... > \delta_m$ be a list of random integers arranged in ascending order where $\delta_1 \leq n^2$ and $\delta_m \geq 1$;
  $M \leftarrow \delta_1 + z_u - z_l$;
  Construct $\hat{C}$ given l, u, $z_1, \ldots, z_m$, $\delta_1, \ldots, \delta_m$, M, and a random number generator;
  $(length, tour) \leftarrow \alpha(n, \hat{C})$;
  $min\_cost \leftarrow \min\{c_{ij} : (i, j) \in tour\}$;
  $max\_cost \leftarrow \max\{c_{ij} : (i, j) \in tour\}$;
  if $length \leq nM$ **then**
    return $(TRUE, tour, min\_cost, max\_cost)$;
  end
end
return $(FALSE, \emptyset, -\infty, \infty)$;

Algorithm 4.2: DT$(G, C, H', H'', LB, \alpha, \mu)$

Input: A graph G with cost matrix C, BTSP tour H', MSTSP tour H'', Balanced TSP lower bound LB, TSP solver/heuristic α, and integer μ which represents the number of iterations with cost matrix $\hat{C}$.

Output: The (optimal/heuristic conclusion) on the Balanced TSP objective value and tour.

Let $z_1 < z_2 < ... < z_m$ be a list of the unique costs from C in non-increasing order;
Choose l such that $z_l = min\{c_{ij} : (i,j) \in H'\}$;
Choose u such that $z_u = max\{c_{ij} : (i,j) \in H'\}$;
Choose L such that $z_L = min\{c_{ij} : (i,j) \in H''\}$;

$OBJ \leftarrow z_u - z_l$; $best\_tour \leftarrow H'$;
if $OBJ = LB$ **then**
  return $(OBJ, best\_tour)$;
end

$l \leftarrow l + 1$;
while $(l \leq u)$, $(l \leq L)$, *and* $(u \leq m)$ **do**
  $(hamiltonian, tour) \leftarrow$ IsHamiltonian$(G, C, l, u, \alpha, \mu)$;
  if *hamiltonian* **then**
    Choose p such that $z_p = \min\{c_{ij} : (i,j) \in tour\}$;
    Choose q such that $z_q = \max\{c_{ij} : (i,j) \in tour\}$;
    if $z_q - z_p < OBJ$ **then**
      $OBJ \leftarrow z_q - z_p$; $best\_tour \leftarrow H^0$;
      if $OBJ = LB$ **then**
        return $(OBJ, best\_tour)$;
      end
    end
    $l \leftarrow p + 1$;
    if $z_u - z_l < LB$ **then**
      Choose smallest t such that $z_t - z_l \geq LB$;
      $u \leftarrow t$;
    end
  else
    $u \leftarrow u + 1$;
  end
end
return $(OBJ, best\_tour)$;

$O(n^2)$ when G is a complete graph, for very large n the DT-Algorithm might be impractical. Here we present two theorems which could reduce the number of iterations of the outer 'while' loop in the DT-algorithm. The first provides a new termination condition, while the second provides a means to rapdily increase the lower threshold l.

Let $S = \{H_1, H_2, \ldots, H_p\}$ be the Hamiltonian cycles found in order of discovery by a Balanced TSP algorithm where H_t indicates the Hamiltonian cycle found at iteration $t \in \{1, \ldots, p\}$. Further, let $r_t = \max\{c_{ij} : (i,j) \in H_t\}$ and $s_t = \min\{c_{ij} : (i,j) \in H_t\}$ be the smallest and largest cost in H_t, respectively. We will assume the Balanced TSP algorithm finds solutions in such an order that

$$r_1 \leq r_2 \leq \cdots \leq r_p \quad \text{and} \quad s_1 < s_2 < \cdots < s_p. \tag{4.6}$$

For $k = 1, \ldots, p$ choose an index $t(k)$ such that

$$r_{t(k)} - s_{t(k)} = \min\{r_i - s_i : i \in 1, \ldots, k\}. \tag{4.7}$$

The index $t(k)$ represents which of the known k solutions is the best in terms of Balanced TSP objective value. We can define the set of optimal Balanced TSP tours as the set

$$S^0 = \{H_i : r_i - s_i = r_{t(p)} - s_{t(p)}\}. \tag{4.8}$$

Finally, let γ be a parameter such that $\gamma \geq \max\{s_i : H_i \in S^0\}$.

Theorem 15. *For any $k \in 1, \ldots, p$, if $r_{t(k)} - s_{t(k)} + \gamma \leq r_k$, then $H_{t(k)} \in S^0$.*

Proof. Suppose $H_{t(k)} \notin S^0$. There must then exist an index $i > k$ such that $H_i \in S^0$ and $r_i - s_i < r_{t(k)} - s_{t(k)}$. We see $r_i < s_i + r_{t(k)} - s_{t(k)} \leq \gamma + r_{t(k)} - s_{t(k)} \leq r_k$ as $s_{t(k)} \leq \gamma$ and $r_k \geq r_{t(k)}$. Thus a contradiction presents itself, so $i \leq k$. □

Theorem 15 provides a new termination condition for the DT-algorithm. To be effective parameter γ should be as tight as possible to $\max\{s_i : H_i \in S^0\}$. We use the smallest cost in the MSTSP tour H'' for γ, although any upper bound on the MSTSP objective value could also be used.

Theorem 16. *If $H_{t(k)} \notin S^0$, then there exists $H_i \in S^0$, $q > k$ such that $s_q > r_k - r_{t(k)} + s_{t(k)}$.*

Proof. Suppose it is possible for $s_q \leq r_k - r_{t(k)} + s_{t(k)}$. Since $r_k \leq r_q$, it must be true that $s_q \leq r_q - r_{t(k)} + s_{t(k)}$. This implies $r_q - s_q \geq r_{t(k)} - st(k)$, but since $H_q \in S^0$, it must also be true that $H_{t(k)} \in S^0$. Thus, a contradiction. □

Practically, Theorem 16 can be used to rapidily increase the index of the lower threshold l.

Ahuja [1], Punnen [83], Punnen and Nair [89], and Martins [75] all used results similar to Theorems 15 and 16 for exact BOP algorithms. The theorems presented above are slightly more general than those considered in [1, 75, 83, 89]. Both theorems are only true if we can solve the BOP exactly.

As we use a TSP heuristic to determine if $G[l, u]$ is Hamiltonian or not, it is possible the conditions of Equation (4.6) may not hold. We choose to apply these theorems in a heuristic way and assume that the conditions of Equation (4.6) are true. If our TSP heuristic is powerful, these conditions should rarely be violated. If the conditions of Equation (4.6) are violated, solution quality may deteriorate but at the benefit of shorter running-times. The resulting trade-off between solution quality and running time must be analyzed experimentally. It should be noted that the conditions of Equation (4.6) will always hold if we used an exact TSP algorithm.

Incorporating both these modifications into the DT-Algorithm, we present the Modified Double-Threshold (MDT) Algorithm in Algorithm 4.3 for the Balanced TSP.

4.2 Double Bottleneck and Iterative Bottleneck Algorithms

The DT and MDT-algorithms gradually update two indices l and u using a TSP heuristic to determine if $G[l, u]$ is Hamiltonian or not. The double bottleneck (DB) and iterative bottleneck (IB) algorithms start from a similar place, but instead use a BTSP heuristic.

For any integer l, $1 \leq l \leq m$, and integer $M > z_m$, define the $n \times n$ cost matrix $C^l = (c^l_{ij})_{n \times n}$ where

$$c^l_{ij} = \begin{cases} c_{ij} & \text{if } c_{ij} \geq z_l \\ M & \text{otherwise.} \end{cases} \tag{4.9}$$

Let T be a BTSP tour on C^l, and let $z_u = \max\{c_{ij} : (i, j) \in T\}$. It is clear $c_{ij} \in [z_l, z_u]$ for all $(i, j) \in T$ so long as $G[l, m]$ is Hamiltonian. Further, this indicates the Balanced TSP objective value is at most $z_u - z_l$. We can therefore define the Balanced TSP problem using the BTSP as follows:

Algorithm 4.3: MDT$(G, C, H', H'', LB, \alpha, \mu)$

Input: A graph G with cost matrix C, BTSP tour H^0, MSTSP tour H^∞, TSP solver/heuristic α, Balanced TSP lower bound LB, and integer μ which represents the number of iterations with cost matrix $\hat{C}$.

Output: The (optimal/heuristic conclusion) on the Balanced TSP objective value and tour.

