Energy-efficient and Safe Routing for Last-mile Logistics

A Thesis submitted for the completion of
requirements for the degree of

Bachelor of Science (Research)

by

Debojjal Bagchi
Undergraduate Program
Indian Institute of Science, Bengaluru

Under the supervision of

Prof. Tarun Rambha
Department of Civil Engineering
Indian Institute of Science, Bengaluru

April 2023

To

My Parents, Friends

Acknowledgements

I wish to begin by conveying my heartfelt gratitude for the encouragement and cooperation I have received throughout the past year. The contributions of numerous individuals have significantly enhanced this work. This work would not have been possible without their support and motivation at every step.

To begin with, I would like to express my sincere gratitude to Prof. Tarun Rambha for giving me an opportunity to work and participate in his group Transportation Networks Lab (TNL). My past two years at the TNL have been an enriching learning experience. He has been an extraordinary mentor and advisor, providing guidance and encouragement throughout the project. I am also grateful for his help in determining my academic pursuits, future studies, and career path. He has devoted countless hours with me in discussing the project and my academic aspirations, maintaining an optimistic attitude despite challenges. I have found his optimism inspiring and learned a lot from it.

Additionally, I am grateful to the professors and seniors in the Civil Engineering, Computer Science and Automation, and Mathematics departments. Their courses and insights have proved invaluable to my work.

Moreover, I am grateful to Prof. Venktesh Pandey, Assistant Professor at North Carolina Agricultural and Technical State University, for his valuable insights and inputs throughout this thesis. In addition to contributing to this work, he greatly aided me in shaping my future academic goals.

I also want to express my deepest gratitude to Prateek Agarwal, my graduate student mentor, for his unwavering support throughout my research. Prateek has been the ideal senior, helping me with code development, guiding me in academic writing, and discussing alternative approaches when we faced difficulties. I also want to sincerely thank Helen

Thomas for her gracious efforts in presenting our work at the TRB Annual Meeting in Washington, going above and beyond to share our research at the conference.

On a personal note, I express my warmest gratitude to my parents, Subhabrata and Ankita Bagchi, for their unwavering belief in me, even at times when I doubted myself. Their encouragement and passion for academia have been indispensable, and I am forever grateful for their love and support.

Lastly, I would also like to mention my second family, i.e., all my close friends, especially Sneha, Manas, Aranya, Shreyasi, Sayan, Adrita, and Shraman, for their invaluable support, encouragement, and patience in withstanding my tantrums. Your presence during both the good and difficult times has been a source of great comfort, and I cannot imagine my life without you all.

DEBOJJAL BAGCHI

Declaration

I, **Debojjal Bagchi**, with SR No. **11-01-00-10-91-19-1-16849** hereby declare that the material presented in the thesis titled

Energy-efficient and Safe Routing
for Last-mile Logistics

represents original work carried out by me in the **Department of Civil Engineering** at **Indian Institute of Science** during the years 2022-2023.

With my signature, I certify that:

- I have not manipulated any of the data or results.
- I have not committed any plagiarism of intellectual property.
- I have clearly indicated and referenced the contributions of others.
- I have explicitly acknowledged all collaborative research and discussions.
- I have understood that any false claim will result in severe disciplinary action.
- I have understood that the work may be screened for any form of academic misconduct.

Date: 26th April, 2023 Student Signature

In my capacity as supervisor of the above-mentioned work, I certify that the above statements are true to the best of my knowledge, and I have carried out due diligence to ensure the the originality of the thesis.

Advisor Name: Dr. Tarun Rambha Advisor Signature

List of Publications

Conferences

1. **Bagchi, D.**, Agarwal, P., Rambha T. and Pandey V. (2023). A Local Search Heuristic for Bi-Criterion Steiner Traveling Salesman Problem. Transportation Research Board, Washington, USA.

2. **Bagchi, D.**, Agarwal, P., Rambha T. and Pandey V. (2022). A Local Search Heuristic for Bi-Criterion Steiner Traveling Salesman Problem. The Institute for Operations Research and the Management Sciences, Indianapolis, USA.

3. **Bagchi, D.**, Agarwal, P., Rambha T. and Pandey V. (2022). A Local Search Based Solution for the Bi-criteria Asymmetric Steiner Traveling Salesman Problem. Transportation Research Group of India 11th Foundation Day Celebration, Surat, India.

Working Papers

1. **Bagchi, D.**, Agarwal, P., Rambha T. and Pandey V. (2023). A Local Search Heuristic for Bi-Criterion Steiner Traveling Salesman Problem with Time Windows.

Abstract

Bi-criterion tours in the Steiner Traveling Salesman Problem, which visit a set of terminals while minimizing total energy consumption and the number of left-hand turns (in right-hand traffic), are ideal for logistics companies. These routes are fuel-efficient and safer since they avoid stopping at junctions. Further, with the advent of electric trucks for delivering goods, there is a need to incorporate electric vehicle energy consumption for delivery logistics. Electric vehicles can save energy by converting kinetic energy at wheels and recharging batteries through the process of regenerative braking. Finally, logistic companies often have to deal with time windows while delivering goods, adding an additional constraint to the complexity of last-mile logistics problems.

The problem of finding efficient routes that are bi-criteria Pareto-optimal in energy consumption and the number of left turns can be modeled as a Bi-objective Steiner Traveling Salesman. Although the solution approaches single criteria, Steiner Traveling Salesman can be extended to bi-objective cases using scalarization, these suffer from scalability issues and cannot deal with time windows. Further, due to regenerative braking, energy consumption in certain sections of the road network can be negative, indicating that energy was gained in those links. Additionally, turns as an objective depend on three nodes, rather than most cost functions like distance and time that depend on a single edge. These complexities cannot be handled using conventional solution approaches for the Steiner Traveling Salesman problem.

This thesis proposes exact and heuristic solution approaches for solving the routing problem for last-mile logistics considering the above-mentioned objectives. The problem is modeled as a Bi-criterion asymmetric steiner traveling salesman problem with time windows (bSTSPTW).

Our solution approach starts with a subset of exact solutions of bSTSPTW without time window constraints. Then it uses a local search with repair heuristics to enumerate new points on the efficiency frontier which satisfy time windows. To do so, several neighborhood structures designed explicitly for the bSTSPTW problem have been proposed. Optimal paths between delivery points are computed using a multi-objective label setting algorithm on a line graph, which makes counting the number of turns easier. In addition to the local search, we also propose two new exact methods based on brute force search — Graph layering, and *Graph Concatenation.*

To benchmark the results, we extend the single-objective integer program to the multi-objective case using the scalarization technique. Results indicate that the exact approaches work better with smaller graphs and fewer terminals, and the local search works best with large graphs with a larger number of terminals. Empirically, local search outperforms the integer program in most cases.

Lastly, the thesis presents an empirical case study in which the aim is to find efficient routes for the last-mile logistics for Amazon, satisfying time windows when delivering goods using electric vehicles in the city of Austin, Texas, USA. We start by quantifying the energy consumption in each road link of Austin and create an underlying graph with turns at intersections, energy consumption, and time to travel road links as attributes for the graph. Our heuristic provides multiple route options to drivers, all of which are Pareto-optimal.

Contents

List of Tables

List of Figures

List of Algorithms

List of Acronyms

TSP	Traveling Salesman Problem
TSPTW	Traveling Salesman Problem with Time Windows
ATSP	Asymmetric Traveling Salesman Problem
STSP	Steiner Traveling Salesman Problem
STSPTW	Steiner Traveling Salesman Problem with Time Windows
SOTSP	Single Objective Traveling Salesman Problem
MOTSP	Multi Objective Traveling Salesman Problem
bSTSP	Bi-criterion Steiner Traveling Salesman Problem
bSTSPTW	Bi-criterion Steiner Traveling Salesman Problem with Time Windows
MOSTSP	Multiobjective Steiner Traveling Salesman Problem with Time Windows
EVs	Electric Vehicles
IP	Integer Program
MNDS	Maximal Non-Dominated Subset
HV	Hypervolume
CM	Cardinality Metric

Keywords

eco-routing • energy-efficient routing • last mile logistics • steiner traveling salesman problem • line graphs • traveling salesman problem with time windows • multiobjective traveling salesman problem

Chapter 1

Introduction

Two roads diverged in a yellow wood,
and sorry I could not travel both.

Robert Frost,
The Road Not Taken

Chapter Overview. This chapter presents an introduction to the thesis. Specifically, Section 1.1 provides the motivation behind the work. Next, Section 1.2 describes the problem statement and outlines the significant contributions of the work. Section 1.3 details the various datasets used in the thesis, and Section 1.4 introduces the notation and definitions used in the present work. Finally, Section 1.5 provides an outline for the rest of the thesis.

1.1 Motivation

The transportation sector is responsible for 19% global energy use and 23% of global carbon-dioxide emissions. These numbers are expected to grow by nearly 50% by 2030 (De Cauwer et al., 2017). Electric Vehicles (EVs) offer the substantial potential to reduce these environmental impacts. However, a limited number of studies have been on harnessing EVs' benefits, specifically in route design by logistic companies. In addition, about 1.3 million commuters succumb to fatal traffic crashes every year, costing countries

an average of 3% of their Gross Domestic Product (World Health Organization, 2004). A significant portion of these fatal accidents directly involves the logistics sector. Recent studies have revealed that the transportation and logistics sector has a fatal injury rate of approximately twice the average across all industries (RTITB, 2023). Consequently, there is an ardent need for energy-efficient and safe routing, especially within the context of logistics companies.

Traveling Salesman Problem (TSP) is one of the most famous problems in combinatorial optimization because of its applicability in several scenarios ranging from logistics to circuit board design. The objective of TSP is to find the shortest tour, visiting each node of a graph exactly once. Several TSP variants have been formulated to fit practical scenarios. One such example is the Steiner Traveling Salesman Problem (STSP). Broadly speaking, the STSP has two main variants—*node routing* and *arc variant*. In the node routing variant of the problem, STSP aims to find the tours that visit a subset of nodes (designated as *terminals*). However, the customer locations and depots are generally located along the edges and not at junctions. In the arc routing version, the objective is to enumerate the shortest tours which visit a subset of edges (also designated as *terminals*) at least once and return to the origin. Thus, the arc routing version of the problem is more significant for delivery systems.

However, the shortest tour (in terms of distance or time) is often not the only criterion of interest. Generalized versions of STSP can include multiple objectives, leading to the Multi-Objective Steiner Traveling Salesman Problem (MOSTSP). One such problem is to find bi-objective tours that minimize energy consumption and left turns (in countries with right-hand traffic), which is relevant for city logistics. The idea was motivated by United Parcel Service (UPS) research, which pointed out how routing vehicles along paths that minimize both distance and the number of left turns could reduce emissions, fuel usage, and accidents. Holland et al. (2017) observed that such tours not only reduced carbon emissions by 20,000 tons and fuel consumption by 10 million gallons but also increased the successfully delivered packages by 350,000 per year by UPS. There have been other extensive studies revealing that turn movements at an intersection have a significant impact on accident rates, fuel consumption, and emissions (Wood, 2020). Choi (2010), for instance, observed that 61% of accidents occur in left-hand turns. The infographic in Figure 1.1 illustrates some of the benefits of using left-hand turns.

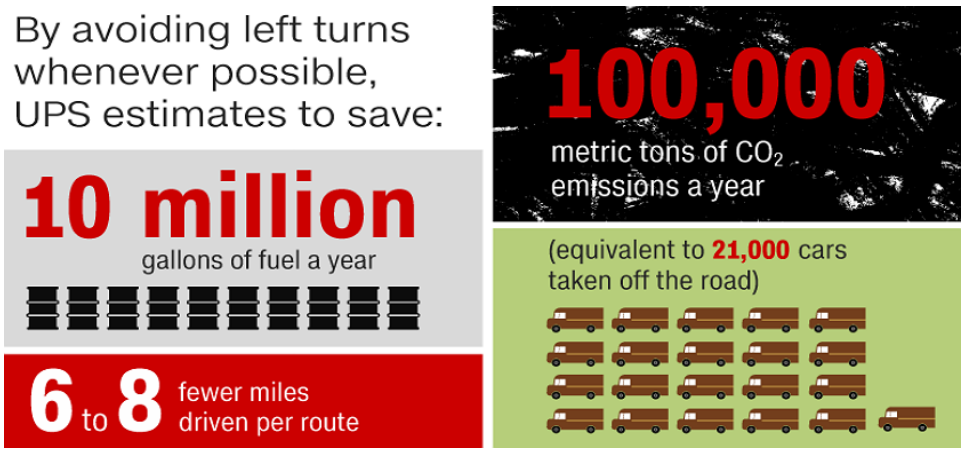


Figure 1.1: Infograph illustrating benefits of reducing left hand turns
(Source: https://www.bromfordlab.com/lab-diary/2019/4/9/why-do-ups-trucks-only-turn-right)

With the advent of electric vehicles, energy-efficient routing has become a significant area of research. A wide range of road characteristics influences the energy consumption on a road segment. In the context of last-mile delivery, factors such as road gradient, speed, and distance traveled significantly affect energy consumption. There have been numerous studies to estimate energy consumption in routes by modeling various road characteristics. However, most of these studies are limited to building an energy consumption model for range prediction using physics-based models, linear regression, and neural networks. This poses a gap in incorporating road characteristic-based energy consumption in routing models, especially in the context of TSP and real-life logistics.

Moreover, electric vehicles have the potential to save energy through regenerative braking in specific road segments. Figure 1.2 illustrates this with an example. In the case of regenerative braking, the potential energy stored in a vehicle at a height is transformed into kinetic energy at wheels during downward road gradients. This kinetic energy can be harnessed to recharge the vehicle's battery. This scenario represents a rare instance where the underlying graph of the problem exhibits negative edge weights, signifying road segments where energy is gained rather than expended.

Though necessary, turning direction and energy consumption by incorporating road characteristics, particularly in TSP, is rarely studied in the literature. The uniqueness of energy consumption and turns as parameters are that while an edge's energy consumption depends upon two adjacent nodes, the turning direction at a junction is determined by

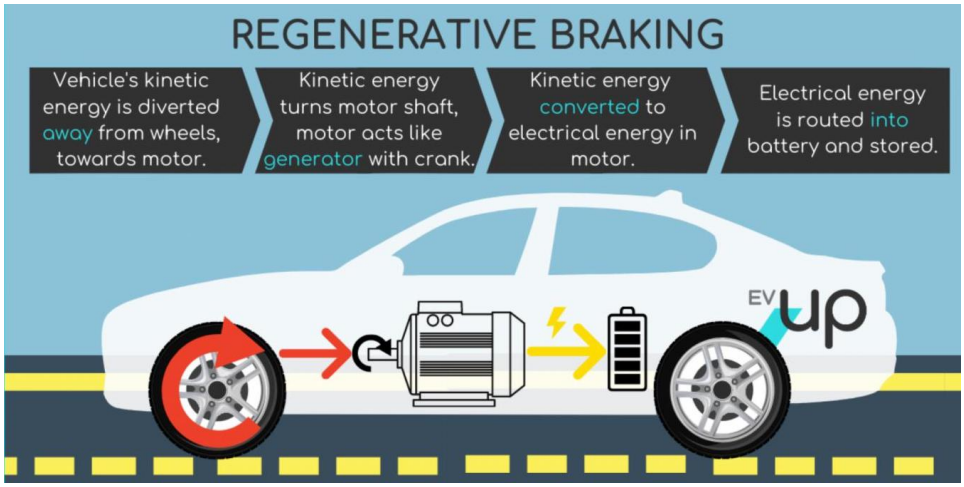


Figure 1.2: Illustration of regenerative braking
(Source: https://www.evup.com.au/about-evup/ev-news/how-does-regenerative-braking-work)

three nodes (i.e., predecessor, current, and successor node). This adds another layer of complexity as the cost of the shortest path for multi-objective problems is calculated using algorithms like Martins (1984). However, such algorithms fail when turns are taken as objectives since the edge weights depend on three nodes (due to turning direction).

Another variant of the (TSP) is the TSP with Time Windows or TSPTW. In the TSPTW, the objective is to find a minimum-cost tour in which each city is assigned a specific time window. This variant is more relevant in the context of last-mile delivery logistics, as it considers the time-sensitive nature of deliveries. The TSPTW can be further extended to incorporate (STSP), resulting in the Steiner Traveling Salesman Problem with Time Windows (STSPTW). In this version, only the terminal nodes have associated time windows.

1.2 Problem Statement and Contributions

In this study, we propose energy-efficient and safe routing for last-mile logistics. To do so, we model the problem as a bi-objective version of STSPTW. We refer to this model bi-objective Steiner Traveling Salesman Problem with Time Windows or bSTSPTW. The model considers energy consumption and the number of turns as the two objectives. The goal is to find all Pareto-optimal tours that visit all the terminals (at least once) and

return to the depot. This approach provides drivers and logistics companies with a set of route options from which they can select based on their preferences. The problem can be defined as follows.

Consider a graph representing a road network and a list of delivery/customer locations such that some or all customers have a time window representing the period within which goods must be delivered. The goal is to find a set of Pareto-optimal tours that minimize energy consumption and turns at intersections while delivering goods to each terminal, adhering to the specified time windows, and returning to the depot.

Our energy consumption model is a physics-based formulation derived from Travesset-Baro et al. (2015). This model captures both road characteristics and vehicle attributes. The energy consumption model is given in Equation 1.1.

$$dE = \frac{1}{3600}\left[mg(f\cos\phi + \sin\phi) + \frac{1}{2}\left(\rho C_x A\frac{(v_{EV}+v_w)^2}{3.6}\right) + (m+m_f)\frac{dv}{dt}\right]ds \quad (1.1)$$

where dE is the energy required to drive distance ds between points i and j, m is the vehicle mass, m_f is the fictive mass of rolling inertia, f is the coefficient of rolling resistance, ϕ is the road gradient angle, ρ is the air density, C_x is the drag coefficient of the vehicle, A is the equivalent vehicle cross-section, v_{EV} is the vehicle speed between points i and j, and v_w is the wind speed in the opposing direction to driving.

Note that dE can be negative when there are downward slopes, thereby causing energy to be gained. This energy can be saved using regenerative braking with a regenerative efficiency η. By incorporating these considerations, the bSTSPTW model enables the development of energy-efficient and safe routing strategies tailored to the unique constraints of electric vehicles and modern logistics systems. The major contributions of this paper are as follows:

- Using concepts from graph theory, we propose two exact brute force methods to solve bSTSP and bSTSPTW—*Graph Concatenation* and *Graph layering*.
- Though effective, the brute force approach fails to give satisfactory results for real-world situations involving large graphs. Since we focus on practical applications, we

also introduce a new local search-based heuristic method for finding near-optimal bi-objective tours with time windows.

- To provide a more comprehensive analysis, we also formulate the bSTSP and bSTSPTW problem as an Integer Program (IP).

All experiments are conducted on real-world networks on Amazon delivery routes in the city of Austin, Texas, in the United States (Merchan et al., 2022). Note that the study assumes right-hand moving traffic. Thus, throughout the study, "turns" refer to left turns. Similar methods can be used for left-handed traffic.

