DISSERTATION
submitted
to the
Combined Faculty for the Natural Sciences and Mathematics
of
Heidelberg University, Germany
for the degree of
Doctor of Natural Sciences

Put forward by

Eng. Sc.: Tuan Nam Nguyen .....
Born in: Thanh Hoa, Vietnam .....
Oral examination: .....

Doctoral Thesis

Solving Assignment and Routing Problems in Mixed Traffic Systems

Author:
Tuan Nam Nguyen

Supervisor:
Prof. Dr. Gerhard Reinelt
Second supervisor:
Prof. Dr. Michael Gertz

Heidelberg, Germany

To my beloved niece - Yen Linh Do
to my beloved nephew - Xuan Thang Do
and to my beloved family

Abstract

This doctoral thesis presents not only a new traffic assignment model for mixed traffic systems but also new heuristics for multi-paths routing problems, a case study in Hanoi Vietnam, and a new software, named TranOpt Plus, supporting three major features: map editing, dynamic routing, and traffic assignment modeling.

We investigate three routing problems: k shortest loop-less paths (KSLP), dissimilar shortest loop-less paths (DSLP), and multi-objective shortest paths (MOSP). By developing loop filters and a similarity filter, we create two new heuristics based on Eppstein's algorithm: one using loop filters for the KSLP problem (HELF), the other using loop-and-similarity filters for the DSLP problem (HELSF). The computational results on real street maps indicate that the new heuristics dominate the other algorithms considered in terms of either running time or the average length of the found paths.

In traffic assignment modeling, we propose a new User Equilibrium (UE) model, named GUEM, for mixed traffic systems where 2- and 4-wheel vehicles travel together without any separate lanes for each kind of vehicle. At the optimal solution to the model, a user equilibrium for each kind of vehicle is obtained. The model is applied to the traffic system in Hanoi, Vietnam, where the traffic system is mixed traffic dominated by motorcycles. The predicted assignment by the GUEM model using real collected data in Hanoi is in high agreement with the real traffic situation in Hanoi.

Finally, we present the TranOpt Plus software, containing the implementation of all the routing algorithms mentioned in the thesis, as well as the GUEM model and a number of popular traffic assignment models for both standard traffic systems and mixed traffic systems. With its intuitive graphical user interface (GUI) and its strong visualization tools, TranOpt Plus also enables users without any mathematical or computer science background to use conveniently. Nevertheless, TranOpt Plus can be easily extended by further map-related problems, e.g., transportation network design, facility location, and the traveling salesman problem.

Keywords: mixed traffic assignment modeling, routing algorithms, shortest paths, dissimilar paths, Hanoi, TranOpt Plus, map visualization

Zusammenfassung

Diese Dissertation präsentiert nicht nur ein neues Verkehrszuordnungs-Modell für gemischte Verkehrssysteme, sondern auch neue Heuristiken für Mehrwege-Routing-Probleme, sowie eine Fallstudie in Hanoi und eine neu entwickelte Software namens TranOpt Plus mit drei Hauptanwendungen: Kartenbearbeitung, dynamisches Routing und Modellierung der Verkehrszuordnung.
Wir untersuchen drei Routing-Probleme: k kürzeste schleifenfreie Wege (KSLP), ungleiche kürzeste schleifenfreie Wege (DSLP) und Pareto optimale kürzeste Wege (MOSP). Durch die Entwicklung von Schleifenfiltern und eines Ähnlichkeitsfilters, ergeben sich zwei neue Heuristiken auf Basis des Eppstein-Algorithmus: Eine mit Schleifenfilter für das KSLP-Problem (HELF) und eine mit Schleifen- und Ähnlichkeitsfiltern für das DSLP-Problem (HELSF). Die Rechenexperimente mit realen Straßenkarten einiger Städte zeigen, dass die neuen Heuristiken die bisherigen in Bezug auf Laufzeit oder durchschnittliche Pfadlänge dominieren.
Im neuen Verkehrszuordnungs-Modell (TAM) schlagen wir eine neues User-Equilibrium-Modell (UE) für gemischte Verkehrssysteme vor, in denen sich 2- und 4-rädrige Fahrzeuge gemeinsam bewegen, ohne gesonderte Spuren für jede Fahrzeugart zu verwenden. Bei der optimalen Lösung für das Modell bleibt das Gleichgewicht für jede Fahrzeugart erhalten. Das Modell wird auf das Verkehrssystem in Hanoi, wo der gemischte Verkehr von Motorrädern dominiert wird, angewendet. Die vom Modell produzierten Ergebnisse stimmen mit der realen Verkehrssituation in Hanoi deutlich überein.
Schließlich präsentieren wir die Software TranOpt Plus, welche alle in der Arbeit erwähnten Routing-Algorithmen sowie die Verkehrszuordnungs-Modelle sowohl für Standard- als auch gemischten Verkehr umfasst. Durch die grafische Benutzeroberfläche (GUI) und die umfangreichen Visualisierungstools, kann die Software auch von Benutzern ohne weiterführende Informatik- oder Mathematik-Kenntnisse verwendet werden. Dennoch kann TranOpt Plus leicht um zusätzliche Graphenprobleme, wie beispielsweise Netzwerkdesign, das Facility-Location- und das Traveling-Salesman-Problem, erweitert werden.