Let $z_1 < z_2 < ... < z_m$ be a list of the unique costs from C in non-increasing order;
Choose l such that $z_l = min\{c_{ij} : (i,j) \in H'\}$;
Choose u such that $z_u = max\{c_{ij} : (i,j) \in H'\}$;
Choose L such that $z_L = min\{c_{ij} : (i,j) \in H''\}$;

$OBJ \leftarrow z_u - z_l$; $best\_tour \leftarrow H'$;
if $OBJ = LB$ **then**
  return $(OBJ, best\_tour)$;
end

$l \leftarrow l + 1$;
while $(l \leq u)$, $(l \leq L)$, *and* $(u \leq m)$ **do**
  $(hamiltonian, tour) \leftarrow$ IsHamiltonian$(G, C, l, u, \alpha, \mu)$;
  if *hamiltonian* **then**
    Choose p such that $z_p = \min\{c_{ij} : (i,j) \in tour\}$;
    Choose q such that $z_q = \max\{c_{ij} : (i,j) \in tour\}$;
    if $z_q - z_p < OBJ$ **then**
      $OBJ \leftarrow z_q - z_p$; $best\_tour \leftarrow H^0$;
      if $OBJ = LB$ **then**
        return $(OBJ, best\_tour)$;
      end
    end
    if $OBJ + L \leq z_p$ **then** `/* Theorem 15 */`
      return $(OBJ, best\_tour)$;
    end
    $l \leftarrow p + 1$;
    Choose smallest $k \geq l$ such that $z_k \geq z_u - OBJ$; `/* Theorem 16 */`
    $l \leftarrow k$;
    if $z_u - z_l < LB$ **then**
      Choose smallest t such that $z_t - z_l \geq LB$;
      $u \leftarrow t$;
    end
  else
    $u \leftarrow u + 1$;
  end
end
return $(OBJ, best\_tour)$;

$$\begin{aligned} &\text{minimize } BTSP(C^l) - z_l \\ &\text{subject to } BTSP(C^l) < M, \\ &l = 1, \ldots, m. \end{aligned} \tag{4.10}$$

We see the Balanced TSP may be solved by evaluating $O(m)$ BTSP instances. As with the DT-Algorithm, we know the BTSP is infeasible if $z_l > z_L = \text{MSTSP}(C)$ as $G[l, m]$ is non-Hamiltonian for $l > L$. An informal description of how to solve the Balanced TSP using an exact BTSP solver is as follows.

1. Set $l \leftarrow 1$, $l^* \leftarrow 1$, $u^* \leftarrow m$.
2. Let z_u be the optimal BTSP objective value on C^l.
3. If $z_u - z_l < z_{u^*} - z_{l^*}$, let $l^* \leftarrow l$, $u^* \leftarrow u$
4. If $l > L$, return any tour in $G[l^*, u^*]$. Otherwise, set $l \leftarrow l + 1$ and go to step 2.

While this method is correct, solving $O(m)$ BTSP instances is likely computationally infeasible as the BTSP is NP-Hard, especially when m is large. Chapters 2 and 3 present a BTSP heuristic that could be used in place of an exact BTSP solver, much as how we replaced the exact TSP solver of the DT and MDT-algorithms with the Lin-Kernighan TSP heuristic. Even with a good heuristic, this procedure only increases the lower threshold l by one at each iteration. The double bottleneck (DB) and iterative bottleneck (IB) algorithms improve the basic framework outlined above to make solving the Balanced TSP with a BTSP heuristic computationally efficient.

4.2.1 Avoiding Non-Improving Searches with the BTSP Heuristic

In Chapter 2 we discuss the bottleneck biconnected spanning subgraph problem (BBSSP) lower bound for the symmetric BTSP. Similarly, in Chapter 3 we discuss the bottleneck assignment problem (BAP) bound. The optimal objective values for the BBSSP and the BAP are lower bounds on the optimal BTSP objective value on a cost matrix C.

Observation 1. *Let H^* be the best known Balanced TSP tour (not necessarily optimal) with objective value OPT. For any l, let z_B be a lower bound on the optimal BTSP objective*

value on C^l. If $OPT < z_B - z_l$, then the BTSP tour on C^l will not be a better Balanced TSP tour than H^*.

Proof. $OPT < z_B - z_l \leq BTSP(C^l) - z_l$. □

Let z_B be the BBSSP bounds on cost matrix C^l. Following this observation, we can compare $z_B - z_l$ to the objective value of our current best known tour to decide if solving the BTSP on C^l is worth the effort. Our implementation of the BBSSP algorithm is an $O(m \log m)$-time implementation and our BAP algorithm is an $O(n^{2.5})$ implementation. Better implementations exist for both these bounds. Readers should consult Chapters 2 and 3 for a more indepth discussion of these bounds.

4.2.2 The Double-Bottleneck (DB) Algorithm

The basic framework presented above solves the BTSP on cost matrix C^l, Equation (4.9), for some $l \in 1, \ldots, m$. Let H^l be the resulting BTSP tour in C^l. Choose indices p_1 and u such that $z_{p_1} = \min\{c_{ij} : (i,j) \in H^l\}$ and $z_u = \max\{c_{ij} : (i,j) \in H^l\}$. As we are solving these problems in the order $l = 1, \ldots, m$, if $z_u > z_m$, then the BTSP problem is infeasible and we can terminate the search. Otherwise, z_u may be used to apply Theorems 15 and 16. While at this stage our intention is only to update the upper threshold (and thus best known solution), if $z_{p_1} > z_l$ we can set $l = p_1 + 1$ (instead of incrementing it by a single value).

Once the upper threshold z_u is fixed we attempt to find the largest lower index p_2 with value z_{p_2} such that $\Pi(G[p_2, u]) \neq \emptyset$. To do this, we can solve the MSTSP on the cost matrix $C^u = (c^u_{ij})_{n \times n}$ where

$$c^u_{ij} = \begin{cases} c_{ij} & \text{if } c_{ij} \leq z_u \\ z_l - 1 & \text{otherwise.} \end{cases} \tag{4.11}$$

Let H^u be the resulting MSTSP tour in C^u. Choose indicies p_2 and q such that $z_{p_2} = \min\{c_{ij} : (i,j) \in H^u\}$ and $z_q = \max\{c_{ij} : (i,j) \in H^u\}$. According to the BTSP tour H^l a MSTSP tour exists in C^u with objective value at least z_l, but if we use a heuristic it may not be able to find this tour again. If it cannot (i.e. $z_{p_2} = z_l - 1$) we abandon the attempt to increase l further. Otherwise, if $p_2 > p_1$ we increase $l = p_2 + 1$. We may also update the best known solution to H^u if necessary. It may be true that $q < u$, but we do not lower the upper threshold u to q (this, of course, will not happen if we use an exact BTSP and exact MSTSP solver).

We discuss how to calculate a Balanced TSP lower bound in Section 4.3. Let the lower bound value be LB. If at any time we find a tour whose Balanced TSP objective value equals LB we may terminate the algorithm with a provably optimal solution. If after updating indices l and u we find $z_u - z_l < LB$ we may increase u until $z_u - z_l = LB$. The DB algorithm continues until a tour with objective value LB is discovered, or lower index l is increased beyond L where $z_L = MSTSP(C)$. It may be noted that the conditions of Theorems 15 and 16 are valid for the DB-algorithm, so may be used to expedite the search. We synthesize all these ideas formally into Algorithm 4.4. As the BTSP and MSTSP heuristics each have an excepted running-time of $O(n^{2.2} \log m)$, the DB-algorithm will have an expected running-time of $O(n^{2.2} m \log m)$.

4.2.3 The Iterative Bottleneck (IB) Algorithm

The DB-algorithm solves a BTSP and MSTSP problem in each iteration to update the lower and upper threshold values. While this has the potential to reduce the number of iterations when compared to the DT-algorithm, the BTSP or MSTSP heuristics developed in the previous chapters are $O(\log m)$ more expensive than the TSP heuristic used in the DT-algorithm. The iterative bottleneck (IB) algorithm offers a compromise between these two algorithms. Like the DB-algorithm, it uses a BTSP heuristic to update the the upper threshold u; like the DT-algorithm, it uses a TSP heuristic to update the lower threshold l. To do the later operation, it again uses cost matrix $\hat{C}$, Equation 4.10. A formal description of this procedure is given in Algorithm 4.5. As with the DB-algorithm, the running-time of the IB-algorithm is $O(n^{2.2} m \log m)$.