1.3 Description of Dataset

In this study, we use real-world data from the 2021 Amazon Last Mile Routing Research Challenge Dataset (Merchan et al., 2022). The dataset contains route details such as terminal coordinates and time windows associated with these terminals for historical routes used by Amazon drivers for last-mile delivery in six cities in the United States. With the coordinates of terminals extracted from this dataset, an underlying road network graph is created using OpenStreetMaps (OpenStreetMap contributors, 2023). Road link lengths and velocities are obtained from OpenStreetMaps, followed by computing turn direction at intersections. To do so, the bearing angle is computed for every pair of links at an intersection. For two points A and B with respective latitude-longitude coordinate (ϕ_1, λ_1) and (ϕ_2, λ_2), the bearing angle can be computed using Equation 1.2, where ϕ_1 and ϕ_2 are the latitudes, $\Delta\lambda$ is the difference in longitudes $(\lambda_2 - \lambda_1)$, With this angle, we assign each turn as "left" or "right".

$$\text{bearing} = \tan^{-1}\left(\frac{\sin(\Delta\lambda)\cdot\cos(\phi_2)}{\cos(\phi_1)\cdot\sin(\phi_2) - \sin(\phi_1)\cdot\cos(\phi_2)\cdot\cos(\Delta\lambda)}\right) \tag{1.2}$$

To determine road gradients, we use the United States Geological Survey (USGS) Elevation Point Query Service (EPQS) (USGS, 2023). Subsequently, we calculate the time spent in each link as the length of the link divided by the velocity in that link. These data enable the characterization of the network to simulate real-life environments

to a considerable extent. Figure 1.3 summarizes some vehicular parameters. We use the parameters for lion-8 all-electric trucks for our analysis. The reason for choosing lion-8 was that Amazon plans to procure 2500 lion-8 trucks for logistic purposes (Electrive, 2021).

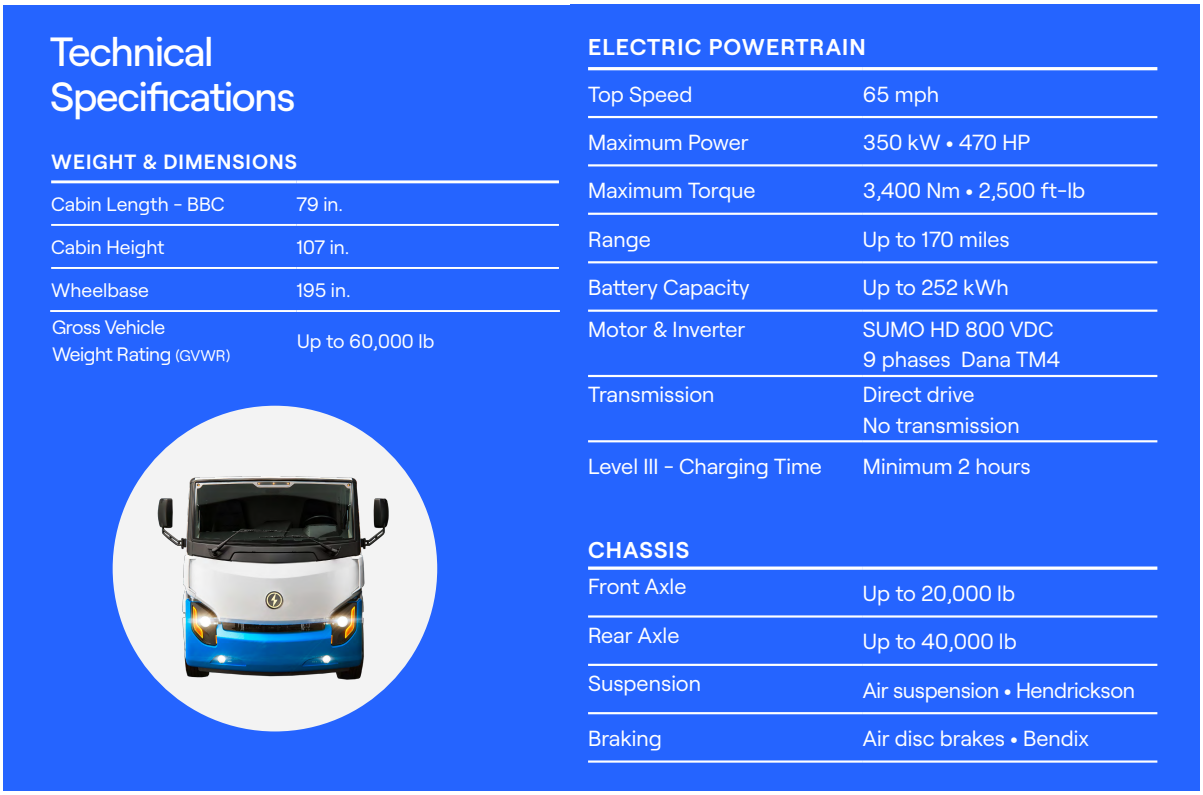


Figure 1.3: Technical specifications of lion-8 electric truck
(Source: https://pages.thelionelectric.com/lion8/)

Figure 1.4 and 1.5 illustrate the energy and slope distribution for all six cities. It is worth noting that all road gradients with a downward slope do not contribute to energy capture. This is because a significant gradient is required to have energy capture through regenerative braking (see equation 1.1). In the figures, negative energy indicates energy is gained and conserved through regenerative braking, while positive energy signifies energy is lost. A regenerative efficiency of 70% was assumed.

Figure 1.6 illustrates the distribution of energies on the Austin network. Green links signify energy could be saved through regenerative braking on these links. Furthermore, Figure 1.7 presents the location of terminals for one of the routes in the dataset.

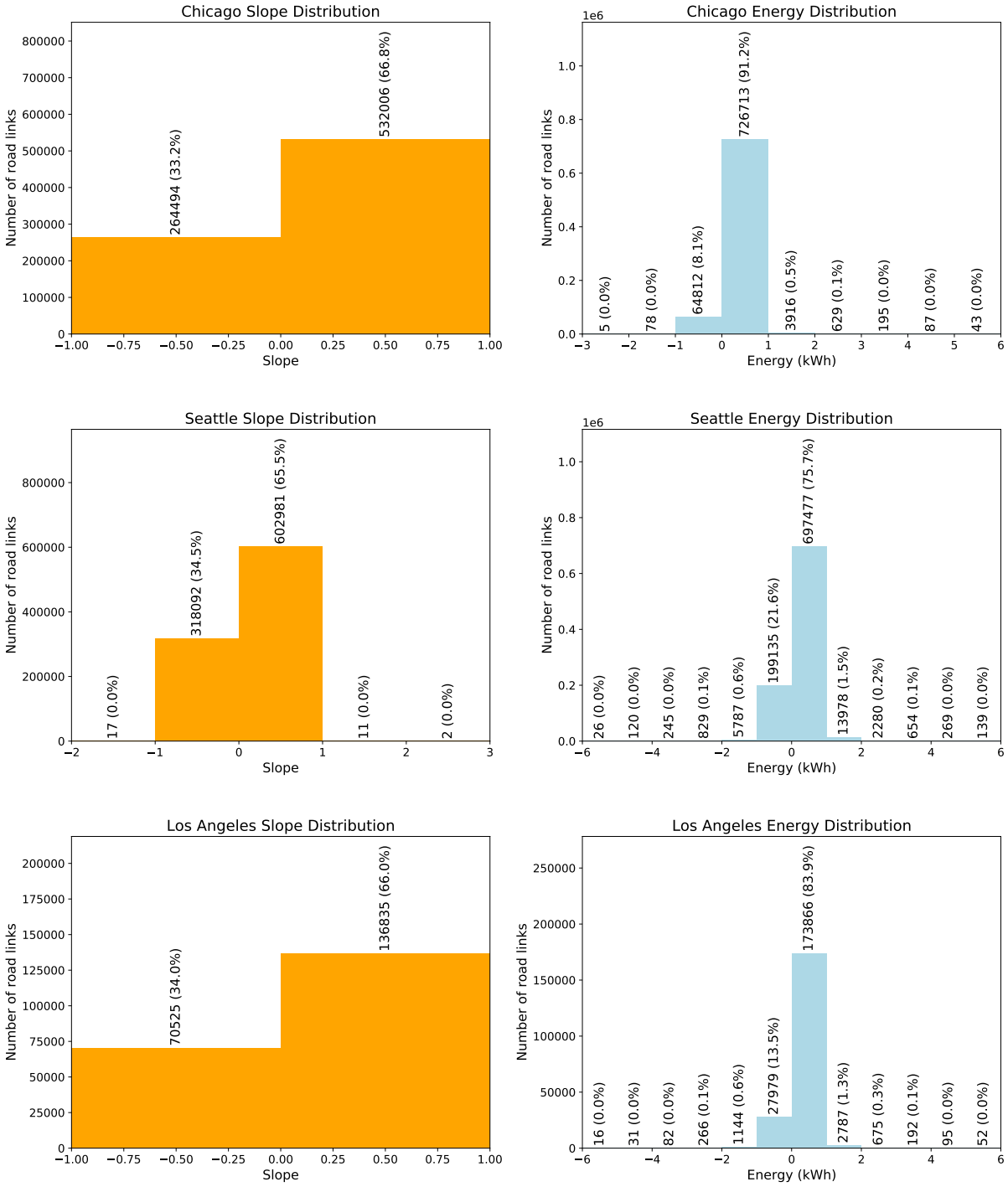


Figure 1.4: Slope and energy distribution for Chicago, Seattle, and Los Angeles

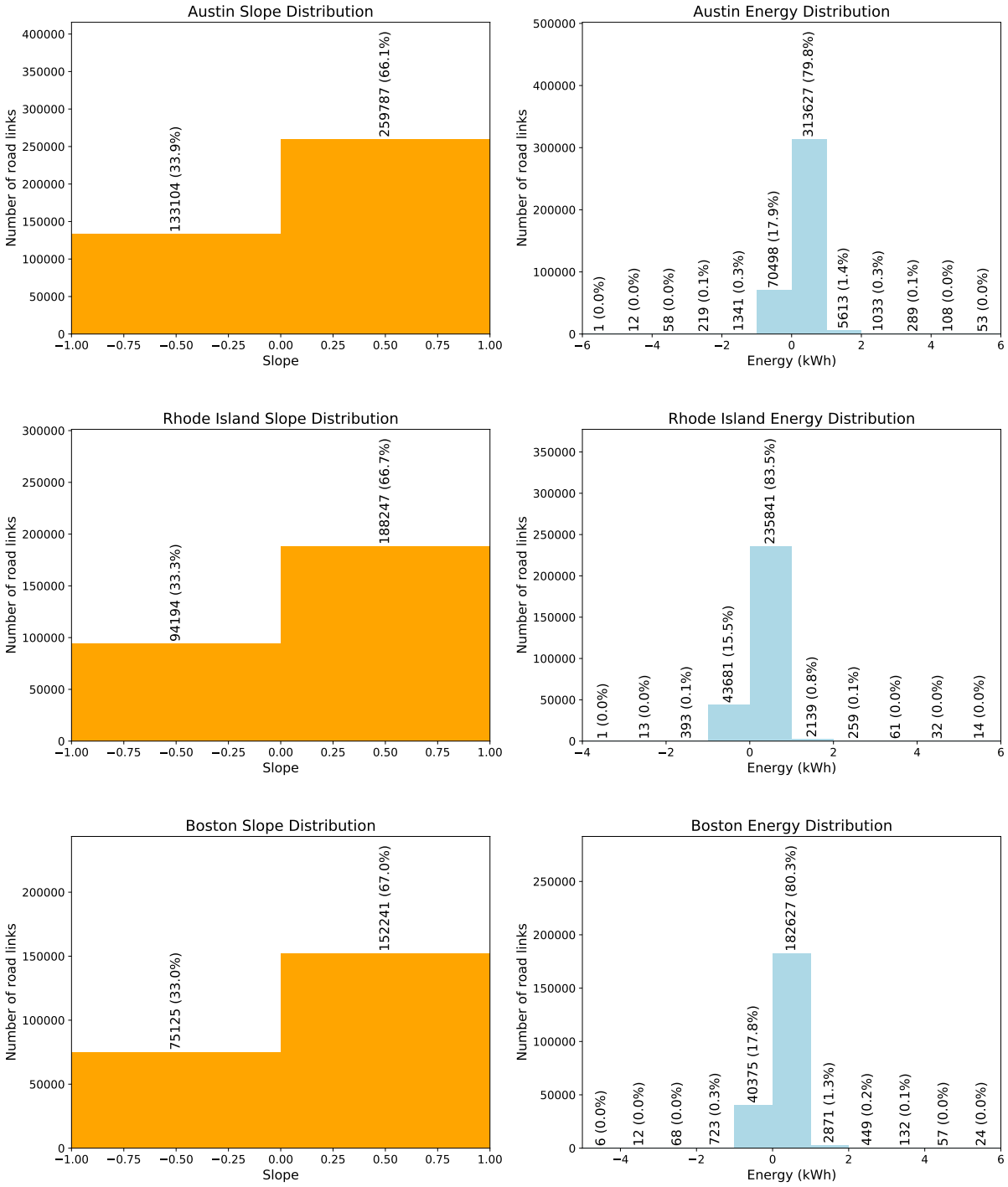


Figure 1.5: Slope and energy distribution for Austin, Rhode Island, and Boston

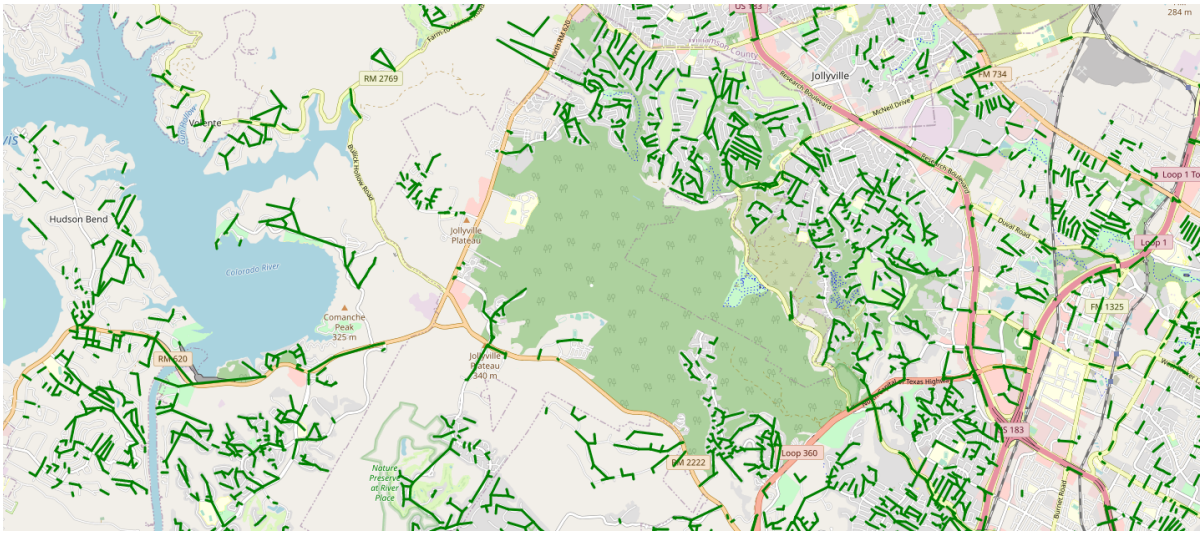


Figure 1.6: Distribution of energies on the Austin network
(energy could be saved through regenerative braking in green links.)

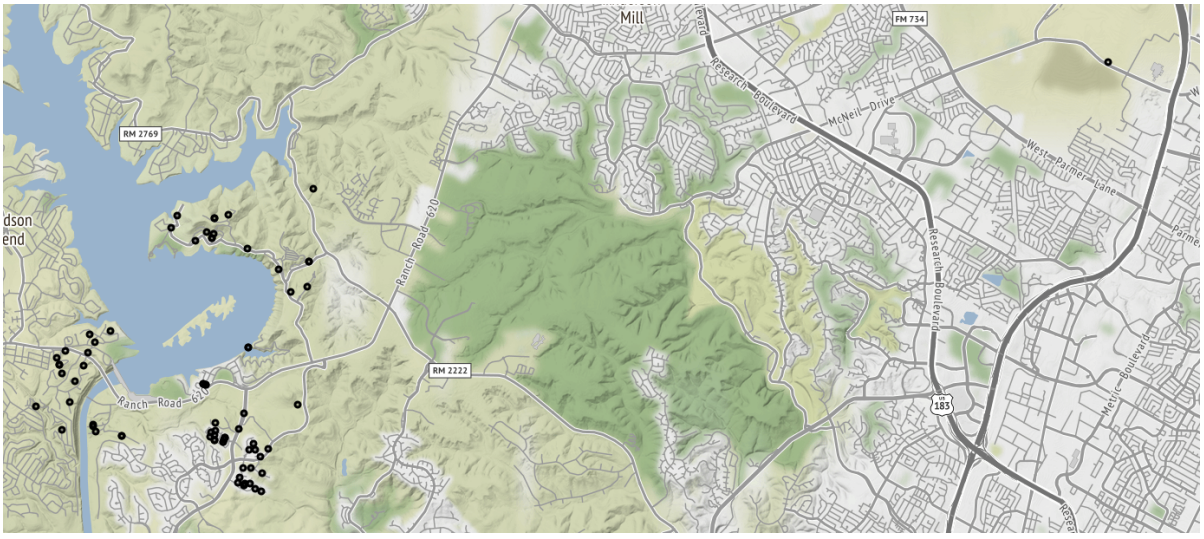
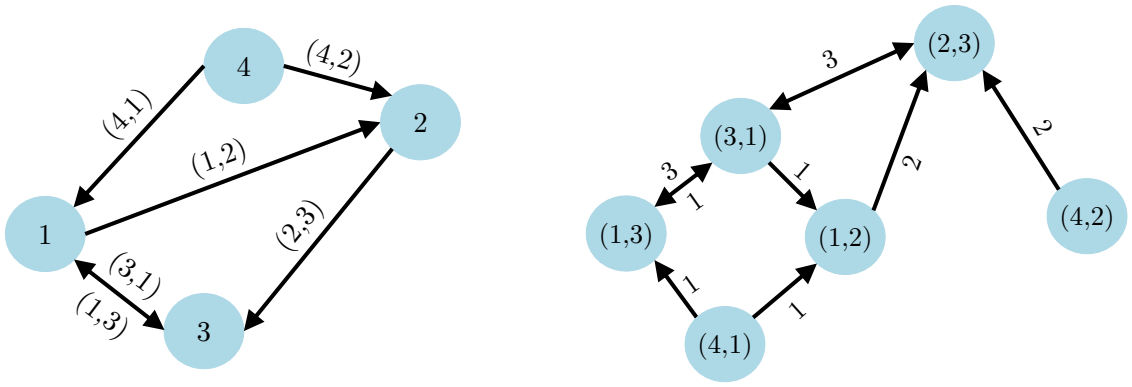


Figure 1.7: Location of terminals
(black dots represent terminals, the terminal at the right corner represents the depot)



(a) Original graph G **(b)** line graph L

Figure 1.8: Line graph illustration

1.4 Definitions and Notations

This section introduces the terminology used in the present work. The notations are summarized in Table 1.1.

Let $G = (V, E, E_R)$ denote a directed graph where V and E are the set of nodes and edges. $E_R \subseteq E$ represents the set of terminal edges or *terminals* that are required to be visited. Let the cardinality of E_R be n_R. Let $L = (V', E', V'_R)$ denote the line graph of G in which the edges (nodes) in G are nodes (edges) in L.

Line Graph: As described earlier, bi-criteria shortest-path algorithms like Martins (1984) can not be directly applicable in our case as one of the cost attributes (number of turns) depends on three nodes. To overcome this issue, we solve the problem on a line graph. See Figure 1.8 for illustration. The Subfigure 1.8b is the line graph for Subfigure 1.8a. An edge $e = (v_1, v_2)$ of the original graph is labeled as a node (v_1, v_2) in the line graph. Similarly, a node v in the original graph is labeled as an edge v in the line graph.

Consider the following example w.r.t Figure 1.8. While traveling along the path $(1 - 2 - 3)$ in Subfigure 1.8a, the turning direction at node 2 depends on the edges $(1, 2)$ and $(2, 3)$. However, in the line graph L in Subfigure 1.8b, it depends on the two corresponding nodes labeled $(1, 2)$ and $(2, 3)$. However, the energy attribute of the edge connecting these two nodes (labeled as 2 in Subfigure 1.8b) now depends only on one node. Hence to define the energy attribute in the line graph, we make the following

assumption: The energy attribute for edges of the line graph is set to half the sum of energy consumption in the corresponding edges in the original graph. For example, for edge labeled 2 (connecting nodes $(1,2)$ and $(2,3)$) in Subfigure 1.8b, $d_e = \frac{1}{2} * (d_{e_1} + d_{e_2})$, where e_1 and e_2 are the edges (1,2) and (2,3) in Subfigure 1.8a. In other words, we assume that the delivery locations are at the mid-points of the edges. This assumption is reasonable since customer locations and depots are generally located along the edges and not at junctions.

Note that since our focus is on real-world networks, which are often directed, there can be more edges in the line graph than nodes in the original graph. We now present a few definitions required throughout the thesis.

A *path* p is a set of nodes visited in a particular order. A *tour* is a closed path visiting each terminal of G. Let X be the set of all feasible tours. For a tour $x \in X$, its *cost* $z(x)$ is defined as a two-dimensional vector $[z_1(x), z_2(x)]$, where $z_1(x)$ and $z_2(x)$ are the cost attributes for the bi-criteria case. Given tours $x_1, x_2 \in X$, x_2 is said to be *dominated* by x_1, denoted by $z(x_1) \preceq z(x_2)$, iff $z_1(x_1) \leq z_1(x_2)$ and $z_2(x_1) \leq z_2(x_2)$. In other words, x_1 *dominates* x_2 iff x_1 is better than x_2 in all attributes. A tour x_1 is said to be *optimal* if no solution exists $x_2 \in X$ such that x_1 is dominated by x_2. Let $\hat{E}$ denote the *Pareto optimal set* containing all the optimal solutions. We say a set S is non-dominated if no element in S is dominated by another element of S. A Maximal Non-Dominated Subset (MNDS) of a set S, denoted by MNDS(S), is its largest non-dominated subset.

A *Decision Space* comprises all candidate solutions to a multi-objective problem. An *objective function* maps the decision space into a *Objective Space*. In our case, the decision space includes all tours, and the objective space contains the corresponding energy consumption and turn count for each tour in the decision space. A *Pareto-front* or a *efficiency frontier* refers to the image of a *Pareto optimal set* on the objective space. Figure 1.9 (Maier et al., 2019) explains decision space, objective space, non-dominated solutions, and Pareto front when both objectives are minimized.

bSTSP and bSTSPTW formulation: Given a graph $L = (V', E', V'_R)$, the objective of the bSTSP is to find a set of Pareto optimal tours that return to origin after visiting all terminal nodes in V'_R (at least once). To do so, the traveler starts with $n_R - 1$ units of a commodity and drops off exactly one unit at each terminal node (no commodity

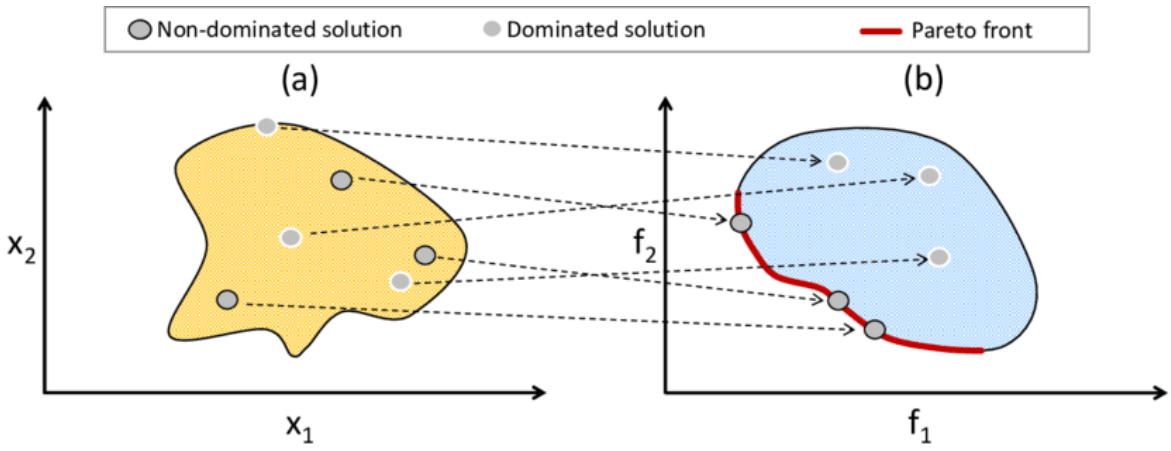


Figure 1.9: Mapping of a decision space onto an objective space
The non-dominated solutions are shown in black border circles, and the Pareto front is shown in red. (Maier et al., 2019)

is delivered at the source node). Note that there is no restriction on the number of times a node is traversed. If each terminal $v \in V_R$ has a time window $[R_v, D_v]$ associated with it, then the formulation is called to be bSTSPTW

The following additional notations are required for our analysis. For each edge $e \in E'$, its *cost* c_e is represented as a two-dimensional vector $[d_e, t_e]$ representing energy consumption and turn count for e in the line graph L. Let $\delta^+(v)$ $(\delta^-(v))$ denote the set of edges in E' whose tail (head) node is v . For each edge $e \in E$, let f_e represent the amount of the commodity passing through the edge e.

The above notations (along with a few extra notations that will be defined in subsequent chapters) have been summarized in Table 1.1.

Table 1.1: Notations and abbreviations

Notation	Description
$G = (V, E, E_R)$	Original graph with V vertices, E edges, and E_R terminals
$L = (V', E', V'_R)$	line graph with V' vertices, V'_R terminals and E' edges
X	Set of feasible tours of bSTSP
S	Any generic set
E	Pareto optimal set for bSTSPTW
$\hat{E}$	Non-dominated set which approximates E
$\hat{X}$	Faesible set for bSTSP containig promising tours for bSTSPTW
P	Set of paths
$[R_v, D_v]$	Time window at node v
$\delta^+(v)$	Set of edges in E' whose tail node is v
$\delta^-(v)$	Set of edges in E' whose head node is v
i	General indices
n_R	Cardinality of set E_R (and V'_R)
v, v_i	Any vertex
v_o	Source node (depot)
e	Any edge
$p, p', \hat{p}, \tilde{p}, p_i$	Path $\in P$
$z(p) = [z_1(p), z_2(p)]$	Cost of the path p
d_e	Energy parameter of edge $e \in E'$
t_e	Turn of an edge $e \in E'$
f_e	Amount of commodities passing through the edge $e \in E'$
s, s_v	Time (Time required to reach node v from depot
α, β	Scalerization parameters for IP
τ	Maximum run time for an algorithm
k	Expected number of Pareto-optimal solutions
δ	Phase budget (a heuristic parameter used in local search)

1.5 Thesis Outline

The remainder of the thesis is structured as follows. Chapter 2 provides a brief literature review of TSP and related problems. Next, Chapter 3 introduces the exact methods and integer program formulations for bSTSP and bSTSPTW. A new local search method for bSTSPTW is proposed in Chapter 4. Chapter 5 compares the proposed techniques' performance on different real-world network instances. Finally, Chapter 6 summarizes our findings and suggests potential extensions to the current research.

Note: Throughout the thesis, we also use such Notes to present extra information that requires special attention.

Chapter 2

A Brief History of Various Variants of Traveling Salesman Problem

Those that fail to learn from history are doomed to repeat it.

Winston Churchill

Chapter Overview. This chapter provides a brief literature review on TSP and its related problems. In particular, Section 2.1 encapsulates the original TSP problem. Within this section, Subsection 2.1.1 explores Integer Program-based methods, while Subsection 2.1.2 discusses the use of heuristics. Section 2.2 explores different variants of the TSP problem. Subsection 2.2.1 addresses the asymmetric version of TSP, Subsection 2.2.2 explores the multiobjective version, Subsection 2.2.3 discusses the TSP problem incorporating time windows, and Subsection 2.2.4 examines the Steiner version.

Note that the TSP is a nearly century-old problem that has been extended to several variants and remains one of the most challenging problems in combinatorial optimization. This chapter only covers of few of its variants, which are essential in the present context.

2.1 Traveling Salesman Problem (TSP) – The vanilla version

Traveling Salesman Problem (TSP) seeks to find the shortest tour that visits all nodes of a given graph and returns to the origin. This is a specific TSP variant called Single Objective Symmetric Traveling Salesman Problem (SOSTSP). As the name suggests, SOTSP aims to find tours based on a single objective (typically distance or time). The graph in this problem (unless stated otherwise) is assumed to be symmetric, where the cost to travel from node i to node j, denoted by c_{ij}, is the same as that of c_{ji}. This makes sense when the cost matrix has objectives like distance. However, when there are objectives like time, the time to travel in different directions on the same links may vary, necessitating an asymmetric version of the TSP. In the following sections, we will discuss some popular methods for solving the symmetric and asymmetric TSP (ATSP).

2.1.1 Integer Programs

The earliest formulations of TSP were modeled as Integer Linear Programs (ILPs), with the Dantzig, Fulkerson, and Johnson (DFJ) formulation being among the first (Dantzig et al., 1954). This formulation introduces a decision variable x_{ij} for each edge, representing whether the edge (i, j) is part of the optimal tour. The edge (i, j) cost is denoted by c_{ij}. The objective function aims to minimize the total cost of the tour. While the constraints (2.20) and (2.21) ensure that each node should be visited exactly once, constraint (2.4) eliminates subtours. Subtours are defined as closed paths that visit only a subset of nodes.

TSP DFJ Formulation

$$\text{minimize} \quad \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} c_{ij} x_{ij} \tag{2.1}$$

$$\text{subject to} \quad \sum_{j=1, j\neq i}^{n} x_{ij} = 1, \quad \forall i = 1, \ldots, n \tag{2.2}$$

$$\sum_{i=1, i\neq j}^{n} x_{ij} = 1, \quad \forall j = 1, \ldots, n \tag{2.3}$$

$$\sum_{i \in S} \sum_{j \in S} x_{ij} \leq |S| - 1, \quad \forall S \subseteq \{1, \ldots, n\}, 2 \leq |S| \leq n - 1 \tag{2.4}$$

$$x_{ij} \in \{0, 1\}, \quad \forall i, j = 1, \ldots, n, i \neq j \tag{2.5}$$

Letchford et al. (2013) explained that the above formulation has an exponential number of subtour elimination constraints. Consequently, even for small values of n, solving this formulation using modern solvers is impractical. A notable contribution in the field is by Laporte (1992), in which the authors attempted to solve the problem by relaxing some constraints.

Another famous ILP formulation for the standard TSP is the Miller, Tucker, and Zemlin (MTZ) formulation by Desrochers and Laporte (1991). The MTZ formulation addresses the exponential constraint issue present in the DFJ formulation by introducing additional decision variables. It modifies the subtour elimination constraints by adding decision variables u_i for each node, representing the node's position within the tour. The modified subtour elimination constraint is in (2.10).

TSP MTZ Formulation

$$\text{minimize} \quad \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} c_{ij} x_{ij} \tag{2.6}$$

$$\text{subject to} \quad \sum_{j=1, j\neq i}^{n} x_{ij} = 1, \quad \forall i = 1, \ldots, n \tag{2.7}$$

$$\sum_{i=1, i\neq j}^{n} x_{ij} = 1, \quad \forall j = 1, \ldots, n \tag{2.8}$$

$$u_i - u_j + n x_{ij} \leq n - 1, \quad \forall i, j = 2, \ldots, n, i \neq j \tag{2.9}$$

$$x_{ij} \in \{0, 1\}, \quad \forall i, j = 1, \ldots, n, i \neq j \tag{2.10}$$

$$u_i \in \{1, \ldots, n-1\}, \quad \forall i = 2, \ldots, n \tag{2.11}$$

The MTZ formulation has $\mathcal{O}(n^2)$ variables and $\mathcal{O}(n^2)$ constraints. Unfortunately, the relaxation of this formulation to its Linear Program (LP) results in very weak bounds, even weaker than those of the DFJ formulation (Letchford et al., 2013).

To address this issue, Gavish and Graves (1978) proposed a Single Commodity Flow (SCF) formulation. The LP relaxed bound of SCF formulation is situated between the MTZ and DFJ formulations (Padberg and Sung, 1991). In the SCF, the salesman begins at the origin with $n-1$ commodities and drops off one commodity at each node visited, effectively depleting all commodities upon returning to the origin. This constraint inherently acts as a sub-tour elimination constraint by imposing an order on the nodes visited. To model this, a decision variable f_{ij} is assigned to each edge, representing the number of commodities flowing through edge (i, j). The SCF formulation consists of $\mathcal{O}(n^2)$ variables and $\mathcal{O}(n)$ constraints.

TSP SCF Formulation

$$\text{minimize} \quad \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} c_{ij} x_{ij} \tag{2.12}$$

$$\text{subject to} \quad \sum_{j=1, j\neq i}^{n} x_{ij} = 1, \quad \forall i = 1, \ldots, n \tag{2.13}$$

$$\sum_{i=1, i\neq j}^{n} x_{ij} = 1, \quad \forall j = 1, \ldots, n \tag{2.14}$$

$$\sum_{j=1, j\neq i}^{n} f_{ij} - \sum_{j=2, j\neq i}^{n} f_{ji} = 1, \quad \forall i = 2, \ldots, n \tag{2.15}$$

$$f_{ij} \leq (n-1) x_{ij}, \quad \forall i, j = 1, \ldots, n, i \neq j \tag{2.16}$$

$$x_{ij} \in \{0, 1\}, \quad \forall i, j = 1, \ldots, n, i \neq j \tag{2.17}$$

$$f_{ij} \geq 0, \quad \forall i, j = 1, \ldots, n, i \neq j \tag{2.18}$$

In addition to the three prominent ILP formulations for (TSP) discussed earlier, several alternative ILP formulations also exist in the literature. Gouveia and Voß (1995) offers an exhaustive survey that delves into the various formulations proposed for addressing TSP.

2.1.2 Heuristic Methods

TSP being a $\mathcal{NP}$-hard, solving them as ILP is generally time-consuming (Laporte, 1992). Consequently, many heuristic approaches for solving TSP have been proposed in the literature. In the present context, three main categories are important: Greedy methods, Insertion heuristics, and Node-exchange methods. (Laporte, 1992) showed that these methods generally present a trade-off as while some guarantee specific bounds on worst-case performance or others are known to empirically known to perform well. The most simplistic heuristic is the nearest-neighbor search (Karkory and Abudalmola, 2013), which starts with the origin, finds the nearest node, locates the nearest node to the previous node, and continues iteratively. The nearest-neighbor search falls under the category of greedy heuristics. Additionally, various insertion heuristics have been proposed for TSP. Hassin and Keinan (2008) provides a review of the insertion heuristics for TSP. These methods iteratively insert unvisited cities into the current tour at the position

that minimizes the increase in tour length. Christofides' algorithm (Christofides et al., 1981) is an approximation algorithm that guarantees a solution within 3/2 times the optimal value for the symmetric TSP. The authors create a minimum spanning tree of the underlying graph, find the minimum-weight perfect matching, and finally form a tour for TSP. Several other minimum-spanning tree-based heuristics have been proposed for TSP. Held and Karp (1970) offers a review of such methods.

Numerous node-exchange-based methods, or swapping, are also popular for heuristically solving TSP. These swaps are called 2-opt if only two pairs of nodes can be swapped (Croes, 1958). Extensions of this method include 3-opt (Bock, 1958), 4-opt (Blazinskas and Misevicius, 2011), and k-opt (Potvin et al., 1989). In general, lower-order swaps are faster, while larger-order swaps cover broader areas of the search space (Lin, 1973). The Lin-Kernighan (LK) heuristic (Lin and Kernighan, 1973) addresses this effectively by iteratively improving a tour through a sequence of edge swaps (2-opt, 3-opt, and so on) and accepting the best improving move. Through several implementation improvements, the LK heuristic was further enhanced by Helsgaun (2000). Other heuristics for solving TSP include Tabu search (Basu, 2012), Genetic algorithms (Basu, 2012), Simulated annealing (Skiscim and Golden, 1983), and Ant colony optimization (Stützle et al., 1999).

2.2 Variants of TSP

2.2.1 Asymmetric TSP (ATSP)

Before proceeding to the ATSP problem, defining the *Assignment Problem* in the context of ATSP (Burkard, 1979) is essential. In the Assignment Problem or AP, we model a linear program that seeks to determine a tour or a set of subtours. Note that AP, combined with subtour elimination constraints, gives the solution to TSP. The ILP formulations designed for symmetric TSP also apply to ATSP. Öncan et al. (2009) carried out a comprehensive comparison of 24 such formulations, providing valuable insights into their performance in real-world instances.

Assignemnt Problem (AP)

$$\text{minimize} \quad \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} c_{ij} x_{ij} \tag{2.19}$$

$$\text{subject to} \quad \sum_{j=1, j\neq i}^{n} x_{ij} = 1, \quad \forall i = 1, \ldots, n \tag{2.20}$$

$$\sum_{i=1, i\neq j}^{n} x_{ij} = 1, \quad \forall j = 1, \ldots, n \tag{2.21}$$

$$x_{ij} \in \{0, 1\}, \quad \forall i, j = 1, \ldots, n, i \neq j \tag{2.22}$$

Johnson et al. (2007) showed that the heuristics designed for symmetric TSP generally fail or are ineffective for ATSP. However, promising results can be expected for ATSP when the gap between the AP bound and the optimal tour length is small. The Repeated Assignment heuristic is known to have the best bound on worst-case performance (Frieze et al., 1982). This heuristic builds on the Christofides heuristic for symmetric TSP. However, while the bound for the Christofides heuristic for symmetric TSP is 3/2, it can be shown that the bound for the Repeated Assignment heuristic is $log_2 N$ (Frieze et al., 1982). Another popular heuristic for ATSP is Zhang's heuristic (Zhang, 1993), which uses a depth-first search to explore solutions and solve the assignment problem-based branch and bound for the ATSP.

In the case of local search-based heuristics, the 2-opt sequence does not yield effective results for ATSP, as it inverts parts of a tour (Matsushita et al., 2011). 3-opt-based methods, on the other hand, work effectively (Sierksma, 1994). However, in contrast to all possible swaps, only one swap that does not invert subpaths is selected (Frieze et al., 1982). The Kanellakis-Papadimitriou (KP) heuristic builds on the LK heuristic for symmetric TSP (Kanellakis and Papadimitriou, 1980). It employs only odd k-opt swaps and uses a depth-first search to find the best k. Furthermore, a double-bridge type swap is implemented for diversification. Other popular heuristics for ATSP include Helsgaun's Heuristic (Helsgaun, 2000), which transforms an ATSP instance into a symmetric TSP instance, Iterated 3-opt that performs $10n$ iterations for n cities, and Iterated KP (Cirasella et al., 2001), which introduces a variable depth search in the k-opt phase.

2.2.2 Multi Objective Traveling Salesman Problem (MOSTSP)

Tours with the least cost or distance are often impractical in real-world applications (Holland et al., 2017). However, given that the SOTSP is an $\mathcal{NP}$-complete problem, incorporating multiple objectives in the MOTSP variant introduces an additional layer of complexity. Consequently, the most general approach for solving the MOTSP involves employing heuristics. These methods rely on multi-objective metaheuristics (MOMHs) based on evolutionary algorithms or local search. A comparison of the performance of evolutionary algorithms for MOTSP is presented in Qamar et al. (2018). Angel et al. (2004) introduced a ds-2opt (dyna-search) neighborhood designed explicitly for MOTSP, which explores the non-dominated neighborhood for a tour. A dynamic program was utilized to calculate all independent 2-opt moves comprising the neighborhood. While this method yielded satisfactory results, it proved to be computationally expensive (Lust and Teghem, 2010a).

Paquete et al. (2004) proposed a Pareto local search method akin to Angel et al. (2004), with the distinction that the neighborhood in Paquete et al. (2004) did not include any tour generated from a dominated solution, thus limiting the neighborhood. Paquete and Stützle (2003) presented a new two-phase local search and demonstrated superior results compared to conventional algorithms. In the first phase, single-criteria versions of the problem were explored, while the second phase investigated the multi-objective version using weights on the objectives. Gupta and Warburton (1987) employed a scalarization technique that assigned weights to different objectives using the Tchebycheff function for scalarization. Lust and Teghem (2010b) expanded upon this direction, proposing a two-phase Pareto local search. In the first phase, the convex subset of efficient solutions was computed using weights on the objectives, while in the second phase, the remaining efficient solutions were proposed using swaps.