4.3 Lower Bounds for the Balanced TSP

Either the DT-algorithm, or the DB/IB-algorithm can be easily modified to calculate a lower bound on the optimal Balanced TSP objective value by simply removing the tour finding piece. As both are asymptotically similar in running-time, either is a good choice. We present such a modification of the DB/IB algorithm to illustrate this procedure in Algorithm 4.6.

Algorithm 4.4: DB-Algorithm$(G, C, H', H'', LB, \beta, \gamma)$

Input: A graph G with cost matrix C, BTSP tour H^0, MSTSP tour H^1, Balanced TSP lower bound LB, BTSP heuristic β, and MSTSP heuristic γ.
Output: A Balanced TSP objective value and tour.

Let $z_1 < z_2 < ... < z_m$ be a list of the unique costs from C in non-increasing order;
Choose l such that $z_l = min\{c_{ij} : (i,j) \in H'\}$;
Choose u such that $z_u = max\{c_{ij} : (i,j) \in H'\}$;
Choose L such that $z_L = min\{c_{ij} : (i,j) \in H''\}$;

$OBJ \leftarrow z_u - z_l$; $best\_tour \leftarrow H^0$;
if $OBJ = LB$ **then return** $(OBJ, best\_tour)$;

$l \leftarrow l + 1$;
while $(l \leq L)$ **do**
  Let z_B be the larger of the BBSSP and BAP bounds on C^l;
  if $z_B - z_l < OPT$ **then**
    $(btsp\_feasible, btsp\_tour) \leftarrow \beta(G, C^l)$; `/* BTSP on` C^l `*/`
    if $btsp\_feasible$ **then**
      Choose p_1 such that $z_{p_1} = min\{c_{ij} : (i,j) \in btsp\_tour\}$;
      Choose q_1 such that $z_{q_1} = max\{c_{ij} : (i,j) \in btsp\_tour\}$;
      if $z_{q_1} - z_{p_1} < OBJ$ **then**
        $OBJ \leftarrow z_{q_1} - z_{p_1}$; $best\_tour \leftarrow btsp\_tour$;
        if $OBJ = LB$ **then return** $(OBJ, best\_tour)$;
      end
      if $OBJ + L \leq z_{p_1}$ **then return** $(OBJ, best\_tour)$; `/* Theorem 15 */`
      $(mstsp\_feasible, mstsp\_tour) \leftarrow \gamma(G, C^u)$; `/* MSTSP on` C^u `*/`
      Choose p_2 such that $z_{p_2} = min\{c_{ij} : (i,j) \in mstsp\_tour\}$;
      Choose q_2 such that $z_{q_2} = max\{c_{ij} : (i,j) \in mstsp\_tour\}$;
      if $z_{q_2} - z_{p_2} < OBJ$ **then**
        $OBJ \leftarrow z_{q_2} - z_{p_2}$; $best\_tour \leftarrow mstsp\_tour$;
        if $OBJ = LB$ **then return** $(OBJ, best\_tour)$;
        if $OBJ + L \leq z_{p_2}$ **then return** $(OBJ, best\_tour)$; `/* Theorem 15 */`
      end
      $l \leftarrow 1 + \max\{p_1, p_2\}$;
      Choose smallest $k \geq l$ such that $z_k \geq z_{q_1} - OBJ$; Let $l \leftarrow k$; `/* Thm 16 */`
      if $z_{q_1} - z_l < LB$ **then**
        Choose smallest t such that $z_t - z_l \geq LB$ and let $u \leftarrow t$;
      end
    end
  else
    $l \leftarrow l + 1$;
  end
end
return $(OBJ, best\_tour)$;

Algorithm 4.5: IB-Algorithm$(G, C, H', H'', LB, \beta, \alpha)$

Input: A graph G with cost matrix C, BTSP tour H^0, MSTSP tour H^1, Balanced TSP lower bound LB, BTSP heuristic β, and TSP heuristic α.
Output: A Balanced TSP objective value and tour.

Let $z_1 < z_2 < ... < z_m$ be a list of the unique costs from C in non-increasing order;
Choose l such that $z_l = min\{c_{ij} : (i,j) \in H'\}$;
Choose u such that $z_u = max\{c_{ij} : (i,j) \in H'\}$;
Choose L such that $z_L = min\{c_{ij} : (i,j) \in H''\}$;

$OBJ \leftarrow z_u - z_l$; $best\_tour \leftarrow H^0$;
if $OBJ = LB$ **then return** $(OBJ, best\_tour)$;

$l \leftarrow l + 1$;
while $(l \leq L)$ **do**
  Let z_B be the larger of the BBSSP and BAP bounds on C^l;
  if $z_B - z_l < OPT$ **then**
    $(btsp\_feasible, btsp\_tour) \leftarrow \beta(G, C^l)$; /* BTSP on C^l */
    if $btsp\_feasible$ **then**
      Choose p_1 such that $z_{p_1} = min\{c_{ij} : (i,j) \in btsp\_tour\}$;
      Choose q_1 such that $z_{q_1} = max\{c_{ij} : (i,j) \in btsp\_tour\}$;
      if $z_{q_1} - z_{p_1} < OBJ$ **then**
        $OBJ \leftarrow z_{q_1} - z_{p_1}$; $best\_tour \leftarrow btsp\_tour$;
        if $OBJ = LB$ **then return** $(OBJ, best\_tour)$;
      end
      if $OBJ + L \leq z_{p_1}$ **then return** $(OBJ, best\_tour)$;
      ; /* Theorem 15 */
      $(tsp\_feasible, tsp\_tour) \leftarrow \alpha(G, \hat{C})$; /* TSP on $\hat{C}$ */
      Choose p_2 such that $z_{p_2} = min\{c_{ij} : (i,j) \in tsp\_tour\}$;
      Choose q_2 such that $z_{q_2} = max\{c_{ij} : (i,j) \in tsp\_tour\}$;
      if $z_{q_2} - z_{p_2} < OBJ$ **then**
        $OBJ \leftarrow z_{q_2} - z_{p_2}$; $best\_tour \leftarrow tsp\_tour$;
        if $OBJ = LB$ **then return** $(OBJ, best\_tour)$;
        if $OBJ + L \leq z_{p_2}$ **then return** $(OBJ, best\_tour)$; /* Theorem 15 */
      end
      $l \leftarrow 1 + \max\{p_1, p_2\}$;
      Choose smallest $k \geq l$ such that $z_k \geq z_{q_1} - OBJ$; Let $l \leftarrow k$; /* Thm 16 */
      if $z_{q_1} - z_l < LB$ **then**
        Choose smallest t such that $z_t - z_l \geq LB$ and let $u \leftarrow t$;
      end
    end
  else
    $l \leftarrow l + 1$;
  end
end
return $(OBJ, best\_tour)$;

Algorithm 4.6: IB-LowerBound(G, C, z_M)

Input: A problem on n nodes with cost matrix C, a MSTSP upper bound z_M.
Output: A lower bound on the Balanced TSP objective value.

Let $z_1 < z_2 < ... < z_m$ be a list of the unique costs from C in non-increasing order;
$l^* \leftarrow 1$; $u^* \leftarrow m$;
for $l = 1, \ldots, M$ **do**
  Let z_B be the larger of the BBSSP and BAP bounds on C^l;
  if $z_B - z_l < z_{u^*} - z_{l^*}$ **then**
    $l^* \leftarrow l$; $u^* \leftarrow B$;
  end
end
return $z_{u^*} - z_{l^*}$;

4.4 The Asymmetric Balanced TSP

We now outline the changes required for the DT and DB/IB-Algorithms if C is an asymmetric cost matrix, or, equivalently, G is a directed graph.

DT-Algorithm

Ensuring a directed graph $G[l, u]$ is biconnected is too strong of a condition as Hamiltonian cycles in directed graphs are not biconnected. Instead, one may construct a symmetric relaxation $\bar{G}[l, u] = (V, \bar{E}^{l,u})$ where $(i, j) \in \bar{E}$ if $(i, j) \in E^{l,u}$ or $(j, i) \in E^{l,u}$. The edge set $\bar{E}^{l,u}$ is clearly a set of undirected edges.