Li (2005) introduced an interesting new concept of "attraction" for each objective, which consisted of sets comprising solutions for individual objectives. Different parts of the attraction set were combined based on a local search, which calculated the overlap of these paths concerning a new tour.

2.2.3 Traveling Salesman Problem with Time Windows (TSPTW)

The TSPTW (Traveling Salesman Problem with Time Windows) is an extension of the classic TSP that integrates time constraints into the problem (De Schreye, 1999). In this version, each node has a specific time window within which it must be visited. The objective is to find a TSP tour that adheres to the time windows for each node. This additional layer of complexity makes the problem more challenging to solve. Nonetheless, the TSPTW is commonly utilized in real-world applications, particularly in modern logistics.

Several approaches have been proposed to solve the TSPTW. One popular method involves applying penalties to nodes where time windows are infeasible (Da Silva and Urrutia, 2010). Ohlmann (2007) employed a simulated annealing procedure with a variable penalty approach, while Helsgaun (2017) extended the Lin-Kernighan (LK) heuristic for symmetric TSP by using penalties separately rather than adding them to the objective. Numerous studies (e.g., Li (2008); Tsitsiklis (1992); Kara et al. (2013)) have sought to minimize travel times as an objective, which is a comparatively more straightforward method than cases where the objective differs from time. Li (2008) utilized a dynamic programming-based approach to solving the TSPTW, while urgen Antes and Derigs (1995) employed a two-phase method. Initial solutions without time windows were generated using parallel nearest-neighborhood searches in the first phase. The authors calculated the earliest and latest arrival times at a node to address the time windows. A local search method was employed in the second phase to improve these tours. Solomon (1987) proposed a construction heuristic for the TSPTW, using a time-oriented nearest-neighbor search, and further applied insertions, deletions, and exchanges in an improvement heuristic to obtain better solutions. Cheng and Mao (2007) utilized an ant colony-based optimization for solving the TSPTW. Russell (1977) compared various exchange-based local search heuristics (such as 2-opt and 3-opt) in the context of the TSPTW and proposed a new variant of the 2-opt for the TSPTW that selected nodes without violating time window feasibility.

2.2.4 Steiner Traveling Salesman Problem (STSP)

As discussed in Chapter 1, finding optimal tours that pass through a subset of nodes (as opposed to visiting all the nodes) of the graph makes the solution particularly useful in the context of logistics (Zhang et al., 2015) and routing (Letchford et al., 2013). Such problems are popularly known as Steiner Traveling Salesman Problem (STSP).

A single objective STSP instance can be transformed into a TSP problem by computing the shortest path between each pair of terminals and generating a complete graph (Álvarez-Miranda and Sinnl, 2019). The nodes of the complete graph represent the terminals, and the edges are assigned costs equivalent to the shortest paths between the terminals. By solving the TSP on this complete graph, a solution for the corresponding STSP can be obtained. However, computation of a complete graph can sometimes be expensive (Letchford et al., 2013). As a result, the latest trend in the field is to directly solve the STSP problem with heuristics and IP formulations. For example, Letchford et al. (2013) proposed a single commodity flow based IP formulation, and Interian and Ribeiro (2017) proposed a Greedy Randomized Adopted Search Procedure (GRASP) based on 2-opt moves for symmetric STSP. The common underlying motivation behind all these methods is that the computation of a complete graph is time-intensive. Álvarez-Miranda and Sinnl (2019) compared the techniques proposed by Letchford et al. (2013) and Interian and Ribeiro (2017) to those that utilize a complete graph and then solve using state-of-the-art TSP solvers like CONCORDE (Applegate et al., 2003). The authors concluded that, in most practical cases, the latter approach yields faster results.

In recent years, a few variants of the problem have been explored—edges having correlated stochastic costs (Letchford and Nasiri, 2015), and accommodating online and real-time edge blockages (Zhang et al., 2015). However, literature on this particular variant is relatively sparse.

Table 2.1 highlights the key features of some relevant studies and how they differ from the present work.

Paper	Problem	Obj.	Type	Solution	Approach
Dantzig et al. (1954)	TSP	Single	Asymmetric	Heuristic	Interger Program
Desrochers and Laporte (1991)	TSP	Single	Asymmetric	Heuristic	Interger Program
Gavish and Graves (1978)	TSP	Single	Asymmetric	Heuristic	Interger Program
Lust and Teghem (2010b)	TSP	Multi	Symmetric	Heuristic	Pareto local search
Yan et al. (2003)	TSP	Multi	Asymmetric	Heuristic	Evolutionary Multi-objective Optimization
Agrawal et al. (2021)	TSP	Multi	Asymmetric	Heuristic	Hybrid of local search and genetic algorithm
Lin and Kernighan (1973)	TSP	Single	Symmetric	Heuristic	Local search
Kanellakis and Papadimitriou (1980)	TSP	Single	Asymmetric	Heuristic	Local search
Gabrel et al. (2020)	STSP	Single	Symmetric	Exact & Heuristic	Branch-and-Cut & Branch-and-Price
Letchford et al. (2013)	STSP	Single	Asymmetric	Exact	SCF based IP formulation
Interian and Ribeiro (2017)	STSP	Single	Symmetric	Heuristic	2-opt based GRASP
Álvarez-Miranda and Sinnl (2019)	STSP	Single	Asymmetric	Exact	Complete graph and TSP solvers (CONCORDE)
Zhang et al. (2015)	STSP	Single	Asymmetric	Heuristic	Minimum spanning trees
Li (2008)	TSPTW	Single	Asymmetric	Exact	Dynamic programming
Solomon (1987)	TSPTW	Single	Symmetric	Heuristic	Local search
Cheng and Mao (2007)	TSPTW	Single	Asymmetric	Heuristic	Ant colony optimization
Russell (1977)	TSPTW	Single	Symmetric	Heuristic	Exchange-based local search heuristics
Helsgaun (2017)	TSPTW	Single	Symmetric	Heuristic	Local search
Ohlmann (2007)	TSPTW	Single	Symmetric	Heuristic	Simulated annealing
This work	**STSPTW**	**Multi**	**Asymmetric**	**Exact & Heuristic**	**IP and Local search**

Table 2.1: Research gaps

Chapter 3

Exact Solution Methods

The test of a good algorithm is not whether it's clever, but whether it works.

Brian Kernighan

Chapter Overview. This chapter presents exact methods for solving both the bSTSP and bSTSPTW. Specifically, Section 3.1 presents integer program formulations for bSTSP and bSTSPTW. Next, Section 3.2 introduces two exact brute force methods for bSTSP — *Graph layering* and *Graph Concatenation*. Finally, Section 3.3 outlines the limitations of the exact methods and sets the context for the next chapter.

Note: The primary contributions of this chapter are the two brute force methods for bSTSPTW and bSTSP, and the IP formulation for bSTSPTW.

3.1 Integer Program for bSTSP and bSTSPTW

Recall that the problem bSTSP involves finding all Pareto-optimal tours for the bi-objective Steiner traveling salesman problem. Previously, Letchford et al. (2013) proposed an IP formulation for solving STSP. We extend it to the bi-objective case using the scalarization technique discussed in Dial (1996). The decision variable is x_e, which is

equal to one if the edge e is part of the optimal tour, else 0.

bSTSP IP Formulation

$$\min \quad \sum_{e \in E'} (\alpha d_e + \beta t_e) x_e \tag{3.1}$$

$$\text{s.t.} \quad \sum_{e \in \delta^+(v)} x_e \geq 1 \qquad \forall\, v \in V'_R \tag{3.2}$$

$$\sum_{e \in \delta^+(v)} x_e = \sum_{e \in \delta^-(v)} x_e \qquad \forall\, v \in V' \tag{3.3}$$

$$\sum_{e \in \delta^-(v)} f_e - \sum_{e \in \delta^+(v)} f_e = 1 \qquad \forall\, v \in V'_R \setminus \{v_o\} \tag{3.4}$$

$$\sum_{e \in \delta^-(v)} f_e - \sum_{e \in \delta^+(v)} f_e = 0 \qquad \forall\, v \in V' \setminus V'_R \tag{3.5}$$

$$0 \leq f_e \leq (n_R - 1) x_e \qquad \forall\, e \in E' \tag{3.6}$$

$$x_e \in \{0, 1\} \qquad \forall\, e \in E' \tag{3.7}$$

The objective function (3.1) minimizes the weighted sum of the two parameters for each edge, where α and β indicate the weights of the two parameters. Constraint (3.2) stipulates that the traveler departs from each terminal at least once. Constraint (3.3) imposes that the traveler must depart the node they visit. Constraints (3.4) and (3.5) ensure that while exactly one unit of the commodity is delivered at each terminal, no deliveries are made at non-terminals. Constraint (3.6) ensures that commodities only pass through an edge that is a part of the optimal tour. Boolean constraints on the decision variables are imposed in Constraint (3.7).

While bSTSP without time windows may be useful, it often falls short in practical applications for delivery companies. Therefore, to effectively handle time windows, we build upon the works of Dash et al. (2012). For each node $v \in V_R$, consider following additional parameters:

- R_v: Earliest arrival time, i.e., delivery cannot be made before this.
- D_v: Latest departure time, i.e., delivery must be made before this

Further, let θ_e represent travel time on an edge e. The IP for bSTSPTW can thus be formulated as follows. Constraint (3.8) – (3.13) are same as before. Constraint (3.14) guarantees that arrival times at nodes increase along any path within a tour, while Constraint (3.15) ensures that the time of arrival at a terminal v adheres to the time window (R_v, D_v). The decision variable s_v refers to the arrival time at node v.

bSTSPTW IP Formulation

$$\min \quad \sum_{e \in E'} (\alpha d_e + \beta t_e) x_e \tag{3.8}$$

$$\text{s.t.} \quad \sum_{e \in \delta^+(v)} x_e \geq 1 \qquad \forall\, v \in V'_R \tag{3.9}$$

$$\sum_{e \in \delta^+(v)} x_e = \sum_{e \in \delta^-(v)} x_e \qquad \forall\, v \in V' \tag{3.10}$$

$$\sum_{e \in \delta^-(v)} f_e - \sum_{e \in \delta^+(v)} f_e = 1 \qquad \forall\, v \in V'_R \setminus \{v_o\} \tag{3.11}$$

$$\sum_{e \in \delta^-(v)} f_e - \sum_{e \in \delta^+(v)} f_e = 0 \qquad \forall\, v \in V' \setminus V'_R \tag{3.12}$$

$$0 \leq f_e \leq (n_R - 1) x_e \qquad \forall\, e \in E' \tag{3.13}$$

$$s_{v_i} + \theta_e - (1 - x_e) M_e \leq s_{v_j} \qquad \forall\ e = (v_i, v_j)\ \in E' \tag{3.14}$$

$$R_v \leq s_v \leq D_v \qquad \forall\ v \in V_R \tag{3.15}$$

$$x_e \in \{0, 1\} \qquad \forall\ e \in E' \tag{3.16}$$

Note that theoretically, though the efficiency frontier given by an IP is always convex, studies (Clímaco and Pascoal (2016)) have shown that the actual efficiency frontier may/may not be convex. Thus, the current formulation of the IP only provides the convex subset of the Pareto optimal frontier.

Let the solution to the IP for any value of α and β be given by IntProg$(\alpha, \beta, \mathit{time})$ where *time* refers to the amount of time for which the IP is allowed to run. For exact solutions, we set $\mathit{time} = \infty$. For a configuration of α and β, let the output of IntProg be p, d, and t, where p represents the optimal path constructed from the set of edges e s.t. $x_e = 1$, and d and t are the cost of the parameters (energy consumption and the number

of turns, respectively) of the optimal path.

To enumerate the points on the efficiency frontier, we first find the extreme points by setting (α, β) equal to $(0, 1)$ and $(1, 0)$. These two cases correspond to the SOSTSP, where we optimize only one objective (energy or the number of turns). For all subsequent iterations, we use the scalarization approach described below.

Setting the ratio of the weights (i.e., $\frac{\alpha}{\beta}$) as the slope of the line connecting the extreme points, we employ a recursive algorithm as delineated by Dial (1996). The algorithm is depicted in Algorithm 1 and will be referred to as MidPaths. Line 1 sets the slope of the line. Line 2 solves the IP using IntProg function. Next, Lines 3–4 check for the termination criteria, i.e., if the new point generated by IntProg coincides with the points from which the slope was initially derived, then the recursion is exited. The newly found path p is added to the set of optimal paths E in Line 5.

The complete algorithm is summarized in Algorithm 2 (IPSolver). The algorithm takes three inputs: Line graph L, maximum run time τ, and expected number of tours on efficiency frontier k, and returns the set of optimal paths E. Lines 1 and 2 call IntProg for the extreme points. Line 3 adds the newly discovered tours to set E. Line 4 calls the recursive function MidPaths until the termination condition is satisfied. Further details on how to calibrate parameters τ and k are provided in Chapter 5.

Algorithm 1 MidPaths

Input: $\hat{d}, \tilde{d}, \hat{t}, \tilde{t}, \tau, k$

Output: E : Optimal Paths

1: $\gamma \leftarrow \frac{\hat{d}-\tilde{d}}{\hat{t}-\tilde{t}}$
2: $p, d, t \leftarrow$ IntProg$(\alpha = \gamma, \beta = 1, time = \tau/k)$
3: **if** $(\hat{d}, \hat{t}) = (d, t)$ **or** $(\tilde{d}, \tilde{t}) = (d, t)$ **or** *time_spent* $\frac{(k-2)\tau}{k}$ **then**
4: **return**
5: $E \leftarrow E \cup \{p\}$
6: MidPaths$(d, \hat{d}, t, \hat{t}, \tau, k)$
7: MidPaths$(\tilde{d}, d, \tilde{t}, t, \tau, k)$

Algorithm 2 IPSolver

Input: $L = (V', E', V'_R)$, τ, k

Output: E : Optimal paths for MOSTSP

1: $E \leftarrow \varnothing$

2: $\hat{p}, \hat{d}, \hat{t} \leftarrow$ IntProg$(\alpha = 0, \beta = 1, time = \tau/k)$ ▷ Endpoint of efficiency frontier

3: $\tilde{p}, \tilde{d}, \tilde{t} \leftarrow$ IntProg$(\alpha = 1, \beta = 0, time = \tau/k)$ ▷ Endpoint of efficiency frontier

4: $E \leftarrow E \cup \{\hat{p}, \tilde{p}\}$

5: MidPaths$(\hat{d}, \tilde{d}, \hat{t}, \tilde{t}, \tau, k)$ ▷ Midpoints of efficiency frontier

Note: The output of the IP is the set of edges belonging to the optimal tour. However, the actual tour sequence has to be constructed from these edges as part of the post-processing, and this exercise is not trivial because of potential revisits.

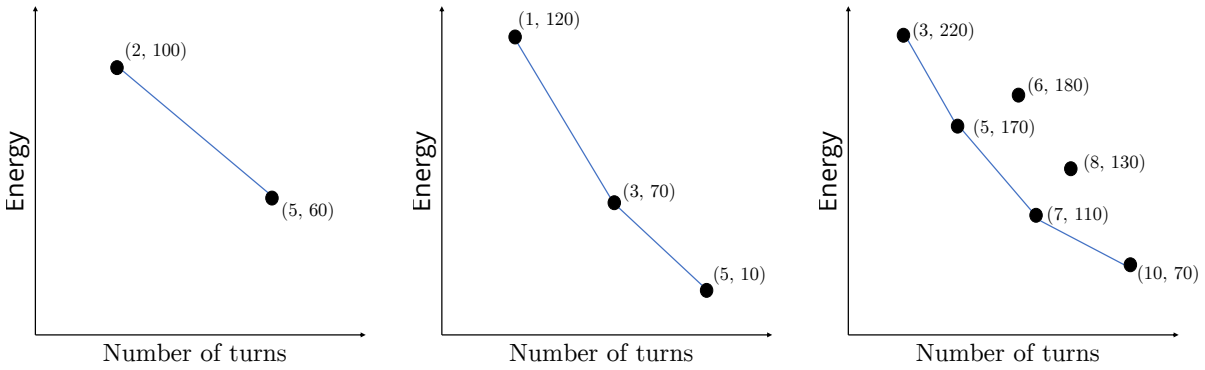
3.2 Brute Force Methods for solving bSTSP and bSTSPTW

This section proposes two brute force methods for solving the MOSTSP—*Graph Concatenation* and *Graph Layering*. Both methods use the line graph $L = (V', E', V'_R)$ as their input.

3.2.1 Graph Concatenation

For a random permutation of $V'_R = (v_1, v_2, \ldots, v_{n_R})$, we first find all optimal bicriteria paths between every pair of consecutive terminals using a bi-objective label correcting algorithm. Let $E_{i,i+1}$ denote a set of all such optimal paths between (v_i, v_{i+1}). The set of optimal paths from v_{n_R} to v_1 be denoted by E_{n_R,n_R+1}. Next, for $i = 1, \ldots, n_R - 1$, we concatenate each path in $E_{i,i+1}$ to paths in $E_{i+1,i+2}$. Let E' be the resulting set of paths. We then take the maximal non-dominated subset of E', denoted by $E_{permute}$. For the assumed permutation, $E_{permute}$ denotes the set of optimal tours. This process is repeated

for all permutations of V'_R . Finally, the solution set E to the MOSTSP is an MNDS of the union of all $E_{permute}$.



(a) Efficiency frontier of paths from v_1 to v_2

(b) Efficiency frontier of paths from v_2 to v_1

(c) Efficiency frontier with optimal tours from v_1–v_2–v_1

Figure 3.1: Graph concatenation illustration

Figure 3.1 illustrates this procedure on a toy example where $n_R = 2$, i.e., there are two terminal nodes v_1 and v_2. Figure 3.1a shows the efficiency frontier from v_1 to v_2. Similarly, Figure 3.1b shows the efficiency frontier from v_2 to v_1. Recall that since our problem is an Asymmetric STSP, optimal paths from v_2 to v_1 may be different from v_1 to v_2. For the complete tour (v_1–v_2–v_1), the efficiency frontier is shown in Figure 3.1c. Note that the method described above requires $n_R!$ number of iterations, and in each permutation, Martin's algorithm is used n_R times.

The pseudocode for the above procedure is given in Algorithm 3. The algorithm takes the line graph as input and outputs the set of optimal paths E for MOSTSP. Line 1 iterates over possible permutations of V'_R. For each permutation, Line 2 adds the origin node at the end of the terminal set. Lines 3–4 find the efficient paths between the terminals for a particular permutation. Lines 5–15 join the efficient paths to form tours that visit all terminals. Line 13 calculates the MNDS of all these tours to find the optimal tours of the MOSTSP for a particular permutation. Lastly, Line 17 computes the MNDS of optimal paths across all permutations.

Algorithm 3 Graph Concatenation (GraphConcat)

Input: $L = (V', E', V'_R)$

Output: E: Optimal paths for MOSTSP

for all $permut = (v_1, v_2 \dots v_{n_R}) \in$ permutation of V'_R **do**
  append v_1 at the end of V'_R
  for v_i in V'_R **do**
    $E_{i,i+1} \leftarrow$ EffPaths(v_i, v_{i+1})
  $E' \leftarrow \{\}$
  for $i \in n_R - 1$ **do**
    for $path \in E_{i,i+1}$ **do**
      if $|E'| = 0$ **then**
        $E' \leftarrow E' \cup \{path\}$
      else
        $E'' = \{\}$
        for $path\_trail \in E'$ **do**
          Append $path$ to the end of $path\_trail$
          $E'' \leftarrow E'' \cup \{path\_trail\}$
        $E' \leftarrow E''$
  $E_{permut} \leftarrow$ MNDS(E')
$E \leftarrow \text{MNDS}\left(\bigcup_{permut} E_{permut}\right)$

3.2.2 Graph Layering

This section presents an alternate approach to compute the efficiency frontier, termed *Graph Layering*. The process starts with a random permutation of V'_R , $(v_1, v_2 \dots v_{n_R})$, and copies the graph $L = (V', E', V'_R)$ into n_R additional layers. A node v_i in $L = (V', E', V'_R)$ is labelled as $v_i^{(1)}$ in the first layer, $v_i^{(2)}$ in the second layer, and so on. For every node $v \in V'_R$, its copies are connected in the consecutive levels by an edge with an attribute $(0, 0)$. We henceforth refer to this structure as a *composite graph*. See Figure 3.2 for reference. The two terminal nodes v_1 and v_2 are indicated in pink. Edges in E' are shown in black. Blue lines are extra edges added to connect different levels.

Using $v_1^{(1)}$ and $v_1^{(n_R+1)}$ as origin and destination, we find the set of optimal paths $E_{permute}$ using bi-objective label correcting algorithm in the *composite graph*. This procedure is repeated for all possible permutations of V'_R . Finally, as before, we calculate the maximal non-dominated subset of the union of all such $E_{permute}$. Algorithm 4 describes the pseudocode for the above procedure. Line 1 starts a loop that runs over all permutations. For each permutation, Line 2 creates n_R additional copies of L. Lines 3–5 find the optimal paths of MOSTSP but for a fixed permutation by finding the efficient paths from the origin of level 1 to the origin of level n_R +1. Lastly, Line 6 finds the MNDS of optimal paths across all permutations.

To see if the application of the graph layering approach is correct, consider the illustration given in Figure 3.2. To reach a node at a particular level, a path must pass through the terminal nodes of previous levels. For example, to reach $v_1^{(2)}$, one must enter level 2, which is possible only via the terminal node $v_1^{(1)}$.

Note: Although the graph layering approach iterates $n_R!$ times, note that Martin's Algorithm is called only once in each permutation. On the other hand, the graph concatenation approach calls Martin's Algorithm n_R times for each permutation.

Algorithm 4 Graph layering (GraphLayer)

Input: $L = (V', E', V'_R)$

Output: E: Optimal paths for MOSTSP

1: **for** all $permut = (v_1, v_2 \dots v_{n_R}) \in$ permutation of V'_R **do**
2: Create n_R additional copies of $L = (V', E', V'_R)$
3: **for** v_i in V'_R **do**
4: Add edge connecting $v_i^{(i)}$ in level i to $v_i^{(i+1)}$ in level $i+1$ with weight $(0, 0)$
5: $E_{permut} \leftarrow$ EffPaths$(v_1^{(1)}, v_1^{(n_R+1)})$
6: $E \leftarrow \text{MNDS}\left(\bigcup_{permut} E_{permut}\right)$

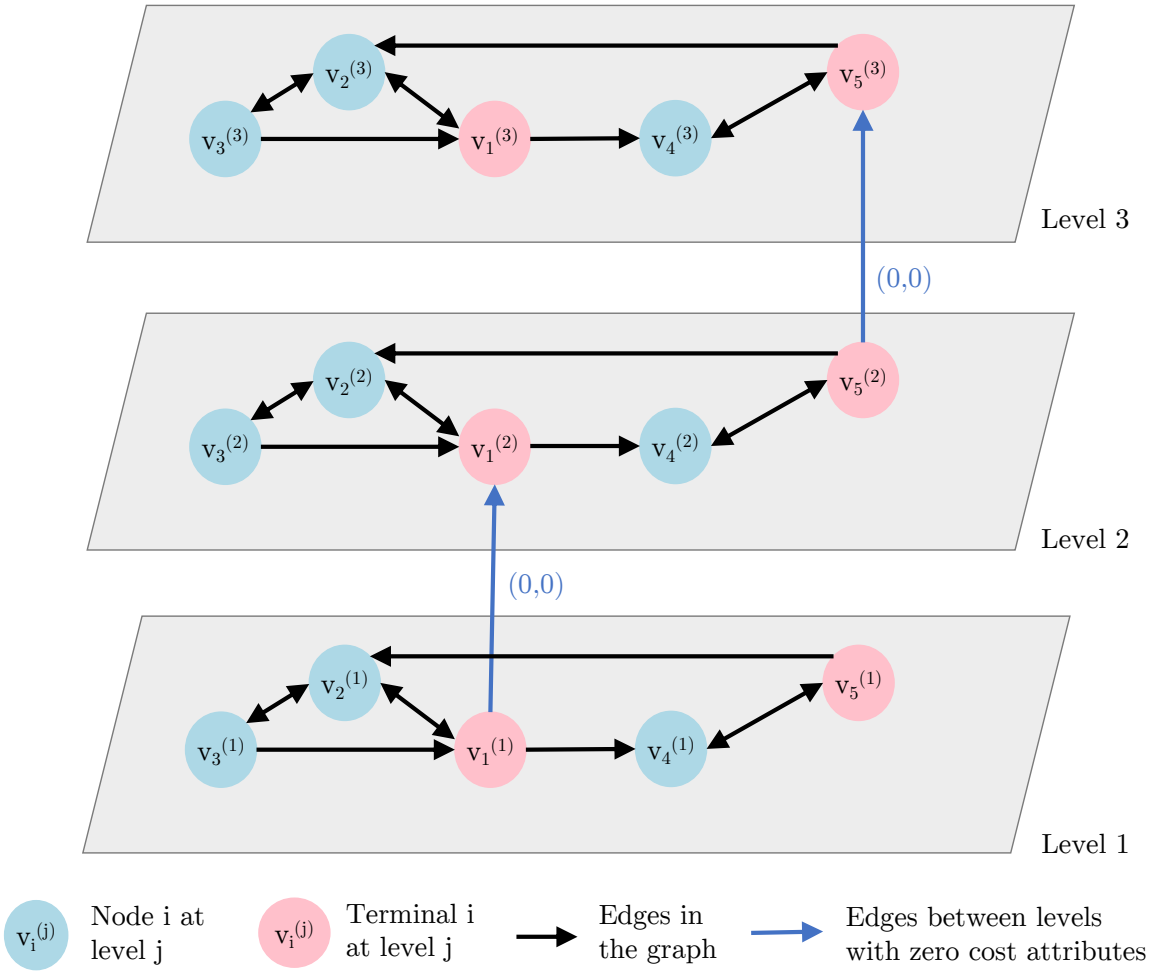


Figure 3.2: Graph layering illustration

Note: The Brute force methods can be trivially extended for bSTSPTW. To incorporate time windows, time window constraints are checked in both graph concatenation and graph layering for each permutation of terminals.

3.3 Limitation of Exact Methods

Both the IP and Brute force methods do not scale well with large graphs. As an edge-based formulation, the IP grows significantly with large graphs. On the other the Brute force method's complexity increases exponentially as it examines all permutations of terminals, resulting in $n_R!$ iterations for each method.

Time-constrained versions of these exact methods can be developed to mitigate these challenges. For example, halting the IP prematurely can provide feasible solutions for both bSTSP and bSTSPTW problems. We will employ a time-constrained IP to benchmark the results of our local search procedure. Further details are provided in Chapter 5. However, it is worth noting that for large graphs, the time-constrained versions of IP may/ mot not yield a feasible solution.

Brute force methods can be halted prematurely after examining a fixed subset of permutations of terminals. However, this approach may not identify high-quality solutions, as Pareto-optimal solutions across all permutations of terminals can significantly vary. Hence, specialized heuristics for bSTSP and bSTSPTW are required. In the subsequent chapter, we will present and discuss a novel local procedure, which is designed to address this issue.

Chapter 4

Heuristic Methods

To achieve great things, two things are needed: a plan and not quite enough time.

Leonard Bernstein

Chapter Overview. This chapter presents a novel heuristic method for solving the bSTSP and bSTSPTW problems. Section 4.1 discusses how to transform an instance of bSTSP into TSP. However, this method fails when time windows are involved. Section 4.2 outlines our local search heuristic devised for addressing the bSTSPTW problem. Specifically, Section 4.2.1 illustrates the technique to generate an initial solution. Next, Section 4.2.2 presents the detail of the neighborhood structure. Lastly, Section 4.2.3 introduces the local search procedure employed in this heuristic approach.

Note: The main contribution of this chapter is the local search heuristic for bSTSPTW

4.1 STSP to TSP Reduction

To convert a single objective STSP instance into a TSP problem, one can calculate the shortest path between each pair of terminals and generate a complete graph. The

terminals are represented by the nodes of the complete graph, and the edges are assigned costs equivalent to the shortest paths between the terminals. By solving the TSP on this complete graph, a solution for the corresponding STSP can be obtained. However, computation of a complete graph can sometimes be expensive (Letchford et al., 2013). As a result, the latest trend in the field is to directly solve the STSP problem by using heuristics and IP formulations. For example, Letchford et al. (2013) proposed a single commodity flow based IP formulation, and Interian and Ribeiro (2017) proposed a Greedy Randomized Adopted Search Procedure (GRASP) based on 2-opt moves for symmetric STSP. The common underlying motivation behind all these methods is that the computation of a complete graph is time-intensive. Álvarez-Miranda and Sinnl (2019) compared the techniques proposed by Letchford et al. (2013) and (Interian and Ribeiro, 2017) to those that utilize a complete graph and then solve using state-of-the-art TSP solvers like CONCORDE (Applegate et al., 2003). The authors concluded that, in most practical cases, the latter approach yields faster results.

For solving the bi-objective variants of STSP, one can repeatedly solve the single objective STSP in conjunction with a scalarization technique (refer Chapter 3). However, this method fails in our case. This is because when the terminals have time windows associated with them, the least energy / turns paths between terminals computed to convert the STSP instance to a TSP instance may not even be feasible with respect to the time windows.

Therefore, novel heuristics are necessary for solving bSTSPTW. The following section proposes our new local search heuristic for the same.

4.2 Local Search Procedure for bSTSPTW

This section proposes a new local search-based heuristic method for the bSTSPTW.

To start with, we maintain a non-dominated set $\hat{E}$ that approximates the time window satisfied Pareto-optimal tours. The set $\hat{E}$ is updated using the procedure AddSolution (refer Lust and Teghem (2010b) for more details). AddSolution checks if a tour can be added to $\hat{E}$ (a new tour can be added to $\hat{E}$ only if it is non-dominated with respect to $\hat{E}$). If yes, existing tours dominated by the newly added tour (if any) are removed. That

is, AddSolution ensures that $\hat{E}$ is always a non-dominated set. The pseudocode for the procedure is given in Algorithm 5. The algorithm takes the set $\hat{E}$ and a tour p as input and returns the updated set $\hat{E}$ and a Boolean variable *Efficient* indicating if p was added to $\hat{E}$. Line 1 initializes a Boolean variable *Efficient* as True. For every tour p_{old} in $\hat{E}$, Lines 2–5 check if p_{old} is dominated by p. If that is true, *Efficient* is set to False, and the loop terminates. Else Lines 6–7 remove all tours that are dominated by p. Finally, Line 8–9 adds the new tour p to $\hat{E}$ if *Efficient* is true.

Further, the local search procedure also maintains a set $\hat{X}$ of "promising" tours which do not satisfy time windows at all terminals but are expected to satisfy time windows in all terminals after a few iterations of the local search.

Algorithm 5 AddSolution

Input: $\hat{E}$, p

Output: $\hat{E} : \text{MNDS}(\hat{E} \cup p), Efficient$

1: $Efficient \leftarrow True$
2: **for** $p_{old} \in \hat{E}$ **do**
3: **if** $z(p_{old}) \preceq z(p)$ **then**
4: $Efficient \leftarrow False$
5: **break**
6: **else if** $z(p) \prec z(p_{old})$ **then**
7: $\hat{E} \leftarrow \hat{E} \setminus \{p_{old}\}$
8: **if** $Efficient = True$ **then**
9: $\hat{E} \leftarrow \hat{E} \cup \{p\}$

The subsequent sections describe our local search procedure in detail.

4.2.1 Initial Solution

The initial solution to the bSTSPTW is obtained by solving a time window relaxed version by converting it into an TSP instance using the procedure detailed in Section 4.1. Out of all the tours obtained from the scalarization technique on TSP, the ones that satisfy time windows are added to $\hat{E}$. For the tours that are not time windows feasible,

we define a penalty function. The penalty at a terminal node v is defined as follows:

$$penalty(v) = \begin{cases} R_v - s & \text{if } s < R_v \\ s - D_v & \text{if } s > D_v \\ 0 & \text{if } R_v \leq s \leq D_v \end{cases} \tag{4.1}$$

where, s is the time to reach terminal v from origin, and the R_v, D_v denote minimum, maximum service time at node v. The total penalty of a path is the sum of the penalties over all terminals.

$$path\_penalty(p) = \sum_{v \in V_R} penalty(v) \tag{4.2}$$

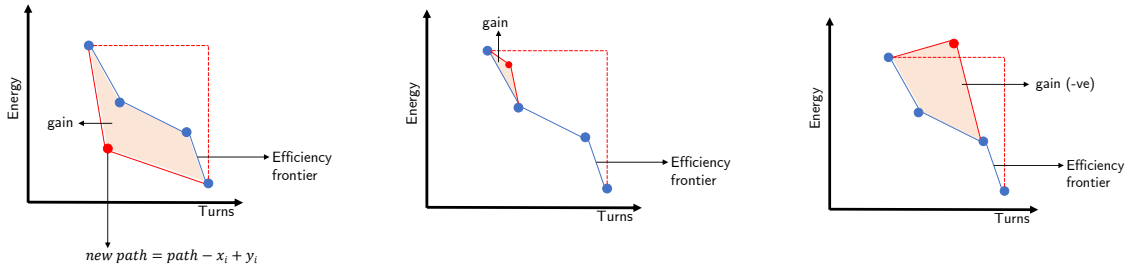
Similar to Helsgaun (2017), our use local search procedure uses penalties to deal with time window constraints. Tours that are not time window feasible, i.e., $path\_penalty(p) > 0$, are added to the $\hat{X}$ set.

4.2.2 Neighborhood Structures

Building on the sequential move neighborhood for symmetric TSP by Lin and Kernighan (1973); Helsgaun (2000) and asymmetric TSP by Kanellakis and Papadimitriou (1980), we propose three neighborhood structures: I3OPTTW, I3OPT, QUAD. The I3OPT and QUAD neighborhood structures replace sections of a tour with shortest or efficient paths with the goal of intensification and diversification, respectively. Note that, unlike previous studies, we limit ourselves to 3-opt and quad moves. The I3OPTTW move attempts to repair neighborhood by transforming time window infeasible tours into feasible ones. Lastly, for a fixed permutation of terminals, the FIXEDPERMNBD move aims to optimize the path between terminals.

A i3optTW and i3opt

To explain the I3OPTTW and I3OPT neighborhood structures, we first define a GAIN function. In a set of non-dominated tours, the GAIN quantifies how "good" a new path is compared to the existing ones. Thus, a higher value of gain means the new path has more



(a) Positive gain **(b)** Positive gain **(c)** Negative gain

Figure 4.1: GAIN function illustration

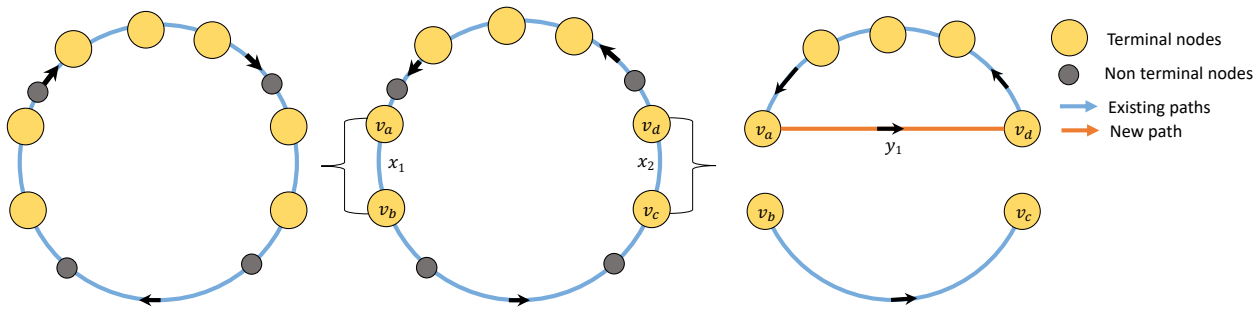
potential to be added into $\hat{E}$. Mathematically, the GAIN function computes the area of the polygon on the objective space formed by joining the efficiency frontier to the image of the path on the objective space. See Figure 4.1 for illustration. The new tour being added is shown in red, and the blue points denote the existing non-dominated tours. The shaded area indicates the gain. If the new path is non-dominated (dominated) by the efficient set, the gain is positive (negative). Figures 4.1a and 4.1b, show the example of positive gain, and Figure 4.1c illustrates negative gain.

Note: From hereon, the gain for a tour p with respect to the efficient set is $\hat{E}$, will be denoted by GAIN (p)

Next, we explain the I3OPTTW and I3OPT move. For a given tour, both I3OPTTW and I3OPT follow a similar procedure, with a few key differences highlighted below. Both moves operate in two steps.

Step 1: In the first step, we find a pair of terminals $x_1 = (v_a, v_b)$ and $y_1 = (v_a, v_d)$ that satisfy the conditions given below. Note that starting node v_a is common in both x_1 and y_1. The section of the path between y_1 is obtained by joining v_a to v_d using the shortest path where the cost matrix is a weighted sum of the attributes, energy, and turns. For illustration, refer to Figure 4.2.

- $x_1 = (v_a, v_b)$ should an adjacent terminal pair. An adjacent terminal pair means no other terminal node exists in-between $x_i = (v_l, v_m)$ (non-terminal nodes are allowed). Ensuring adjacency in x_i is necessary since it will be removed at later stages. If the terminals are adjacent, it guarantees that no terminal is lost in the



(a) Initial Tour **(b)** x_1 and x_2 are selected **(c)** y_1 is selected

Figure 4.2: I3OPT and I3OPTTW step 1 illustration

new_tour after applying the neighborhood move, thereby rendering the *new_tour* feasible for bSTSP.

- There should be at least two terminals between v_d to v_a
- Time to reach terminal v_d using path y_1 must satisfy time window at v_d

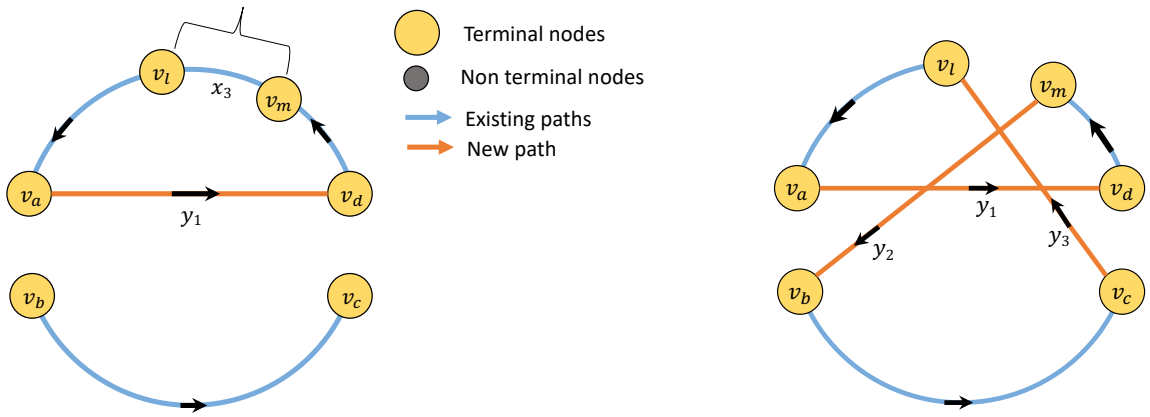
For I3OPTTW move, in addition to the above rules, x_1 is set to be the tour segment that extends from the terminal immediately preceding the violation of the time window constraint to the terminal where the violation occurs.