Theorem 17. *If $\bar{G}[l, u]$ is not biconnected, then $G[l, u]$ does not contain a Hamiltonian cycle.*

Proof. If $\bar{G}[l, u]$ is not biconnected, then it will not be Hamiltonian. As any directed Hamiltonian cycle in $G[l, u]$ corresponds to an undirected Hamiltonian cycle in $\bar{G}[l, u]$, if $\bar{G}[l, u]$ is non-Hamiltonian, then $G[l, u]$ must also be non-Hamiltonian. □

An additional check one can make is to determine if $G[l, u]$ is strongly connected. As any Hamiltonian cycle in $G[l, u]$ is strongly connected, $G[l, u]$ will be non-Hamiltonian if it is not strongly connected. Strong connectivity can be determined in $O(m)$ time [84].

DB and IB-Algorithms

As in the DT-Algorithm, solving the BBSSP directly on an asymmetric cost matrix C is not a valid lower bound for the BTSP on C. Instead, the BBSSP can be solved on a symmetric relaxation of the cost matrix C. We refer the reader to Chapter 3 for more details.

Further, one may additionally solve the Bottleneck Strongly Connected Spanning Subgraph Problem (BSCSSP) on C^l to obtain a tight BTSP lower bound. Solving the BSCSSP on C^l requires $O(m \log m)$ operations, as described in Chapter 3 and [84].

Tour Finding with a Symmetric TSP Heuristic

The $2n$ and $3n$-vertex symmetric transformations presented in Chapter 3 are not valid as throughout this chapter we restrict costs $c_{ij} < z_l$ for some l. Such a formulation can either exclude the fixed edges of Equation (3.1), or disconnect the nodes labelled $n+1$ to $2n$ from Equation (3.2).

To compensate, in any tour finding pieces one must make explicit exceptions for the fixed edges of Equation (3.1), or the zero-cost edges incident on nodes $n+1$ to $2n$ from Equation (3.2). For example, cost matrix $\hat{C}$, Equation (4.10), must be constructed as

$$\hat{c}_{ij} = \begin{cases} -\infty & \text{if } (i,j) \text{ is a fixed edge} \\ z_u - z_k + \delta_k & \text{if } z_l \leq c_{ij} \leq z_u \text{ and } c_{ij} = z_k \\ nM + r_{ij} & \text{otherwise.} \end{cases} \tag{4.12}$$

following the $2n$-vertex symmetric matrix of Equation (3.1) on the asymmetric cost matrix C. Similar modifications must be done to C^l and C^u, but we omit these details.

4.5 Computational Results

Recognizing our Balanced TSP heuristics could prove to be very expensive, even on small problems, our initial test set consisted only of problems of sizes from 42 vertices to 225 vertices. We picked problems `dantzig42`, `kroA100`, `pr124`, `ch150`, `si175`, and `ts225` from Reinelt's TSPLIB [95]. Further, using Johnson and McGeoch's random instance generators [50], we created two problems of 100 vertices and 150 vertices of uniformly distributed points (`E100.0`, `E150.0`), clustered points (`C100.0`, `C150.0`), and random-distance matrices

(M100.0, M150.0). We performed 3 trials with all four algorithms (DT, MDT, DB, and IB) on these 12 problems to compare the solution quality versus the running-time.

For every experiment in this section, the DT, MDT, DB, and IB algorithms are called with the following parameters: $\mu = 1$; α is Concorde's implementation of the Lin-Kernighan algorithm [4]; β is the BTSP binary search threshold heuristic developed in Chapter 2 with $(p, q, r) = (1, 0, 0)$; γ was the MSTSP heuristic developed in Chapter 3 by solving the BTSP with β on cost matrix D from Equation (3.22). All computational experiments in this section were carried out on PC with a 3.40GHz Pentium 4 CPU and 2GB of RAM running Microsoft Windows XP SP2 operating system and Cygwin NT 5.1. Reported running times are in CPU seconds rounded to two decimal places and include input and output times.

Table 4.1 presents the average gap percentage (%) over 3 trials for each of the 12 problems on the four algorithms ('Gap %" is the objective value's difference from the lower bound value as a percentage of the lower bound). Table 4.2 is similar, but presents the average running-time of the algorithms. The running-times do include input and output times, but do not include the time spent calculating a BTSP and MSTSP tour, nor the bottleneck TSP lower bound. We can see some deterioration in the MDT algorithm's solution quality as compared to the DT-algorithm, albeit with improved running-times. The DB-algorithm and IB-algorithm produced more accurate results, but generally took much longer to run compared to the MDT algorithm outside of two exceptions (pr124 and C150.0).

The better solution quality of the DB and IB-algorithms is likely due to the fact the initial cost matrix formulation C^l used in both algorithms, Equation (4.9), is less constrained (i.e. has less edge costs of significantly large weight so as to effectively exclude them from consideration) than the cost matrix $\hat{C}$ used in the DT and MDT-algorithms. The Lin-Kernighan algorithm attempts to replace k edges in a known Hamiltonian cycle with k new edges such that a new Hamiltonian cycle is produced with a better TSP objective value (known as a 'k-swap"). It does not fix k to a certain size, rather in its neighbourhood search it attempts to find long chains of improving swaps. The Lin-Kernighan algorithm is willing to make a non-improving swap in the hopes it may lead to a new series of improving swaps, but it is unlikely to make such a move if the non-improving swap increases the TSP objective value by a significantly large amount. This may explain why the DB and IB algorithms are producing higher quality solutions.

Suppose we generate problems with random distances where the numbers generated are on the range $(0, \kappa]$. Clearly, such a problem will have at most κ unique costs. We

			Avg. Gap from LB (%)			
Problem	Size	LB	DT	MDT	DB	IB
dantzig42	42	13	0.00	0.00	0.00	0.00
C100.0	100	17,391	0.00	0.00	0.00	0.00
E100.0	100	36,364	0.00	0.00	0.00	0.00
kroA100	100	137	2.68	4.38	0.00	0.00
M100.0	100	43,166	5.81	5.81	5.81	5.81
pr124	124	364	27.56	27.66	13.92	13.00
C150.0	150	9,212	0.50	0.50	0.50	0.50
ch150	150	17	0.00	0.00	0.00	0.00
E150.0	150	27,006	0.50	0.76	0.00	0.00
M150.0	150	31,161	2.78	2.45	0.00	0.00
si175	175	7	0.00	0.00	0.00	0.00
ts225*	225	0	-	-	-	-

Table 4.1: Balanced TSP objective value results on initial 12 problems. Results are averages from 3 trials on each problem for each heuristic. *Note that as the lower bound value LB for `ts225` is 0, a gap from the lower bound cannot be calculated due to division by zero. Each heuristic achieved a balanced objective value of 21 on `ts225`.

Problem	Size	DT	MDT	DB	IB
dantzig42	42	0.26	0.24	10.47	2.55
C100.0	100	7.60	7.42	721.60	23.09
E100.0	100	33.71	22.39	458.27	68.99
kroA100	100	251.45	243.59	292.62	93.19
M100.0	100	25.97	7.02	113.27	27.10
pr124	124	5,029.66	4,828.83	1,850.17	500.44
C150.0	150	10,410.84	9,877.43	2,689.05	114.44
ch150	150	35.50	18.62	325.67	36.85
E150.0	150	209.90	104.16	1,269.75	173.92
M150.0	150	74.97	21.33	389.70	23.08
si175	175	113.07	84.44	462.43	132.49
ts225*	225	122.52	50.87	189.48	106.11

Table 4.2: Balanced TSP running-time results on initial 12 problems. Results are averages from 3 trials on each problem for each heuristic.

call a problem 'dense" if a large proportion of its edges share the same cost, therefore a problem with numbers on the range $(0, \kappa]$ can be thought of as dense when κ is small. All the Balanced TSP algorithms we have discussed so far iterate through the list of unique costs in a problem, so very dense problems should have short running-times. On the other hand, extremely 'sparse" problems (i.e. problems where a high proportion of its edge costs are distinct) should allow improvements like Theorems 15 and 16 to be heavily utilized to decrease the running-time.

To test this theory, we created a number of random distance problems with 200 vertices with unique costs on the range $(0, \kappa]$ where $\kappa = 300, 600, \ldots, 3000$ and solved the Balanced TSP on each using the MDT and IB-algorithms. Figure 4.4 plots the average running-time from three trials for each problem from each algorithm versus the number of maximum unique costs in the problem. We can see some evidence that the running-time peaks when $\kappa = 2100$. Of course, this is not to say the density of similar valued edge costs is the only factor explaining running-time, but this is a step towards recognizing the sorts of problems our Balanced TSP heuristics may have some difficulty solving efficiently.