For each $x_1 = (v_a, v_b)$ and $y_1 = (v_a, v_d)$, $x_2 = (v_c, v_d)$ is automatically defined uniquely as v_d is fixed by y_1 while v_c is the terminal just before v_d. This makes $x_2 = (v_c, v_d)$ an adjacent terminal pair. The tuple (x_1, x_2, y_1) is added to the set of INITIALCANDIDATES.

However, in the case of I3OPT, the set INITIALCANDIDATES is large. To overcome this, a *pre-selection* is applied to filter the set INITIALCANDIDATES. To do so, the mean energy consumption and mean turn for all graph edges of the graph are computed. If the energy consumption / turns in $x_1 \leq filter * mean * |x_1|$ then all such tuples (x_1, x_2, y_1) are removed from INITIALCANDIDATES. (Here, $|x_1|$ refers to a number of edges in x_1, the filter is scaler). The *pre-selection* criteria help in identifying the adjacent pair of terminal x_1, removal of which will likely improve the tour.

Finally, the gain of path formed by removing x_1 and adding y_1 is maximized among all $(x_1, x_2, y_1) \in$ INITIALCANDIDATES. This concludes the first step.

Note: Step 1 creates a cycle and sub-path. Following this procedure, the path from origin to v_d satisfies the time window.



(a) x_3 is selected

(b) y_2 and y_3 are selected

Figure 4.3: I3OPT and I3OPTTW step 2 illustration

Step 2: The second step involves breaking the cycle $v_a \rightarrow v_d \rightarrow v_a$. To accomplish this, an adjacent terminal pair is selected in the section $v_d \rightarrow v_a$. Step 1 ensures the existence of at least one such adjacent terminal pair $x_3 = (v_m, v_n)$. Consider the example in Figure 4.3. v_m is connected to v_b using the shortest path y_2, where the cost matrix is a weighted sum of energy and turns. Similarly, v_c is connected to v_l. The process is repeated for all possible (x_3, y_2, y_3) combinations. This generates a set of new feasible tours for bSTSP. The gain for this new tour is GAIN$(tour - x_1 - x_2 - x_3 + y_1 + y_2 + y_3)$, where *tour* is the initial tour on which the neighborhood is applied. The gain and penalty of each tour are computed, and the "best" tour is selected. The best tour is defined using the following relation.

$$P(best\_tour) \leq P(p) \wedge \Big(P(best\_tour) = P_p \wedge \text{GAIN}(best\_tour) \geq \text{GAIN}(p)\Big) \quad \forall\ p \quad (4.3)$$

where $P(p)$ and $gain(p)$ refer to the penalty of gain of a tour p.

Note: Selecting *best_tour* using the criterion given above gives more importance to reducing penalty than improving gain

Algorithm 6 provides the pseudocode for the I3OPT and I3OPTTW neighborhood. Line 1 initializes *tour* with a random tour from $\hat{E}$. Line 2–3 finds the tuple (x_1, x_2, y_1) that maximises the gain for all feasible $(x_{1i}, x_{2i}, y_{1i}) \in$ INITIALCANDIDATES. Finally, BREAKCYCLE in Line 4 performs the operations in Step 2 and returns the best tour along with its penalty and gain.

Algorithm 6 I3OPT and I3OPTTW

Input: Z, $filter$ ▷ $Z = \hat{E}$ for I3OPT and $Z = \hat{X}$ for I3OPTTW

Output: $new\_path, penalty, gain$

1: $tour =$ RANDOM(Z) ▷ If $Z = \hat{E}$, the *tour* is picked based on KDE_RANDOM

2: $x_1, y_1 \leftarrow \arg\max(\text{GAIN}(tour - x_{1i} + y_{1i}))\ \forall\ (x_{1i}, y_{1i}, x_{2i}) \in$ INITIALCANDIDATES

3: x_2 is fixed as described in the text

4: $best\_tour, penalty, gain \leftarrow$ BREAKCYCLE$(tour, x_1, x_2, y_1)$ ▷ As described in text

Note: The *best_tour* obtained at the end of performing a I3OPT or I3OPTTW move might not be time window feasible but is always a feasible tour for bSTSP

The I3OPT and I3OPTTW starts from a *tour* and outputs a *new_tour*. In the case of I3OPTTW, the tour is selected randomly from the set $\hat{X}$, which contains "promising" tours that do not satisfy time windows. However for I3OPT, the *tour* is selected using KDE_RANDOM (illustrated in Algorithm 7). The algorithm KDE_RANDOM allows one to pick a *tour* from a relatively less explored portion of the objective space. KDE_RANDOM works as follows: Line 1 computes the Gaussian Kernel Density Estimation for the given data. Line 2 evaluates the probability density at each data point. The density values are inverted and squared to assign higher probabilities to data points with low density in Line 3. This corresponds to less explored regions in the objective space. Line 4 normalizes the probabilities to sum to one. Finally, a random index is chosen from the data based on the probabilities in Line 5. Thus, when KDE_RANDOM is called with turns or energy cost of the tours in $\hat{E}$, it would pick a *tour* from a relatively less explored region from the objective space.

Algorithm 7 KDE_RANDOM

Input: *data*

Output: *selected_index*

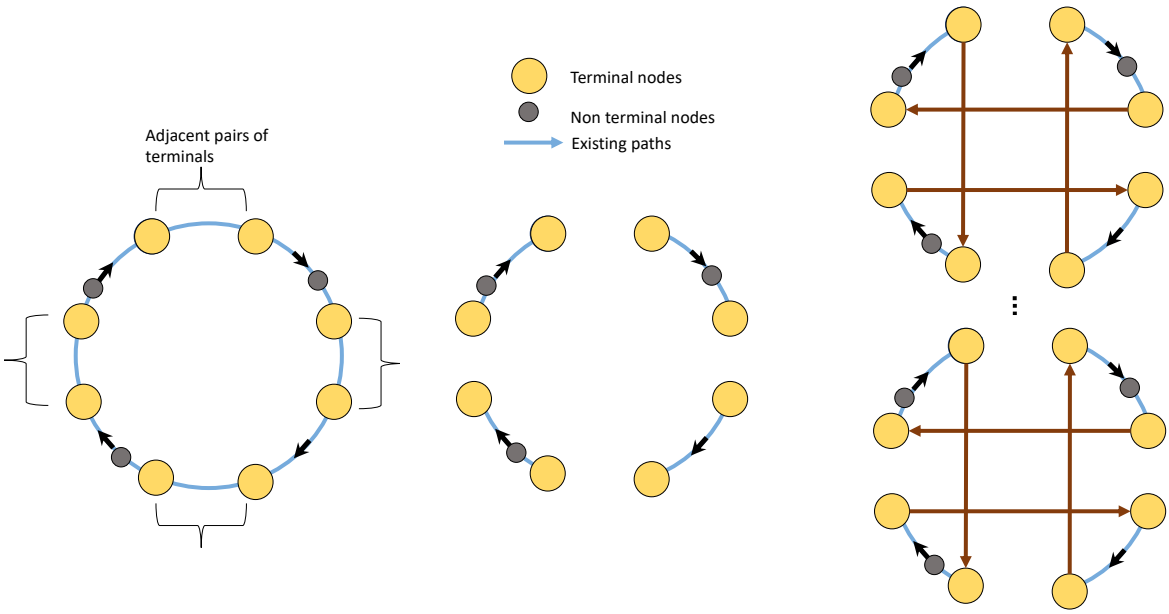
1: $kde \leftarrow$ GAUSSIAN_KDE($data$)
2: $density \leftarrow kde$.evaluate($data$)
3: $inverse\_density \leftarrow (1/density) * (1/density)$
4: $probabilities \leftarrow inverse\_density/\ (\sum inverse\_density)$
5: $selected\_index \leftarrow$ pick a random number in range $[0, |data|]$ based on the probabilities

B Quad move

The QUAD MOVE is a diversification neighborhood aimed at introducing randomness into the local search to avoid getting trapped in local optima. QUAD MOVE randomly selects four adjacent terminal pairs and rearranges them to create a new bSTSP-feasible tour. Figure 4.4 explains the workings of the QUAD MOVE. Subfigure 4.4a presents the initial tour and the selected adjacent terminal pairs, and Subfigure 4.4b demonstrates the breaking of sub-paths. To join the sub-paths, new paths are obtained using a bi-objective label-correcting algorithm, which provides a set of Pareto-optimal paths between a pair of nodes. These paths are then combined to create a new tour. Subfigure 4.4c shows the final output, which is the MNDS from the resulting tours. Since the QUAD MOVE yields multiple tour outputs, it enriches the search process. Algorithm 8 explains the working of the QUAD MOVE neighborhood. Line 1 picks a random tour from $\hat{E}$. Next, Line 2 finds all pairs of adjacent terminals. Line 3–4 shuffles and randomly picks four unique pairs. Line 5–8 performs the quad move on the four selected pairs of terminals. Line 9 joins all the paths to form resulting tours.

C FixedPermNbd

For a given permutation of the terminals, the FIXEDPERMNBD aims to improve paths between terminals. To do so, the graph concatenation approach (described in Chapter 3) is used to get the best set of tours for that permutation. Note that since FIXEDPERMNBD does not change the order of terminal visits, the time windows are not disturbed. However,



(a) Initial tour **(b)** Paths broken **(c)** New tour set

Figure 4.4: QUAD MOVE ILLUSTRATION

Algorithm 8 QUAD_MOVE

Input: $\hat{E}$

Output: *new_path_set*

1: $tour = \text{RANDOM}(\hat{E})$
2: $pairs \leftarrow$ Generate all pairs of adjacent terminals
3: Shuffle $pairs$ randomly
4: Pick the of 4 unique pairs in $tour$ $used[0]$ to $used[7]$ ▷ $used[0], used[1]$ refer to first of adjacent terminal pair
5: **for** $i \in \{1, 3, 5, 7\}$ **do**
6: $\quad p_i \leftarrow$ subpath of $tour$ from index $used[i]$ to $used[i+1]$
7: **for** $i \in \{0, 2, 4, 6\}$ **do**
8: $\quad p_{i\_set} \leftarrow$ bi-objective label correcting paths from $used[i]$ to $used[i+1]$
9: $quad\_paths \leftarrow [p_0 + p_1 + p_2 + p_3 + p_4 + p_5 + p_6 + p_7]$ for p_0 in p_{0_set} for p_2 in p_{2_set} for p_4 in p_{4_set} for p_6 in p_{6_set}]

computing all the bi-objective shortest paths and joining them is computationally expensive, especially when the number of paths is exponential. Therefore, in FixedPermNbd, we only find the bi-objective shortest paths between a subset of adjacent terminal pairs. For all other pairs, shortest paths with cost matrix as a convex combination of energy and turns paths are used.

To start with, the FixedPermNbd move is applied in instances where a considerable skew (either in low energy or low turns) is observed along the efficiency frontier. The tour responsible for the skew is identified, and its neighboring region is investigated while maintaining a fixed permutation of terminals. A tour is considered skewed in cost (energy or turns) if its total energy consumption or the number of turns deviates significantly from the mean energy consumption or turn count of all other tours in the $\hat{E}$ set.

Algorithm 9 illustrates the pseudocode for FixedPermNbd. In Line 1, the function CheckSkewness identifies the skewed tours (w.r.t. energy or turns). Line 2 randomly selects one tour from the outliers. Next, in Line 3, a subset of adjacent terminal pairs is chosen from the tour based on a predetermined probability p. These adjacent terminals are joined using bi-objective shortest paths in Line 4. Subsequently, *new_paths* are constructed in Line 5 by joining the remaining terminals using the shortest paths with cost matrix as a convex combination of energy and turns.

Algorithm 9 FixedPermNbd

Input: $\hat{E}$

Output: *new_path*

1: *outliers* ← CheckSkewness($\hat{E}$)
2: *skew_path* ← Random(*outliers*)
3: Select adjacent pair of terminals from *skew_path* based on a probability p
4: For the selected terminals, change the subpaths using a bi-objective label correcting the shortest path algorithm
5: *new_paths* ← Complete the tour by connecting the remaining terminals with both the least energy/turns path

4.2.3 Local Search Procedure

Algorithm 11 illustrates the pseudocode for the local search procedure. Input to the algorithm is the maximum allowed runtime *max_time*, Line 2 initializes the two sets, $\hat{E}$, and $\hat{X}$, as explained in Section 4.2.1. Line 3 sets the initial values for penalties to infinity. To facilitate the early removal of inferior x_1 subpaths in the I3OPT neighborhood, Line 6 reduces the scaler $filter$ from δ to 1. Subsequently, Line 8 applies the I3OPT neighborhood on $\hat{E}$ tours. If the *new_tour* obtained after applying I3OPT is Pareto-optimal w.r.t. $\hat{E}$, Line 10 calls UPDATESETS and updates the set $\hat{E}$ and $\hat{X}$ depending upon the penalty value.

Next, the I3OPTTW neighborhood is applied to the tours in $\hat{X}$. If the *new_tour* obtained after applying I3OPTTW is Pareto-optimal to $\hat{E}$, and with zero penalty, it is added to $\hat{E}$ and variable *count* is set as 0 using UPDATESETS. In case the *new_tour* has a non-zero penalty, it is added to $\hat{E}$ using UPDATESETS. Note that the set $\hat{X}$ is maintained as a sorted list in the increasing order of penalty, and 10 tours with the least penalty are kept in $\hat{X}$. UPDATESETS also updates the *best_penalty* of *new_path* reduces penalty. If skewness is detected in $\hat{E}$, Lines 14–17 perform the FIXEDPERMNBD neighborhood move. If there are no updates to $\hat{E}$ for ρ consecutive iterations, a QUAD_MOVE is executed. This marks the completion of a single local search iteration.

Algorithm 10 UPDATE_SETS

Input: $new\_tour, \hat{E}, \hat{X}, best\_penalty,$ optional *count*

Output: $\hat{E}, \hat{X}, best\_penalty,$ optional *count*

1: **if** $penalty = 0$ **then**
2: $\quad \hat{E}, Added \leftarrow$ ADDSOLUSTION$(new_tour, \hat{E})$
3: **if** $penalty \leq best\_penalty$ **then**
4: $\quad$ Add *new_tour* to $\hat{X}$
5: **else**
6: $\quad count \leftarrow 0$
7: Update $\hat{E}$ to keep top 10 tours with least *penalty*

Algorithm 11 LOCAL_SEARCH

Input: max_time

Output: $\hat{E}$

1: $count \leftarrow 0$
2: $\hat{E}, \hat{X} \leftarrow$INITIALPOPULATION( )
3: $best\_penalty_E, best\_penalty_X, best\_penalty_F \leftarrow \infty, \infty, \infty$
4: **while** $time\_spent \leq max\_time$ **do**
5: $count \leftarrow count + 1$
6: $filter \leftarrow \delta \times \frac{\text{time\_spent}}{\text{max\_time}}$ ▷ reduces from δ to 1
7: If any of the $best\_penalty$ becomes 0, set that at ∞
8: $new\_tour, penalty, gain \leftarrow$ I3OPT$(\hat{E})$
9: $params = new\_tour, penalty, \hat{E}, \hat{X}, best\_penalty_E, count$
10: $\hat{E}, \hat{X}, best\_penalty_E, count \leftarrow$ UPDATE_SETS$(params)$
11: $new\_tour, penalty, gain \leftarrow$ I3OPTTW$(\hat{E})$
12: $params = new\_tour, penalty, \hat{E}, \hat{X}, best\_penalty_X, count$
13: $\hat{E}, \hat{X}, best\_penalty_X, count \leftarrow$ UPDATE_SETS$(params)$
14: $new\_paths\_set \leftarrow$ FIXEDPERMNBD$(\hat{E})$
15: **for** each new_tour in new_paths_set **do**
16: $params = new\_tour, penalty, \hat{E}, \hat{X}, best\_penalty_F$
17: $\hat{E}, \hat{X}, best\_penalty_F \leftarrow$ UPDATE_SETS$(params)$
18: **if** $count \geq \rho$ **then**
19: $count \leftarrow 0$
20: $new\_paths\_set \leftarrow$ QUADMOVE$(\hat{E})$
21: **for** each new_tour in new_paths_set **do**
22: **if** $penalty = 0$ **then**
23: $\hat{E}, Added \leftarrow$ ADDSOLUSTION$(new_tour, \hat{E})$

Chapter 5

Computational Results

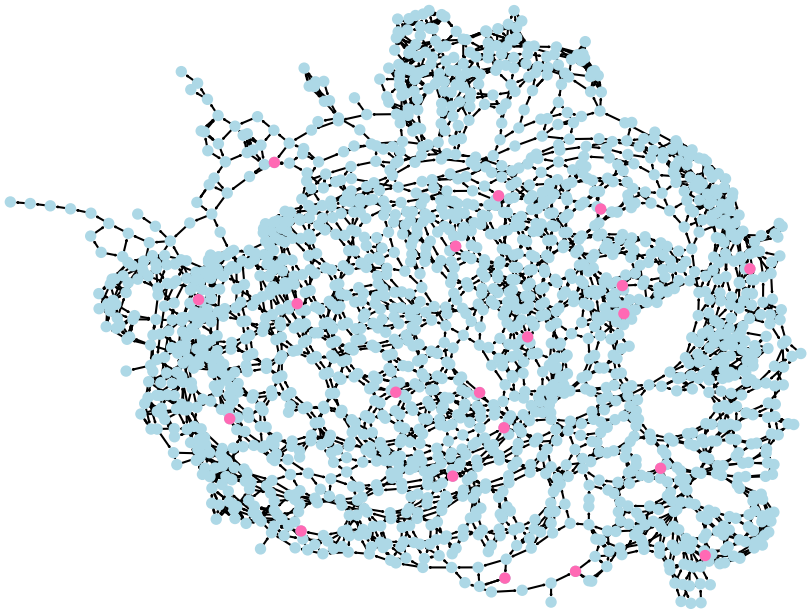
Excellence is a continuous process and not an accident.

A. P. J. Abdul Kalam

Chapter Overview. This chapter presents the experimental setup followed by a comparison of the various exact approaches in Section 5.1 and heuristic approaches in Section 5.2.

All algorithms were implemented in Python 3 and compiled using a 128-core Intel Xeon Gold CPU clocked at 3.0 GHz with 512 GB RAM. The optimization model was solved using IBM's CPLEX. We use two datasets: Amazon Last Mile Routing Research Challenge Dataset and the road network from Bengaluru, India. The network details for the chosen test networks from the Amazon Last Mile Routing Research Challenge Dataset are summarised in Table 5.3. For more details, refer Chapter 1 and Merchan et al. (2022). To access the scalability of the proposed approaches and test their viability in the Indian context, we also experiment using the Bengaluru road network obtained from OpenStreetMaps. Results are presented w.r.t graphs of varying sizes summarized in Table 5.1. Figure 5.1a shows one such graph for a road network of a radius of 1 km and 20 terminals. The corresponding line graph is shown in Subfigure 5.1b. Note that the terminal locations were randomly chosen for the Bengaluru network.

(a) Original road network



(b) Corresponding line graph

Figure 5.1: Test network of radius 1 km. Terminal edges & nodes are indicated in pink.

Radius (km)	Original Graph		Line Graph	
	# of Nodes	# of Edges	# of Nodes	# of Edges
0.5	239	644	644	1 904
1	745	1 944	1 944	5 602
3	6 483	17 724	17 724	53 409
4	13 177	36 904	36 904	113 472

Table 5.1: Details of various graphs used to generate results for exact methods (*Radius*: Radius of the test Network in km, *# of Nodes*: Number of nodes, *# of Edges*: Number of edges)

5.1 bSTSP: Exact approaches performance

Table 5.2 illustrates the time taken (in seconds) by the exact methods discussed in Chapter 3 for the Bengaluru road network. Terminal locations are chosen randomly and vary in the range of $1-6$. The maximum runtime allowed was 2.5 hours. For comparison, we also include the runtime IP method taken to find the exact solutions. To get exact solutions using the IP, all input parameters related to time in Algorithm 1 and Algorithm 2 are initialized to ∞. Results indicate that for all problem instances, GraphConcat and GraphLayer were much faster than IPSolver. Following additional trends can be observed.