With these initial experiments completed, we decided to test the MDT and IB algorithms on the remaining 59 problems of 500 vertices or less from TSPLIB. We used the BTSP tours found through experiments in Chapter 2 as a starting point in our search, but we did not have any MSTSP tours calculated for symmetric problems. Rather than generate such tours, we instead calculated an upper bound on the MSTSP objective value, as described in Chapter 3, and used the MSTSP upper bound value for the purposes of our termination criteria.

Tables 4.3 and 4.4 present the results from a single attempt on each problem. In addition to the gap % and running-time, the column "Iters" gives the number of iterations of the outer 'While" loop for each problem and each algorithm. It is expected the IB-algorithm will generally have much fewer iterations. Columns "Obj Value" and "Optimal?" give the best found objective value for each problem, as well as if the solution is provably optimal. Columns "Best Obj" and "Best Time" indicate which algorithm returned the tour with smaller Balanced TSP objective value for that problem and which algorithm terminated more quickly, respectively. These tables also report the Balanced TSP lower bound ('LB") and the running-time ('LB Time") necessary to calculate the bound. Note that the reported running-time could be improved by using Theorems 15 and 16 to speed up the search.

For 29 of the problems the MDT algorithm had a shorter running-time compared to

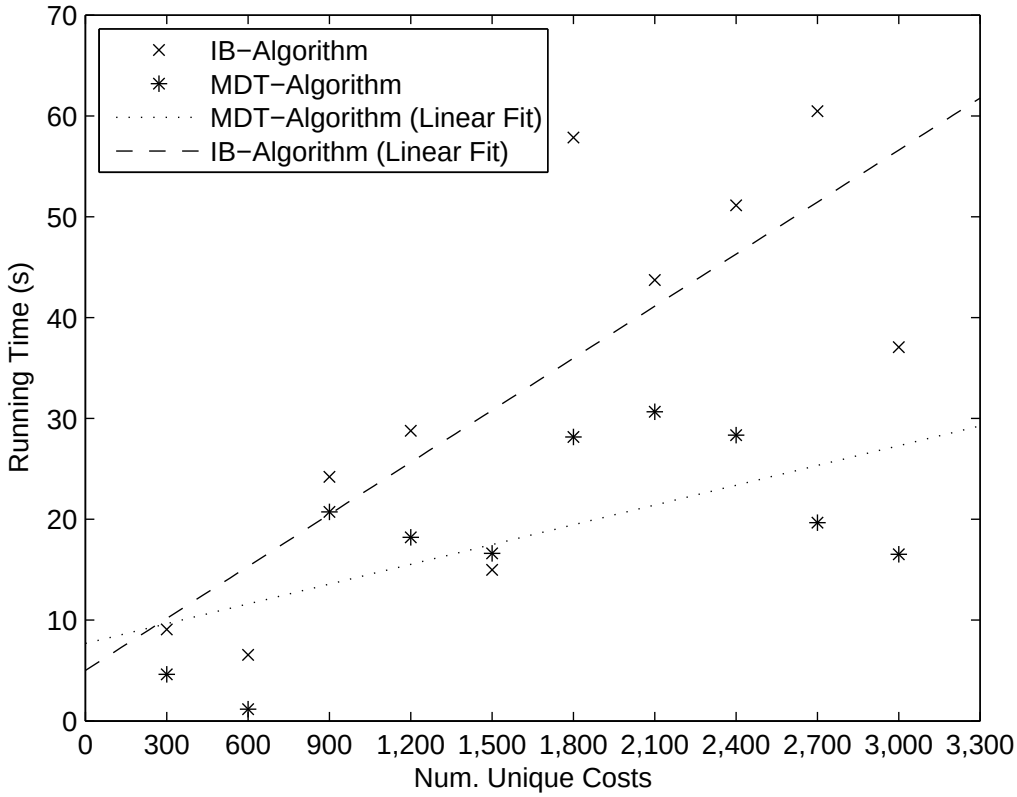


Figure 4.4: Number of unique costs in a 200-vertex problem versus average running-time

the IB algorithm; likewise, for 29 of the problems the IB algorithm has a shorter running-time compared to the MDT algorithm. For 36 of the 59 problems, both algorithms found a solution with an equal objective value. However, for the remaining 23 problems the IB algorithm found a better solution for 20 of those problems. All together, either the MDT or IB algorithm was able to find a provably optimal solution for 23 of the problems. Further, 40 of the 59 problems are provably within 10% of optimality.

To summarize, the theory and results suggest the IB algorithm tends to find better solutions than the MDT algorithm. In terms of running-time, examples exist where either the MDT or IB algorithm runs more quickly than the other in about half the tested cases. Practically speaking, this suggests it might be advantageous to run both algorithms on the same problem simultaneously on a computer with multiple processing cores. Suppose the IB algorithm completed first with a solution of objective value z. Either z can be used as the answer (with the MDT algorithm terminated at this point), or the solution z can be given to the MDT-algorithm to use as the best-known solution. This may allow the MDT-algorithm to terminate more quickly.

			LB	MDT Algorithm			IB Algorithm			Obj		Best	Best
Problem	Size	LB	Time	Gap (%)	Iters	Time	Gap (%)	Iters	Time	Value	Opt?	Obj	Time
burma14	14	120	0.14	11.67	50	0.08	11.67	21	0.08	134		Tie	Tie
ulysses16	16	837	0.28	3.70	11	0.11	3.70	5	0.13	868		Tie	MDT
gr17	17	94	0.02	26.60	47	0.08	26.60	15	0.16	119		Tie	MDT
gr21	21	110	0.03	4.55	34	0.05	4.55	22	0.16	115		Tie	MDT
ulysses22	22	837	1.13	3.70	29	1.05	3.70	10	0.61	868		Tie	IB
gr24	24	33	0.08	0.00	59	0.06	0.00	48	0.30	33	Yes	Tie	MDT
fri26	26	21	0.06	0.00	16	0.03	0.00	16	0.11	21	Yes	Tie	MDT
bayg29	29	23	0.14	26.09	115	0.16	26.09	85	0.99	29		Tie	MDT
bays29	29	36	0.16	5.56	119	0.14	5.56	90	1.17	38		Tie	MDT
swiss42	42	14	0.27	0.00	43	0.08	0.00	36	0.45	14	Yes	Tie	MDT
gr48	48	46	1.47	0.00	210	0.83	0.00	167	6.13	46	Yes	Tie	MDT
hk48	48	138	1.94	15.22	687	61.95	13.04	234	7.81	156		IB	IB
att48	48	156	3.64	23.08	513	19.05	23.08	341	10.49	192		Tie	IB
eil51	51	3	0.11	0.00	23	0.16	0.00	16	1.24	3	Yes	Tie	MDT
berlin52	52	139	2.53	31.65	499	122.22	8.63	103	8.99	151		IB	IB
brazil58	58	912	4.81	25.55	506	136.52	23.36	229	15.00	1,125		IB	IB
st70	70	5	0.39	0.00	47	1.47	0.00	34	5.72	5	Yes	Tie	MDT
eil76	76	2	0.25	0.00	31	0.81	0.00	18	1.84	2	Yes	Tie	MDT
pr76	76	498	9.94	6.02	1,345	426.74	4.82	571	15.56	522		IB	IB
gr96	96	281	595.95	12.10	2,291	1,046.99	11.74	1,168	345.13	314		IB	IB
rat99	99	5	1.58	0.00	67	1.55	0.00	47	9.81	5	Yes	Tie	MDT
rd100	100	43	12.89	0.00	399	10.97	0.00	352	25.19	43	Yes	Tie	MDT
kroC100	100	120	49.50	11.67	2,415	1,915.17	10.83	1,434	86.49	133		IB	IB
kroB100	100	129	38.64	12.40	1,564	224.97	12.40	1,149	72.41	145		Tie	IB
kroE100	100	137	41.55	1.46	1,583	486.28	1.46	1,358	131.97	139		Tie	IB
kroD100	100	140	39.63	0.00	525	27.53	0.00	310	101.64	140	Yes	Tie	MDT
eil101	101	2	0.55	0.00	29	1.86	0.00	17	6.94	2	Yes	Tie	MDT
lin105	105	95	36.36	5.26	1,074	445.59	5.26	939	180.99	100		Tie	IB
pr107	107	877	8.53	0.00	133	75.83	2.62	73	87.27	877	Yes	MDT	MDT
gr120	120	27	13.33	14.81	420	124.09	14.81	350	37.09	31		Tie	IB

Table 4.3: Balanced TSP results on TSPLIB problems from 14 to 120 vertices.