Recall that GraphConcat uses Martin's Algorithm n_R times on the line graph, whereas Graph Layer uses bi-objective shortest path algorithm once on the composite graph, which is n_R times larger in size than the line graph. Our empirical results show that GraphConcat performs better than GraphLayer, suggesting that the complexity of Martin's algorithm increases significantly with the graph size. As expected, all three exact approaches do not scale well. The computational time increases exponentially with the number of terminals (see Figure 5.2). This is why the algorithms were not tested on the Amazon Dataset.

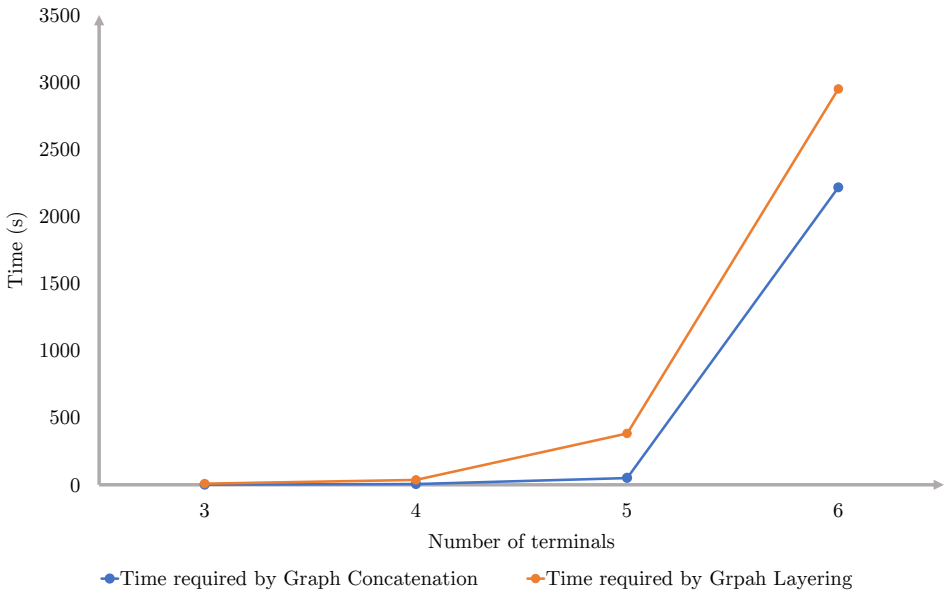


Figure 5.2: Comparison between Graph Concatenation and Graph Layering for different terminals on a road network of 0.5 km radius.

Radius (km)	Terminals	Time required by GraphConcat	Time required by GraphLayer	Time required by IPSolver
0.5	3	1.3	8.3	652.4
	4	5.1	36.0	2 459.9
	5	50.3	381.9	8 380.5
	6	2 214.7	2 948.1	–
1	3	5.4	33.7	99.4
	4	36.5	481.5	3 012.2
	5	185.4	1 4705.0	–
	6	4 4757.6	–	–
3	3	509.9	5 4280.2	–
	4	4 4324.3	–	–
	5	–	–	–
	6	–	–	–

Table 5.2: Time (in seconds) for Exact Methods. Blank fields indicate that a solution was not found under the maximum time limit (2.5 hours). (*Radius*: Radius of the test network, *Terminals*: Number of terminals.)

Route No.	Original Graph		Line Graph		n_R	# TW
	$\|V\|$	$\|E\|$	$\|V'\|$	$\|E'\|$		
18	10 116	24 139	24 139	66 000	110	7
896	7 408	17 593	17 593	47 850	73	6
1643	28 540	70 400	70 400	198 376	87	1
3436	28 531	70 381	70 381	198 323	90	4
4236	22 681	55 913	55 913	157 406	92	6
4260	29 487	72 619	72 619	204 429	143	2
4494	19 353	48 034	48 034	136 065	55	9

Table 5.3: Details of various graphs used to generate results for heuristic methods (*Route No.*: Entry number in the Amazon Dataset, *# TW*: Number of terminals with time windows)

5.2 bSTSPTW: Heuristic approach performance

We benchmark the performance of local search against time-constrained IP. All experiments below are w.r.t the Amazon Dataset. The quality of Pareto-optimal solutions obtained is compared based on two performance indicators described below.

- **Hypervolume (HV)** (Zitzler et al., 2003): Hypervolume finds the area in the objective space that is dominated by the solution set with respect to a reference point. Since evaluating the full efficiency frontier for large graphs is computationally infeasible, comparing HV values makes more sense. Solution quality can be judged directly from its HV value (the higher the HV, the better the quality). In case there are two nadir points (x_1, y_1) and (x_2, y_2) (corresponding to the two efficiency frontiers by IP and local search), the reference point is worse in each objective, i.e., $\max(x_1, x_2)$, $\max(y_1, y_2)$ (A nadir point is defined as the vector for which each component has its maximum value in the points of the non-dominated front. For more details, refer (Audet et al., 2021)). This assures that both solutions always dominate the reference point. Figure 5.3 illustrates the implementation of hypervolume in our case. The figure shows two efficiency frontier and their corresponding nadir points (in blue). Here, the reference point is the nadir point

of the first efficiency frontier. The hypervolume of each efficiency frontier will be as shown. Note that the exact implementation of HV varies in the literature. For the present case, we use a metric similar to the one implemented in pymoo (Blank and Deb, 2020), in which the Hypervolume value is normalized by the area of the rectangle formed with two opposite vertices, one at the nadir point and the other corresponding to a minimum of both objectives ("rectangle area"). Figure 5.4 illustrates this for one of the test networks.

- **Cardinality Metric (CM)** (Riquelme et al., 2015): The CM value of a method is the number of solutions provided by it. A higher value of CM indicates that the solution set will be larger and is, thus, preferred as it provides a more diversified solution set.

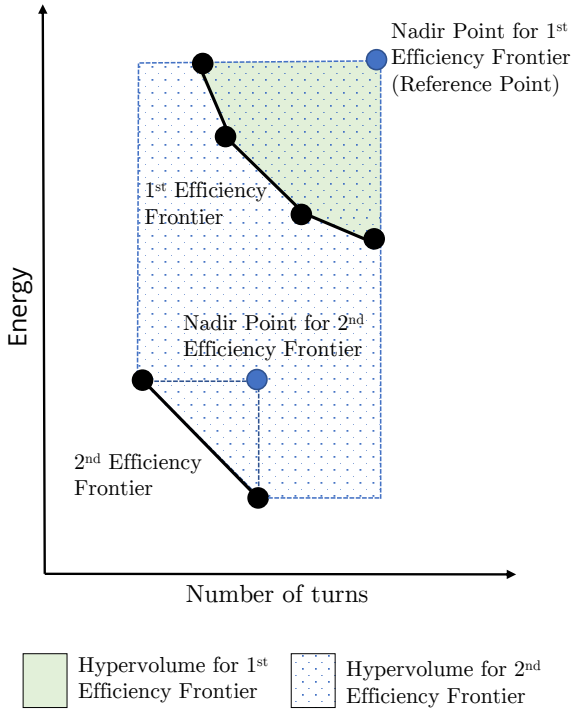


Figure 5.3: Hypervolume indicator and nadir point

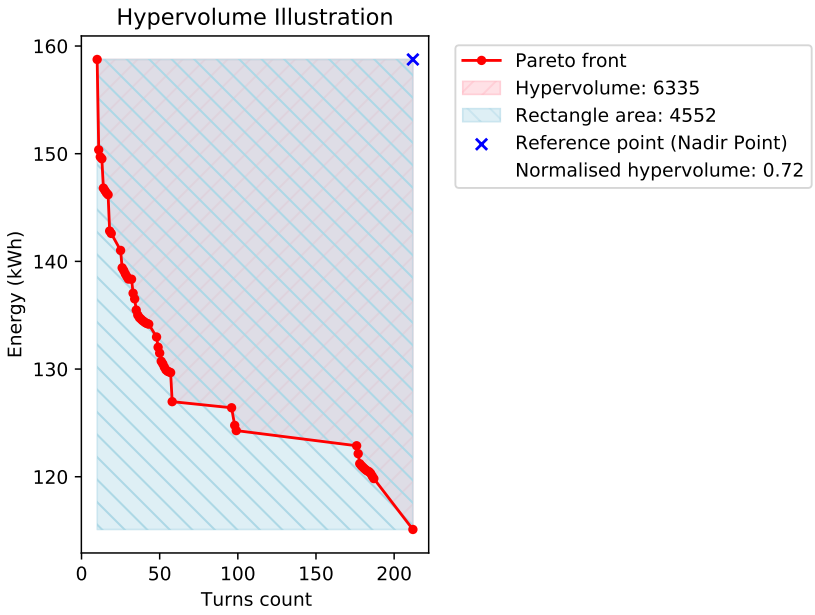


Figure 5.4: Hypervolume and normalised hypervolume

The experimental setup for IP is described as follows. To make a fair comparison, both approaches are run for $\tau = 1$ hour. Recall that the local search works in two stages. First, optimal bSTSP tours are obtained using the scalarization technique. Of all these solutions, the optimal bSTSPTW tours are filtered. On these solution, the neighborhoods are applied. For local search, the time for initial solution generation was set to one-fourth of the total time budget τ. For IP, τ only measures the time for finding the set optimal edges (i.e., post-processing time is not included). The desired number of tours k in IP is set to 2 in order to maximize the time to solve each IP. It is worth noting with a fixed total time budget, increasing k would reduce the time given to solve each IP during scalarization. For a particular instance of α and β, the IP is run for $\frac{\tau}{k}$ time. It should be noted that the final number of tours in the IP output is often less than k because iterations of MidPaths would not generate a new point every time. The results of the local search and time-constrained version of IP for bSTSPTW are summarized in Table 5.5.

Further, to examine the effectiveness of the heuristic technique, the local search method was allowed to run for shorter periods of time — 15 minutes and 30 minutes. The reference point for the hypervolume in these cases was taken to be the same as that of one hour.

By doing so, we are able to quantify how good solutions were obtained in smaller time limits. The results are summarized in Table 5.6 and 5.7, respectively. Figure 5.6 shows the results after 15 minutes, Subfigure 5.6a shows the initial solution using scalarization and the tours that satisfy time windows, Subfigure 5.6b shows the results of the local search that started with the initial solutions in Subfigure 5.6a. Subfigure 5.6c shows both plots together. Similarly, Figure 5.7 and Figure 5.8 show the efficiency frontier after 30 min and 1 hour respectively. Finally, Table 5.4 compares the hypervolume improvement over time. Figure 5.5 visualizes the improvement of hypervolume over time.

Route No.	Hypervolume		
	15 min	30 min	60 min
18	0.32	0.78	0.89
896	0.79	0.8	0.8
1643	0.65	0.86	0.93
3436	0.25	0.96	0.96
4236	0.26	0.88	0.96
4260	0.49	0.78	0.87
4494	0.71	0.75	0.76

Table 5.4: Local search hypervolume at different times

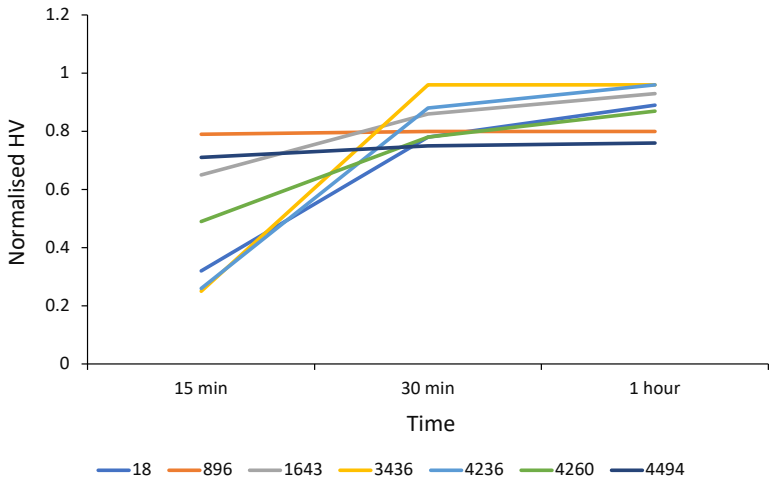


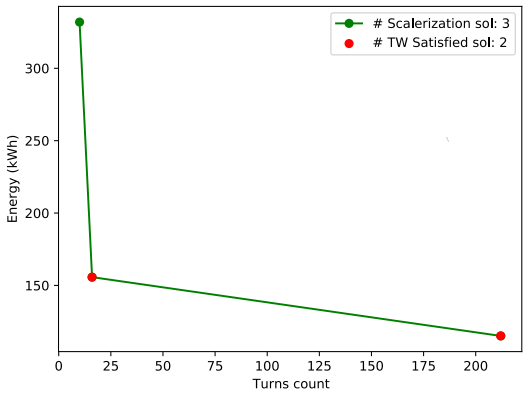
Figure 5.5: Hypervolume at different times of local search

Route No.	Initial solution using scalerization			Local search		IP	
	CM (bSTSP)	CM (bSTSPTW)	HV	CM	HV	CM	HV
18	7	5	0.83	33	0.91	–	
896	5	4	0.79	18	0.8	–	
1643	3	3	0.8	41	0.93	–	
3436	5	2	0.82	42	0.96	–	
4236	4	2	0.87	21	0.88	–	
4260	4	3	0.38	34	0.87	–	
4494	7	6	0.67	55	0.76	–	

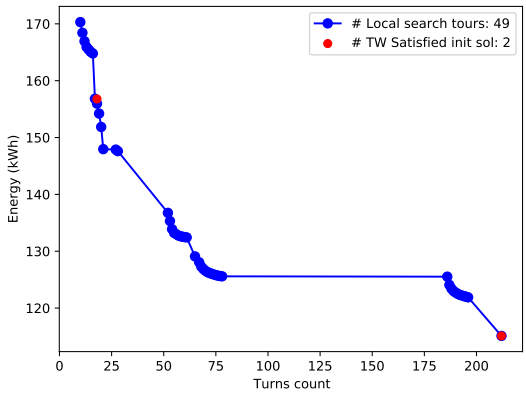
Table 5.5: Local search vs. IP solutions to bSTSPTW after 1 hour (blank field indicates no solutions could be obtained in 1 hour.)

Route No.	Initial solution using scalerization			Local search	
	CM (bSTSP)	CM (bSTSPTW)	HV	CM	HV
18	3	2	0.25	32	0.32
896	5	4	0.79	20	0.79
1643	2	2	0	23	0.65
3436	2	1	NA	21	0.25
4236	2	2	0	11	0.26
4260	2	2	0	17	0.49
4494	3	2	0.24	49	0.71

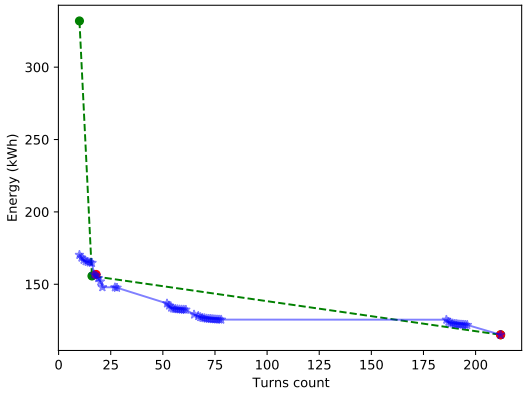
Table 5.6: Local search solutions to bSTSPTW after 15 min



(a) Scalerization results after 4 minutes



(b) Local search results after 15 minutes

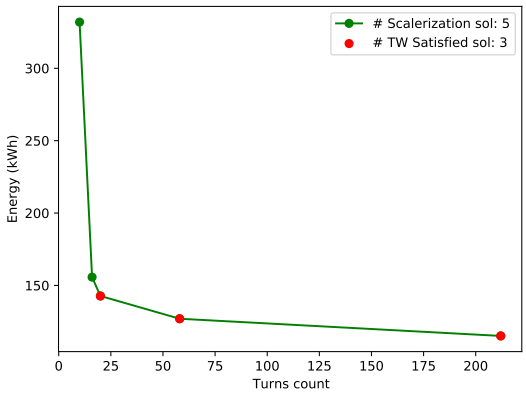


(c) Composite results after 15 minutes

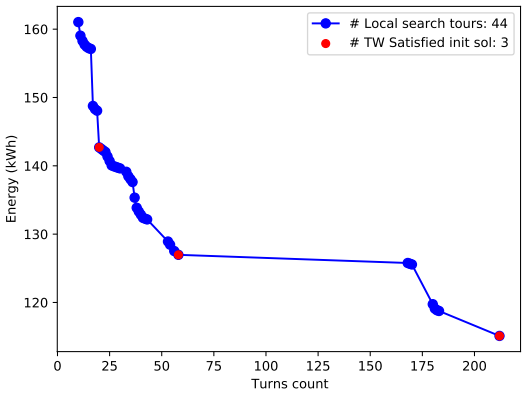
Figure 5.6: Results for route no. 4494 after 15 minutes

Route No.	Initial solution using scalerization			Local Search	
	CM (bSTSP)	CM (bSTSPTW)	HV	CM	HV
18	5	4	0.73	44	0.78
896	5	4	0.79	16	0.8
1643	3	3	0.8	13	0.86
3436	3	1	NA	39	0.96
4236	4	2	0.87	28	0.88
4260	2	2	0	36	0.78
4494	5	3	0.67	44	0.75

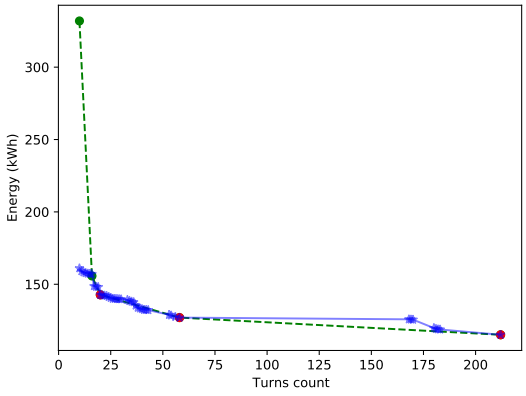
Table 5.7: Local search vs. IP solutions to bSTSPTW after 30 min



(a) Scalerization results after 7.5 minutes

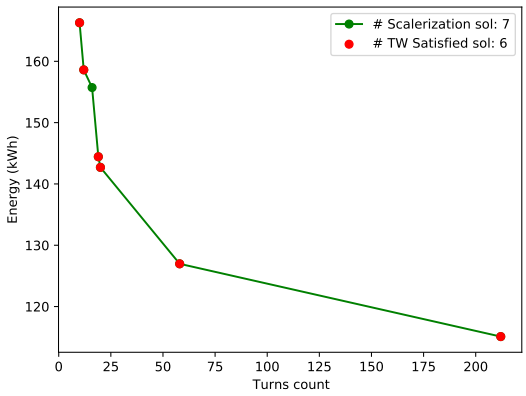


(b) Local search results after 30 minutes

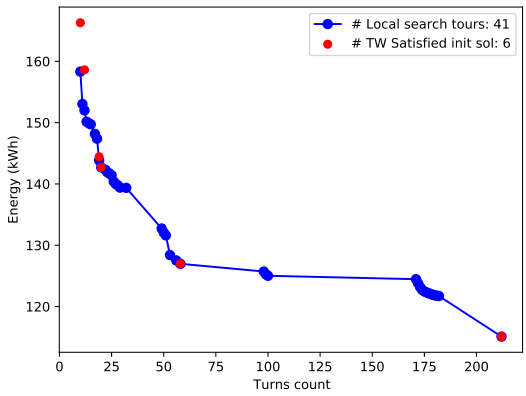


(c) Composite results after 30 minutes

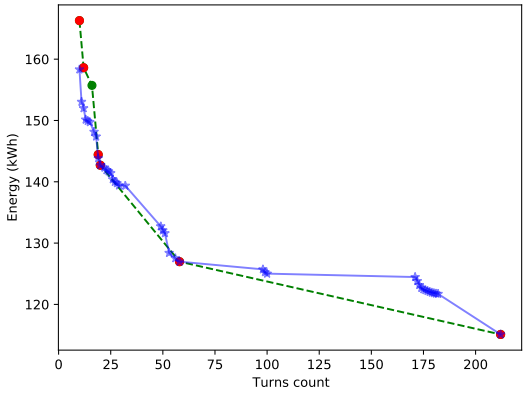
Figure 5.7: Results for route no. 4494 after 30 minutes



(a) Scalerization results after 15 minutes



(b) Local search results after 60 minutes



(c) Composite results after 60 minutes

Figure 5.8: Results for route no. 4494 after 60 minutes

Theoretically, since the IP is an edge-based formulation, its computation time grows only with the size of the graph. On the other hand, the time required for the local search method depends heavily on the number of terminals. Hence, local search is expected to give better results for large graphs. See Table 5.5. Since the IP couldn't give any solution in the stipulated 1 hour in any of the test cases, only the time to get one feasible tour for bSTSPTW using IP formulation was checked. In this case, we set $\alpha = 0$ and $\beta = 1$, which corresponds to the minimum turns case. The reason for choosing this particular alpha, beta combination is that it makes the cost matrix binary, making it easier to solve. Remarkably, the Integer Programming (IP) method was unable to yield a feasible solution within a 24-hour time frame for any of the test cases, indicating the challenging nature of the bSTSPTW problem.