			LB	MDT Algorithm			IB Algorithm			Obj		Best	Best
Problem	Size	LB	Time	Gap (%)	Iters	Time	Gap (%)	Iters	Time	Value	Opt?	Obj	Time
bier127	127	2,915	22.14	55.57	1,037	3,124.72	5.80	163	471.63	3,084		IB	IB
ch130	130	18	13.84	22.22	410	22.69	22.22	303	71.58	22		Tie	MDT
pr136	136	103	21.31	24.27	635	37.50	22.33	491	52.53	126		IB	MDT
gr137	137	403	1,852.28	12.66	4,832	7,155.47	6.20	2,281	1,387.45	428		IB	IB
pr144	144	259	105.83	0.00	635	494.61	0.00	165	242.00	259	Yes	Tie	IB
kroA150	150	89	117.50	7.87	2,020	773.56	2.25	1,641	212.88	91		IB	IB
kroB150	150	103	136.50	5.83	1,934	2,028.67	5.83	1,472	219.53	109		Tie	IB
pr152	152	59	87.75	0.00	795	142.72	0.00	502	244.16	59	Yes	Tie	MDT
u159	159	142	22.48	12.68	546	610.28	0.00	152	88.56	142	Yes	IB	IB
brg180†	180	0	0.02	-	1	0.70	-	1	1.45	0	Yes	Tie	MDT
rat195	195	4	9.03	0.00	75	7.64	0.00	46	17.88	4	Yes	Tie	MDT
d198	198	1,105	83.59	3.17	673	907.88	4.52	425	308.24	1,140		MDT	IB
kroA200	200	71	257.94	7.04	2,087	2,268.16	7.04	1,895	402.36	76		Tie	IB
kroB200	200	81	223.97	4.94	2,123	1,317.14	1.23	1,790	396.95	82		IB	IB
gr202	202	778	1,772.41	63.75	1,372	9,653.92	19.15	312	3,040.72	927		IB	IB
tsp225	225	6	18.16	0.00	210	70.50	0.00	134	122.47	6	Yes	Tie	MDT
pr226	226	450	608.73	304.22	4,417	18,034.09*	12.00	2,644	966.55	504		IB	IB
gr229	229	675	9,751.61	94.67	4,857	18,481.59*	9.93	2,844	5,185.17	742		IB	IB
gil262	262	3	12.61	0.00	149	56.86	0.00	95	153.49	3	Yes	Tie	MDT
pr264	264	238	435.84	308.82	1,862	9,744.22	74.37	602	2,697.05	415		IB	IB
a280	280	3	17.70	0.00	100	31.78	0.00	64	55.17	3	Yes	Tie	MDT
pr299	299	89	1,047.44	0.00	598	126.17	0.00	388	349.36	89	Yes	Tie	MDT
lin318	318	31	521.20	41.94	3,313	18,052.78*	0.00	816	920.81	31	Yes	IB	IB
rd400	400	11	209.72	0.00	629	281.97	0.00	500	541.52	11	Yes	Tie	MDT
fl417	417	199	1,064.74	59.30	1,020	18,024.38*	95.48	522	1,253.56	317		MDT	IB
gr431	431	1,943	24,696.77	45.50	659	18,269.52*	14.77	1,030	21,579.03*	2,230		IB	MDT
pr439	439	810	3,108.25	107.28	1,213	18,048.47*	100.00	50	2,865.61	1,620		IB	IB
pcb442	442	26	851.42	3.85	1,847	451.20	3.85	1,719	936.44	27		Tie	MDT
d493	493	1,191	436.41	37.20	524	18,037.52*	22.50	172	8,979.55	1,459		IB	IB

Table 4.4: Balanced TSP results on TSPLIB problems from 127 to 493 vertices. Note that problem brg180† has a lower bound value of 0, so the objective value gap from the lower bound cannot be calculated due to division by zero. Further, problems with running-times marked by an asterix (*) exceeded the maximum tour search time of 5 hours (18,000s), and so the reported value is the best found up until that point.

Chapter 5

Conclusions

In Chapter 2, we developed a powerful heuristic for the symmetric bottleneck traveling salesman problem (BTSP) which we presented extensive computational results for. The largest problem we solved to optimality involved 31,623 vertices, and larger problems are likely within our reach. Using randomization in a controlled way to guide the heuristic search allowed us to achieve provably optimal solutions in a very reasonable running time. As shown in Chapter 3, this heuristic can be easily modified to solve the maximum scatter traveling salesman problem (MSTSP), as well as the Constrained BTSP where an additional constraint on the total weight (length) of a tour is given.

We defined two new lower bounds for the asymmetric BTSP in Chapter 3 and established a comparison between several well-known lower bounds. Using Theorem 3 we show a recursive way to improve most lower bounding schemes. As with the symmetric BTSP, we presented extensive computational results using our heuristic algorithm on problems of sizes up to 1,000 vertices that produced provably optimal solutions in the majority of instances. Computational results show our heuristic is also effective for solving the asymmetric MSTSP. Finally, we established a $\lceil \frac{n}{2} \rceil$-approximation for the asymmetric BTSP satisfying the triangle inequality, as well as show how to represent any asymmetric BTSP instance on n vertices as a symmetric BTSP instance on $3n$ vertices.

The balanced traveling salesman problem is explored in Chapter 4 and, to the best of our knowledge, this is the first time this problem has been studied in literature, as well as the first NP-Hard balanced optimization problem to be treated with a heuristic. Of the four heuristics we developed, the Modified Double Threshold (MDT) and Iterative Bottleneck (IB) algorithms both proved to be effective. We also show how to adapt these heuristics

to generate a lower bound on the optimal Balanced TSP objective value. Computational results are given for problems up to 500 vertices.

Several interesting theoretical questions emerged from this research. Our approximation bound for the asymmetric BTSP is straightforward, but it would be interesting to explore how this bound could be improved, especially given that a constant bound is available for the symmetric BTSP. Note that a similar difference exists in the case of the approximation ratio for the symmetric and asymmetric TSP, and reducing this gap is an outstanding problem. It would also be interesting to explore a performance ratio of a polynomial approximation algorithm for the Balanced TSP satisfying the triangle inequality.

Bibliography

[1] R.K. Ahuja. Minimum cost to reliability ratio problem. *Computers and Operations Research*, 15(1):83–89, 1988.

[2] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network flows: theory, algorithms, and applications*. Prentice Hall, 1993.

[3] M.W. Altenhöfer, E. Gassner, R. Jacob, and S.O. Krumke. Oracle-guided search in sorted matrices improving balanced flow computation. Report 2008-13, Graz University of Technology, 2008.

[4] D.L. Applegate, R.E. Bixby, V. Chvátal, and W.J. Cook. Concorde TSP Solver. `http://www.tsp.gatech.edu/concorde`, May 2005.

[5] D.L. Applegate, R.E. Bixby, V. Chvátal, and W.J. Cook. *The Traveling Salesman Problem: A Computational Study*. Princeton Series in Applied Mathematics. Princeton University Press, 2006.

[6] E.M. Arkin, Y. Chiang, J.S.B. Mitchell, S.S. Skiena, and T.C. Yang. On the maximum scatter traveling salesman problem. *SIAM Journal on Computing*, 29(2):515–544, 1999.

[7] S. Arora. *Approximation algorithms for geometric TSP*, chapter 5, pages 207–222. In Gutin and Punnen [41], 2002.

[8] C.A. Bailey, T.W. McLain, and R.W. Beard. Fuel-saving strategies for dual spacecraft interferometry missions. *Journal of the Astronautical Sciences*, 49:469–488, 2001.

[9] E. Balas. Personal communication, August 2008.

[10] A. Ben-Dor and B. Chor. On Constructing Radiation Hybrid Maps. *Journal of Computational Biology*, 4(4):517–533, Winter 1997.

[11] M. Bläser. An 8/13-approximation algorithm for the asymmetric maximum TSP. *Journal of Algorithms*, 50(1):23–48, 2004.

[12] M. Bläser. A New Approximation Algorithm for the Asymmetric TSP with Triangle Inequality. *ACM Trans. Algor.*, 4(4):47:1–15, August 2008.

[13] M. Bläser, K.S. Ram, and M. Sviridenko. Improved approximation algorithms for metric maximum ATSP and maximum 3-cycle cover problems. *Operations Research Letters*, 37:176–180, 2009.

[14] J.A. Bondy and C. Thomassen. A short proof of Meyniel's theorem. *Discrete Mathematics*, 19:195–197, 1977.

[15] M. Camerini, F. Maffioli, S. Martello, and P. Toth. Most and least uniform spanning trees. *Discrete Applied Mathematics*, 15(2–3):181–187, 1986.

[16] P. Cappanera and M. G. Scutella. Balanced paths in acyclic networks: Tractable cases and related approaches. *Networks*, 45(2):104–111, 2005.

[17] G. Carpaneto, S. Martello, and P. Toth. An algorithm for the bottleneck traveling salesman problem. *Operations Research*, 32(2):380–389, March–April 1984.

[18] Z.-Z. Chen and T. Nagoya. Improved approximation algorithms for metric max TSP. In *Proc. 13th Annual European Symp. on Algorithms, ESA*, pages 179–190, 2005.

[19] Z.-Z. Chen, Y. Okamoto, and L. Wang. Improved deterministic approximation algorithms for Max TSP. *Information Processing Letters*, 95:333–342, 2005.

[20] N. Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical Report Report 388, CMU, 1976.

[21] J. Cirasella, D.S. Johnson, L.A. McGeoch, and W. Zhang. *The Asymmetric Traveling Salesman Problem: Algorithms, Instance Generators, and Tests*, volume 2153 of *Lecture Notes in Computer Science*, pages 32–59. Springer Berlin / Heidelberg, 2001.

[22] W.J. Cook. VLSI Data Sets. `http://www.tsp.gatech.edu/vlsi/index.html`, March 4 2003.

[23] W.J. Cook. Optimal 85,900-City Tour. `http://www.tsp.gatech.edu/pla85900/index.html`, March 2008.

[24] W.J. Cook. National Traveling Salesman Problems. `http://www.tsp.gatech.edu/world/countries.html`, April 28 2009.

[25] Y. Dai, H. Imai, K. Iwano, N. Katoh, K. Ohtsuka, and N. Yoshimura. A new unifying heuristic algorithm for the undirected minimum cut problem using minimum range cut algorithms. *Discrete Applied Mathematics*, 65(1):167–190, March 1996.

[26] G.B. Dantzig, F.R. Fulkerson, and S.M. Johnson. Solution of a large scale traveling salesman problem. *Operations Research*, 2:393–410, 1954.

[27] N.N. Doroshko and V.I. Sarvanov. A minimax traveling salesman problem and Hamiltonian cycles in powers of graphs. *Vestsi Akad. Navuk BSSR Ser. Fiz. Mat. Navuk*, 6, 1981. (Russian).

[28] C.W. Duin and A. Volgenant. Minimum deviation and balanced optimization: A unified approach. *Operations Research Letters*, 10:43–48, 1991.

[29] J. Edmonds and D.R. Fulkerson. Bottleneck extrema. *Journal of Combinatorial Theory*, 8(3):299–306, April 1970.

[30] D. Eppstein. Minimum Range Balanced Cuts via Dynamic Subset Sums. *Journal of Algorithms*, 23(2):375–385, May 1997.

[31] M.L. Fisher, L. Nemhauser, and L.A. Wolsey. An analysis of approximations for finding a maximum weight Hamiltonian circuit. *Networks*, 12:799–809, 1979.

[32] H. Fleischner. The square of every 2-connected graph is Hamiltonian. *J. Combinatorial Theory, Series B*, 16:29–34, 1974.

[33] A.M. Frieze, G. Galbiati, and F. Maffioli. On the Worst-Case Performance of Some Algorithms for the Asymmetric Traveling Salesman Problem. *Networks*, 12(1):23–39, 1982.

[34] E. Gabovic, A. Ciz, and A Jalas. The bottleneck travelling salesman problem. *Trudy Vycisl. Centra Tartu. Gos. Univ.*, 22:3–24, 1971. (Russian).

[35] Z. Galil and B. Schieber. On finding most uniform spanning trees. *Discrete Applied Mathematics*, 20(2):173–175, 1988.

[36] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.

[37] R.S. Garfinkel and K.C. Gilbert. The Bottleneck Traveling Salesman Problem: Algorithms and Probabilistic Analysis. *Journal of the Association for Computing Machinery*, 25(3):435–448, July 1978.

[38] A. Ghouilà-Houri. Une condition suffisante d'existence d'un circuit hamiltonien. *C.R. Acad. Sci. Paris*, 25:495–497, July 1960.

[39] P.C. Gilmore and R.E. Gomory. Sequencing a one state-variable machine: A solvable case of the traveling salesman problem. *Operations Research*, 12(5):655–679, September–October 1964.

[40] M. Grötschel, M. Jünger, and G. Reinelt. Optimal control of plotting and drilling machines: A case study. *Mathematical Methods of Operations Research*, 35(1):61–84, 1984.

[41] G. Gutin and A.P. Punnen, editors. *The Traveling Salesman Problem and Its Variations*. Combinatorial Optimization. Kluwer Academic Publishers, Secaucus, NJ, USA, 2002.

[42] D. Hartvigsen. *Extensions of Matching Theory*. PhD thesis, Department of Mathematics, Carnegie Mellon University, Pittsburg, PA, 1984.

[43] R. Hassin and S. Rubinstein. Better Approximations for Max TSP. *Information Processing Letters*, 75:181–186, 2000.

[44] R. Hassin and S. Rubinstein. A 7/8-approximation algorithm for metric Max TSP. *Information Processing Letters*, 81(5):247–251, 2002.

[45] M. Held and R.M. Karp. The Traveling Salesman Problem and Minimum Spanning Trees. *Operations Research*, 18:1138–1162, 1970.

[46] M. Held and R.M. Karp. The Traveling Salesman Problem and Minimum Spanning Trees, part II. *Mathematical Programming*, 1:6–25, 1971.

[47] K. Helsgaun. An effective implementation of the Lin-Kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1):106–130, October 2000.

[48] D.S. Hochbaum and D.B. Shmoys. A Unified Approach to Approximation Algorithms for Bottleneck Problems. *Journal of the Association for Computing Machinery*, 33:533–550, 1986.

[49] P. Johanns, T. Lowe, and R. Plante. Selection and sequencing heuristics to reduce variance in gas turbine engine nozzle assemblies. *European Journal of Operational Research*, 132(3):490–504, August 2001.

[50] D.S. Johnson and L.A. McGeoch. Benchmark Code and Instance Generation Codes. `http://www.research.att.com/~dsj/chtsp/download.html`, May 2002.

[51] D.S. Johnson and L.A. McGeoch. *Experimental Analysis of Heuristics for the STSP*, chapter 9, pages 369–444. In Gutin and Punnen [41], 2002.

[52] D.S. Johnson, L.A. McGeoch, F. Glover, and C. Rego. DIMACS Implementation Challenge. `http://www.research.att.com/~dsj/chtsp/`, December 2008.

[53] S. Kabadi. *Polynomial Solveable Cases of the TSP*, chapter 11, pages 697–736. In Gutin and Punnen [41], 2002.

[54] S. Kabadi and A.P. Punnen. *The Bottleneck TSP*, chapter 15, pages 489–584. In Gutin and Punnen [41], 2002.

[55] M.Y. Kao and M. Sanghi. *An Approximation Algorithm for a Bottleneck Traveling Salesman Problem*, volume 3998 of *Lecture Notes in Computer Science*, pages 223–235. Springer Berlin / Heidelberg, 2006.

[56] H. Kaplan, M. Lewenstein, N. Shafrir, and M. Sviridenko. Approximation algorithms for Asymmetric TSP by Decomposing Directed Regular Multigraphs. *Journal of the Association for Computing Machinery*, 52(4):602–626, July 2005.

[57] R.M. Karp. Reducibility among combinatorial problems. In R.E. Miller and J.W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, New York, USA, 1972.

[58] N. Katoh. An ϵ-approximation scheme for combinatorial problems with variance criterion. *Discrete Applied Mathematics*, 35(2):131–141, 1992.

[59] N. Katoh and K. Iwano. Efficient algorithms for minimum range cut problems. *Networks*, 24(7):395–407, 1994.

[60] P.S. Kljaus. A special case of the bottleneck traveling salesman problem. *Vesci Akad. Navuk BSSR Ser. Fi z.-Mat, Navuk*, 1:61–65, 1975. (Russian).

[61] S.R. Kosaraju, J.K. Park, and C. Stein. Long tours and short superstrings. In *Proc. 35th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 166–177, 1994.

[62] A.V. Kostochka and A.I. Serdyukov. Polynomial algorithms with the estimates 3/4 and 5/6 for the traveling salesman problem of the maximum (Russian). *Upravlyaemye Sistemy*, 26:55–59, 1985.