Chapter 6

Conclusions and Future Directions

I am glad you are here with me. Here at the end of all things, Sam.

J.R.R. Tolkien,
The Return of the King

Chapter Overview. This chapter concludes the discussion in the thesis in Section 6.1 and discusses some future research directions of the work in Section 6.2

6.1 Conclusion

Optimizing logistics and delivery operations routes can be complex and challenging since various factors like route length, duration, cost, and safety need to be considered. Additionally, the energy consumption and the number of turns in a route can significantly impact the efficiency of operations, making bi-objective tours that satisfy time windows at terminals an attractive solution for logistics companies. However, finding optimal solutions to this problem is computationally challenging and lacks fast heuristic solution approaches.

To address this gap, this thesis proposes a new local search heuristic based on several new neighborhoods for the Bi-criterion asymmetric steiner traveling salesman problem with time windows (bSTSPTW). The proposed approach is benchmarked against an

IP formulation for bSTSPTW that extends the standard STSP for multiple objectives and time windows. The empirical analysis is conducted on real-world networks from historical route data from the Amazon last-mile routing challenge. The quality of the non-dominated tours on the Pareto-frontier is compared based on several performance indicators like hypervolume and cardinality metrics.

The results indicate that the local search method yields considerably better results and gives many more tours than the IP method, which could not even generate a feasible solution in 24 hours in any test case. Thus, local search characterizes a better frontier. Furthermore, two exact approaches for MOSTSP are proposed, namely *Graph Concatenation* and *Graph Layering*. Compared to the IP, the exact methods were significantly faster. The exact approaches work better with smaller graphs and fewer terminals (less than 6), and the heuristic methods work best with large graphs with a moderate number of terminals (50–200). Thus, the exact approaches provide an alternative for smaller graphs with fewer terminals. The empirical analysis conducted on real-world networks from the Amazon last mile routing challenge demonstrates the effectiveness of the proposed approach in finding non-dominated tours on the Pareto-frontier.

6.2 Future directions

The findings in this thesis call for exploring other advanced heuristic approaches for MOSTSP. Since the quality of solutions provided by local search decreased for higher number of terminals, future heuristics should consider larger-sized neighborhoods than the ones proposed in this work. One such direction is extending the k-opt neighborhoods proposed for standard TSPs towards the MOSTSP, enabling the exploration of a significant portion of the feasible region in each iteration. Another direction would be to explore different scalarization techniques for the IP that better suit the optimization criteria, such as energy consumption and number of turns. Finally, since the problem deals with negative weights, modern approaches that can solve shortest paths in near-linear time can be employed.

Bibliography

Cedric De Cauwer, Wouter Verbeke, Thierry Coosemans, Saphir Faid, and Joeri Van Mierlo. A data-driven method for energy consumption prediction and energy-efficient routing of electric vehicles in real-world conditions. *Energies*, 10(5):608, 2017. 1

World Health Organization. World report on road traffic injury prevention: summary. In *World report on road traffic injury prevention: summary*, pages ix–52. 2004. 2

RTITB. Are we doing enough to prevent workplace fatalities in the transport and logistics industry?, 2023. URL https://www.rtitb.com/. Accessed: 2023-04-21. 2

Chuck Holland, Jack Levis, Ranganath Nuggehalli, Bob Santilli, and Jeff Winters. UPS Optimizes Delivery Routes. *Interfaces*, 47(1):8–23, 2017. 2, 23

Stephen Wood. *Analyzing the Economic Impact of Inefficient Left Turns in Urban Traffic*. PhD thesis, Worcester Polytechnic Institute, 2020. 2

Eun-Ha Choi. Crash Factors in Intersection-Related Crashes: An On-Scene Perspective. 2010. 2

Ernesto Queiros Vieira Martins. on a Multicriteria Shortest Path Problem. *European Journal of Operational Research*, 16(2):236–245, 1984. 4, 11

Oriol Travesset-Baro, Marti Rosas-Casals, and Eric Jover. Transport energy consumption in mountainous roads. a comparative case study for internal combustion engines and electric vehicles in andorra. *Transportation Research Part D: Transport and Environment*, 34:16–26, 2015. 5

Daniel Merchan, Jatin Arora, Julian Pachon, Karthik Konduri, Matthias Winkenbach, Steven Parks, and Joseph Noszek. 2021 amazon last mile routing research challenge: Data set. *Transportation Science*, 2022. 6, 50

OpenStreetMap contributors. Openstreetmap, 2023. URL https://www.openstreetmap.org/. Accessed: 2023-04-21. 6

USGS. Elevation point query service, 2023. URL https://apps.nationalmap.gov/epqs/. Accessed: 2023-04-21. 6

Electrive. Huge order for lion electric trucks from amazon. https://www.electrive.com/2021/01/11/huge-order-for-lion-electric-trucks-from-amazon/, 2021. Accessed: 2023-04-26. 7

Holger R Maier, Saman Razavi, Zoran Kapelan, L Shawn Matott, J Kasprzyk, and Bryan A Tolson. Introductory overview: Optimization using evolutionary algorithms and other metaheuristics. *Environmental modelling & software*, 114:195–213, 2019. 12, 13

George Dantzig, Ray Fulkerson, and Selmer Johnson. Solution of a large-scale traveling-salesman problem. *Journal of the operations research society of America*, 2(4):393–410, 1954. 17, 26

Adam N Letchford, Saeideh D Nasiri, and Dirk Oliver Theis. Compact Formulations of the Steiner Traveling Salesman Problem and Related Problems. *European Journal of Operational Research*, 228(1):83–92, 2013. 18, 19, 25, 26, 27, 38

Gilbert Laporte. The traveling salesman problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59(2):231–247, 1992. 18, 20

Martin Desrochers and Gilbert Laporte. Improvements and extensions to the miller-tucker-zemlin subtour elimination constraints. *Operations Research Letters*, 10(1): 27–36, 1991. 18, 26

Bezalel Gavish and Stephen C Graves. The travelling salesman problem and related problems. 1978. 19, 26

Manfred Padberg and Ting-Yi Sung. An analytical comparison of different formulations of the travelling salesman problem. *Mathematical Programming*, 52(1-3):315–357, 1991. 19

Luis Gouveia and Stefan Voß. A classification of formulations for the (time-dependent) traveling salesman problem. *European Journal of Operational Research*, 83(1):69–82, 1995. 20

Fatma A Karkory and Ali A Abudalmola. Implementation of heuristics for solving travelling salesman problem using nearest neighbour and minimum spanning tree algorithms. *International Journal of Computer and Information Engineering*, 7(10): 1524–1534, 2013. 20

Refael Hassin and Ariel Keinan. Greedy heuristics with regret, with application to the cheapest insertion algorithm for the tsp. *Operations Research Letters*, 36(2):243–246, 2008. 20

Nicos Christofides, Aristide Mingozzi, and Paolo Toth. Exact algorithms for the vehicle routing problem, based on spanning tree and shortest path relaxations. *Mathematical programming*, 20:255–282, 1981. 21

Michael Held and Richard M Karp. The Traveling-Salesman Problem and Minimum Spanning Trees. *Operations Research*, 18(6):1138–1162, 1970. 21

Georges A Croes. A method for solving traveling-salesman problems. *Operations research*, 6(6):791–812, 1958. 21

F Bock. An algorithm for solving'traveling salesman'and related network optimization problems, presented at the 14th national meeting of the operations research society of america, st. *Louis, Missouri*, 1958. 21

Andrius Blazinskas and Alfonsas Misevicius. Combining 2-opt, 3-opt and 4-opt with k-swap-kick perturbations for the traveling salesman problem. *Kaunas University of Technology, Department of Multimedia Engineering, Studentu St*, pages 50–401, 2011. 21

Jean-Yves Potvin, Guy Lapalme, and Jean-Marc Rousseau. A generalized k-opt exchange procedure for the mtsp. *INFOR: Information Systems and Operational Research*, 27 (4):474–481, 1989. 21

Kernighan Lin. Lin s., kernighan bw. *An effective heuristic algorithm for the traveling-salesman problem, Oper. Res*, 21(2):498–516, 1973. 21

Shen Lin and Brian W Kernighan. An Effective Heuristic Algorithm for the Traveling-Salesman Problem. *Operations research*, 21(2):498–516, 1973. 21, 26, 40

Keld Helsgaun. An effective implementation of the lin–kernighan traveling salesman heuristic. *European journal of operational research*, 126(1):106–130, 2000. 21, 22, 40

Sumanta Basu. Tabu search implementation on traveling salesman problem and its variations: a literature survey. 2012. 21

Christopher C Skiscim and Bruce L Golden. Optimization by simulated annealing: A preliminary computational study for the tsp. Technical report, Institute of Electrical and Electronics Engineers (IEEE), 1983. 21

Thomas Stützle, Marco Dorigo, et al. Aco algorithms for the traveling salesman problem. *Evolutionary algorithms in engineering and computer science*, 4:163–183, 1999. 21

Rainer E Burkard. Travelling salesman and assignment problems: A survey. *Annals of Discrete Mathematics*, 4:193–215, 1979. 21

Temel Öncan, İ Kuban Altınel, and Gilbert Laporte. A comparative analysis of several asymmetric traveling salesman problem formulations. *Computers & Operations Research*, 36(3):637–654, 2009. 21

David S Johnson, Gregory Gutin, Lyle A McGeoch, Anders Yeo, Weixiong Zhang, and Alexei Zverovitch. Experimental analysis of heuristics for the atsp. *The traveling salesman problem and its variations*, pages 445–487, 2007. 22

Alan M Frieze, Giulia Galbiati, and Francesco Maffioli. On the worst-case performance of some algorithms for the asymmetric traveling salesman problem. *Networks*, 12(1): 23–39, 1982. 22

Weixiong Zhang. Truncated branch-and-bound: A case study on the asymmetric tsp. In *Proc. Of AAAI 1993 Spring Symposium on AI and NP-hard problems*, volume 160166, 1993. 22

Hitokazu Matsushita, Ogden Mills, Nathan Lambson, and Tony Martinez. The traveling salesman problem: Adapting 2-opt and 3-opt local optimization to branch & bound techniques. 2011. 22

Gerard Sierksma. Hamiltonicity and the 3-opt procedure for the traveling salesman problem. *Applicationes Mathematicae*, 22(3):351–358, 1994. 22

Paris-C Kanellakis and Christos H Papadimitriou. Local Search for the Asymmetric Traveling Salesman Problem. *Operations Research*, 28(5):1086–1099, 1980. 22, 26, 40

Jill Cirasella, David S Johnson, Lyle A McGeoch, and Weixiong Zhang. The asymmetric traveling salesman problem: Algorithms, instance generators, and tests. In *Algorithm Engineering and Experimentation: Third International Workshop, ALENEX 2001 Washington, DC, USA, January 5–6, 2001 Revised Papers*, pages 32–59. Springer, 2001. 22

Nosheen Qamar, Nadeem Akhtar, and Irfan Younas. Comparative Analysis of Evolutionary Algorithms for Multi-Objective Travelling Salesman Problem. *International Journal of Advanced Computer Science and Applications*, 9(2):371–379, 2018. 23

Eric Angel, Evripidis Bampis, and Laurent Gourvès. A dynasearch neighborhood for the bicriteria traveling salesman problem. In *Metaheuristics for Multiobjective Optimisation*, pages 153–176. Springer, 2004. 23

Thibaut Lust and Jacques Teghem. The multiobjective traveling salesman problem: a survey and a new approach. *Advances in Multi-Objective Nature Inspired Computing*, pages 119–141, 2010a. 23

Luis Paquete, Marco Chiarandini, and Thomas Stützle. Pareto local optimum sets in the biobjective traveling salesman problem: An experimental study. In *Metaheuristics for multiobjective optimisation*, pages 177–199. Springer, 2004. 23

Luis Paquete and Thomas Stützle. A two-phase local search for the biobjective traveling salesman problem. In *Evolutionary Multi-Criterion Optimization: Second International*

Conference, EMO 2003, Faro, Portugal, April 8–11, 2003. Proceedings 2, pages 479–493. Springer, 2003. 23

A Gupta and A Warburton. Approximation methods for multiple criteria travelling salesman problems. In *Toward Interactive and Intelligent Decision Support Systems: Volume 1 Proceedings of the Seventh International Conference on Multiple Criteria Decision Making, Held at Kyoto, Japan, August 18–22, 1986*, pages 211–217. Springer, 1987. 23

Thibaut Lust and Jacques Teghem. Two-Phase Pareto Local Search for the Biobjective Traveling Salesman Problem. *Journal of Heuristics*, 16(3):475–510, 2010b. 23, 26, 38

Weiqi Li. Finding pareto-optimal set by merging attractors for a bi-objective traveling salesmen problem. In *Evolutionary Multi-Criterion Optimization: Third International Conference, EMO 2005, Guanajuato, Mexico, March 9-11, 2005. Proceedings 3*, pages 797–810. Springer, 2005. 23

Danny De Schreye. Solving tsp with time windows with constraints. 1999. 24

Rodrigo Ferreira Da Silva and Sebastián Urrutia. A general vns heuristic for the traveling salesman problem with time windows. *Discrete Optimization*, 7(4):203–211, 2010. 24

JW Ohlmann. for the traveling salesman problem with time windows informs journal on computing vol. 19, 2007. 24, 26

Keld Helsgaun. An extension of the lin-kernighan-helsgaun tsp solver for constrained traveling salesman and vehicle routing problems. *Roskilde: Roskilde University*, 12, 2017. 24, 26, 40

Jing-Quan Li. A bi-directional resource-bounded dynamic programming approach for the traveling salesman problem with time windows. *Submitted manuscript*, 2008. 24, 26

John N Tsitsiklis. Special cases of traveling salesman and repairman problems with time windows. *Networks*, 22(3):263–282, 1992. 24

Imdat Kara, Ozge Nimet Koc, Fulya Altıparmak, and Berna Dengiz. New integer linear programming formulation for the traveling salesman problem with time windows:

minimizing tour duration with waiting times. *Optimization*, 62(10):1309–1319, 2013. 24

J urgen Antes and Ulrich Derigs. A new parallel tour construction algorithm for the vehicle routing problem with time windows. Technical report, Technical report, Lehrstuhl f ur Wirtschaftsinformatik und Operations . . . , 1995. 24

Marius M Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations research*, 35(2):254–265, 1987. 24, 26

Chi-Bin Cheng and Chun-Pin Mao. A modified ant colony system for solving the travelling salesman problem with time windows. *Mathematical and Computer Modelling*, 46(9-10): 1225–1235, 2007. 24, 26

Robert A Russell. An effective heuristic for the m-tour traveling salesman problem with some side conditions. *Operations research*, 25(3):517–524, 1977. 24, 26

Huili Zhang, Weitian Tong, Yinfeng Xu, and Guohui Lin. The Steiner Traveling Salesman Problem with Online Edge Blockages. *European Journal of Operational Research*, 243 (1):30–40, 2015. 25, 26

Eduardo Álvarez-Miranda and Markus Sinnl. A note on computational aspects of the steiner traveling salesman problem. *International Transactions in Operational Research*, 26(4):1396–1401, 2019. 25, 26, 38

Ruben Interian and Celso C Ribeiro. A Grasp Heuristic Using Path-Relinking and Restarts for the Steiner Traveling Salesman Problem. *International Transactions in Operational Research*, 24(6):1307–1323, 2017. 25, 26, 38

David Applegate, Robert Bixby, Vašek Chvátal, and William Cook. Implementing the dantzig-fulkerson-johnson algorithm for large traveling salesman problems. *Mathematical programming*, 97:91–153, 2003. 25, 38

Adam N. Letchford and Saeideh D. Nasiri. The Steiner Travelling Salesman Problem with Correlated Costs. *European Journal of Operational Research*, 245(1):62–69, 2015. ISSN 0377-2217. URL https://www.sciencedirect.com/science/article/pii/S0377221715001642. 25

Zhenyu Yan, Linghai Zhang, Lishan Kang, and Guangming Lin. A New Moea for Multi-Objective TSP and Its Convergence Property Analysis. In *International Conference on Evolutionary Multi-Criterion Optimization*, pages 342–354. Springer, 2003. 26

Anubha Agrawal, Nitish Ghune, Shiv Prakash, and Manojkumar Ramteke. Evolutionary Algorithm Hybridized with Local Search and Intelligent Seeding for Solving Multi-Objective Euclidian TSP. *Expert Systems with Applications*, 181:115192, 2021. 26

Virginie Gabrel, A Ridha Mahjoub, Raouia Taktak, and Eduardo Uchoa. The Multiple Steiner TSP with Order Constraints: Complexity and Optimization Algorithms. *Soft Computing*, 24(23):17957–17968, 2020. 26

Robert B Dial. Bicriterion Traffic Assignment: Basic Theory and Elementary Algorithms. *Transportation science*, 30(2):93–111, 1996. 27, 30

Sanjeeb Dash, Oktay Günlük, Andrea Lodi, and Andrea Tramontani. A time bucket formulation for the traveling salesman problem with time windows. *INFORMS Journal on Computing*, 24(1):132–147, 2012. 28

João CN Clímaco and Marta MB Pascoal. An Approach to Determine Unsupported Non-Dominated Solutions in Bicriteria Integer Linear Programs. *INFOR: Information Systems and Operational Research*, 54(4):317–343, 2016. 29

Eckart Zitzler, Lothar Thiele, Marco Laumanns, Carlos M Fonseca, and Viviane Grunert Da Fonseca. Performance Assessment of Multiobjective Optimizers: An Analysis and Review. *IEEE Transactions on evolutionary computation*, 7(2):117–132, 2003. 54

Charles Audet, Jean Bigeon, Dominique Cartier, Sébastien Le Digabel, and Ludovic Salomon. Performance Indicators in Multiobjective Optimization. *European journal of operational research*, 292(2):397–422, 2021. 54

J. Blank and K. Deb. pymoo: Multi-objective optimization in python. *IEEE Access*, 8: 89497–89509, 2020. 55

Nery Riquelme, Christian Von Lücken, and Benjamin Baran. Performance Metrics in Multi-Objective Optimization. In *2015 Latin American Computing Conference (CLEI)*, pages 1–11. IEEE, 2015. 55