[63] L. Kowalik and M. Mucha. 35/44-Approximation for Asymmetric Maximum TSP with Triangle Inequality. In *Proc. 10th Workshop on Algorithms and Data Structures (WADS'07)*, pages 589–600, 2007.

[64] L. Kowalik and M. Mucha. *Deterministic 7/8-Approximation for the Metric Maximum TSP*, volume 5171/2008 of *Lecture Notes in Computer Science*, pages 132–145. Springer Berlin / Heidelberg, 2008.

[65] J. LaRusic. A Heuristic for Solving the Bottleneck Traveling Salesman Problem. Undergraduate honors thesis, University of New Brunswick Faculty of Computer Science, 2005.

[66] H.T. Lau. *Finding Hamiltonian cycle in the square of a block*. PhD thesis, McGill University, March 1980.

[67] H.T. Lau. Finding EPS-graphs. *Monatshefte für Mathematik*, 92(1):37–40, March 1981.

[68] E.L. Lawler, J.K. Lenstra, A.H.G. Rinooy Kan, and D.B. Shmoys, editors. *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. Wiley, Chinchester, 1985.

[69] M. Leclerc and F. Rendl. Constrained spanning trees and the traveling salesman problem. *European Journal of Operational Research*, 39:96–102, 1989.

[70] M. Lewenstein and M. Sviridenko. A 5/8 Approximation for the Maximum Asymmetric TSP. *SIAM Journal on Descrete Mathematics*, 17(2):237–248, 2003.

[71] S. Lin and B.W. Kernighan. An effective heuristic algorithm for the traveling salesman problem. *Operations Research*, 21:972–989, 1973.

[72] V.F. Magirou and A.J. Nicolitas. The efficient drilling of printed circuit boards. *Interfaces*, 16(4):13–23, August 1986.

[73] G.S. Manku. A linear time algorithm for the bottleneck biconnected spanning subgraph problem. *Information Processing Letters*, 59:1–7, 1996.

[74] S. Martello, W. R. Pulleyblank, P. Toth, and D. de Werra. Balanced optimization problems. *Operations research letters*, 3:275–278, 1984.

[75] E.Q.V. Martins. An algorithm to determine a path with minimal cost/capacity ratio. *Discrete Applied Mathematics*, 8(2):189–194, May 1984.

[76] T. Nemoto. An efficient algorithm for the minimum range ideal problem. *Journal of the Operations Research Society of Japan*, 42(1):88–97, March 1999.

[77] K. Paluch, M. Mucha, and A. Madry. A 7/9-Approximation Algorithm for the Maximum Traveling Salesman Problem. Preprint available at arXiv, December 2009.

[78] C. H. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *J. Comput. System Sci.*, 43(3):425–440, 1991.

[79] R.G. Parker and R.L. Rardin. Guaranteed performance heuristics for the bottleneck traveling salesman problem. *Operations Research Letters*, 2(6):269–272, March 1984.

[80] J.M. Philips, A.P. Punnen, and S. Kabadi. A linear time algorithm for the bottleneck traveling salesman problem on a Halin graph. *Information Processing Letters*, 67(2):105–110, July 1998.

[81] R.D. Plante, T.J. Lowe, and R. Chhandrasekaran. The Product Matrix Traveling Salesman Problem: An Application and Solution Heuristic. *Operations Research*, 35(5):772–783, September–October 1987.

[82] T. Pohle, E. Pampalk, and G. Widmer. Generating similarity-based playlists using traveling salesman algorithms. In *Proceedings of the 8th International Conference on Digital Audio Effects*, 2005.

[83] A.P. Punnen. On combined minmax-minsum optimization. *Computers and Operations Research*, 21:707–716, 1994.

[84] A.P. Punnen. Minmax strongly connected subgraphs with node penalties. *Journal of Applied Mathematics and Decision Sciences*, 2005(2):107–111, 2005.

[85] A.P. Punnen and Y.P. Aneja. Lexicographic balanced optimization problems. *Operations Research Letters*, 32(1):27–30, January 2004.

[86] A.P. Punnen and K.P.K. Nair. A fast and simple algorithm for the bottleneck biconnected spanning subgraph problem. *Information Processing Letters*, 50:283–286, 1994.

[87] A.P. Punnen and K.P.K. Nair. Improved complexity bound for the maximum cardinality bottleneck bipartite matching problem. *Discrete Applied Mathematics*, 55(1):91–93, 1994.

[88] A.P. Punnen and K.P.K. Nair. An Improved Algorithm for the Constrained Bottleneck Spanning Tree Problem. *INFORMS Journal on Computing*, 8(1):41–44, 1996.

[89] A.P. Punnen and K.P.K. Nair. Constrained Balanced Optimization Problems. *Computers & Mathematics with Applications*, 37(9):157–163, May 1999.

[90] S. Rajola and M.S. Tallini. Some necessary conditions for a graph to be hamiltonian. *Journal of Geometry*, 88(1–2):140–148, March 2008.

[91] R. Ramakrishnan, P. Sharma, and A.P. Punnen. New heuristics for the bottleneck traveling salesman problem. Technical report, IIT Kanpur, 2000.

[92] "random: Generate pseudo-random numbers". Python v2.6.1 documentation. `http://docs.python.org/library/random.html`, February 2009.

[93] R.L. Rardin and R.G. Parker. An efficient algorithm for producing a Hamiltonian cycle in the square of a biconnected graph. Technical report, Industrial and Systems Engineering Report Series J-82, Georgia Institute of Technology, 1983.

[94] G. Reinelt. The Traveling Salesman: Computational Solutions for TSP Applications. *Lecture notes in computer science*, 840, 1994.

[95] G. Reinelt. TSPLIB. `http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/`, August 2008.

[96] S. Sahni and T. Gonzales. P-complete approximation problems. *Journal of the Association for Computing Machinery*, 23:555–565, 1976.

[97] V.I. Sarvanov. A minimax traveling salesman problem on a plane: complexity of an approximate solution. *Dokl. Akad. Nauk Belarusi*, 39(6):16–19, 1995. (Russian).

[98] G. Schaar and A. P. Wojda. An upper bound for the Hamiltonicity exponent of finite digraphs (Note). *Discrete Math.*, 164(1):313–316, 1997.

[99] M. G. Scutella. A strongly polynomial algorithm for uniform balanced network flow problem. *Discrete Applied Mathematics*, 81(1–3):123–131, 1998.

[100] A.I. Serdyukov. An algorithm with an estimate for the traveling salesman problem of the maximum. *Upravlyaemye Sistemy*, 25:80–86, 1984. (in Russian).

[101] S.I. Sergeev. Algorithms for the Minimax Problem of the Traveling Salesman. I. An approach based on dynamic programming. *Automation and Remote Control*, 56(7):1027–1032, 1995.

[102] T.B. Spears. *100 Years on the Road: The Traveling Salesman in American Culture*. Yale University Press, 1995.

[103] R. Tarjan. Depth-First Search and Linear Graph Algorithms. *SIAM Journal on Computing*, 1(2):146–160, June 1972.

[104] S. Tigan, E. Iacob, and I. M. Stancu-Minasian. Monotonic balanced optimization problems. In *Annals of the Tiberiu Popoviciu Itinerant Seminar of Functional Equations, Approximation and Convexity*, volume 3, pages 183–197, Cluj-Napoca, Romania, 2005.

[105] W.T. Tutte. A short proof of the factor theorem for finite graphs. *Canad. J. Math.*, 6:347–352, 1954.

[106] G.L. Vairaktarakis. On Gilmore-Gomory's open question for the bottleneck TSP. *Operations Research Letters*, 31(6):483–491, November 2003.

[107] L.G. Valiant. The complexity of computing the permanent. *Theoret. Comput. Sci.*, 8(2):189–201, 1979.

[108] J.A.A. van der Veen. An O(n) algorithm to solve the Bottleneck Traveling Salesman Problem restricted to ordered product matrices. *Discrete Applied Mathematics*, 47:57–75, 1993.

[109] V. Vassilevska, R. Williams, and R. Yuster. All-pairs bottleneck paths for general graphs in truly sub-cubic time. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 585–589, San Diego, CA, June 11–13 2007. ACM New York, NY, USA.

[110] G.J. Woeginger. The P-versus-NP Page. `http://www.win.tue.nl/~gwoegi/P-versus-NP.htm`, June 2009.

[111] Z. Zeitlin. Minimization of maximum absolute deviation in integers. *Discrete Applied Mathematics*, 3(3):203–220, 1981.