Stichwörter: Verkehrszuordnungs-Modelle für gemischte Verkehrssysteme, Routing-Probleme, kürzeste Wege, ungleiche kürzeste Wege, Hanoi, TranOpt Plus, Kartenvisualisierung

Acknowledgments

I yield my first thanks to my supervisor Prof. Dr. Gerhard Reinelt for giving me the great chance of studying in his research group at the Faculty of Mathematics and Computer Science, Heidelberg University. He also carefully instructed and encouraged very much during my PhD studies. Without his support, I would not have had enough motivation to finish this study.
I appreciate my second supervisor Prof. Dr. Michael Gertz for giving me valuable instructions, dealing with data. Moreover, I also received from him important comments on the research direction in the second year of study.

I am grateful to all of my friends in the Discrete and Combinatorial Optimization Group, e.g. our secretary Mrs. Catherine Proux-Wieland, Dr. Stefan Wiesberg, Thomas Metz, Dr. Markus Speth, Dr. Achim Hildenbrandt, Dr. Veronika Halasz, Francesco Silvestri, Ekaterina Tikhoncheva, Ruobing Shen, our system administrator Jonas Gröse Sundrup, and all the friends in the Database Systems Research Group, as well as my friends in the Mensa group for helping me not only in my studies but also in everyday life. The working atmosphere around them is somehow like in a big international family where people are very friendly to each other.
I would like to express my gratitude to the EMMA organization and the Heidelberg graduate school (HGS) for providing the scholarships for my PhD studies for more than 4 years. All the financial supports and the working environment are of great importance to me. I appreciate the PTV group for providing a scientific license for the VISSIM and the VISUM software. They are surely powerful for traffic planning as well as traffic visualization.

I am much delighted to all my research partners, e.g. Prof. Dinh The Luc, Dr. Pham Hong Quang, Dr. Do Quang Thuan, and Truong Hoang Hai, for providing me precious traffic data. Thanks also go to my friends, especially Mrs. Ria Hillenbrand-Lynott, Mrs. Ho Hanh, Phan Thu Trang, Dr. Le Van Quoc Anh, and Dr. Nguyen Manh Hung, for supporting me in my daily life and for giving me more motivation for my research.
I really appreciate Tran Van Tra, Dr. Stefan Wiesberg, Mai Tran, Maria Rupprecht, Enrique Guerrero-Merino, and all the other people who helped me edit this thesis. Your comments are very valuable for me.
Finally, I would like to express my large gratitude to my family who have always stood by me in everything I do, whenever I need them. They have given me unconditional love for my whole life.

Contents

Chapter 1

Introduction

Presently urban traffic planning (UTP) plays an increasingly important role in a large number of countries in the world. One of the fundamental problems in UTP is traffic assignment modeling (TAM), that is to forecast traffic flows on the links of a transportation network whenever a change has been made on the system, such as building new roads, closing current ones, changing traffic rules, and modifying the signal light system, etc. Without the ability to predict traffic flows, managers are not able to evaluate UTP projects accurately. Any incorrect decision in UTP projects could result in an unnecessary waste of invested money and/or may lead to fatal accidents. Furthermore, urban traffic systems are becoming more complex, i.e. more vehicles of various kinds are used, more interactions between vehicles and the traffic system are observed, therefore it is essential to continuously improve TAM in order to keep it up-to-date with the development of the modern UTP.

1.1 Motivation

While TAM has been rigorously studied in developed countries, e.g. USA, Germany, and Japan, it is still in the beginning phase of investigation in a number of developing countries, e.g., Vietnam, the Philippines, Indonesia, Thailand, and India. The motivation of this doctoral thesis comes from the urban traffic in Vietnam where commuters in major cities are suffering serious traffic-related problems, such as traffic jams, air pollution, and fatal accidents. According to the investigations of the World Bank [64, 65] and Japan International Cooperation Agency & Hanoi People's Committee [42], the greatest challenge in the coming decades for the two largest cities in Vietnam, i.e. Hanoi—the capital city—and Ho Chi Minh City—the economically most powerful city—is spatial planning for future urbanization.
Developing infrastructure and optimizing traffic systems are two fundamental components in spatial planning, thus the government of Vietnam has invested a large amount of money in them, e.g. 4.5% of the gross domestic product (GDP) of the

country in 2002. However, the effectiveness of these investments is not as high as expected due to a number of factors, such as budget management and a serious lack of dedicated research on the traffic system in Vietnam.
In addition, the urban traffic system in Vietnam is a mixed traffic system dominated by motorcycles (MTSDM) where motorcycles occupy about 75% to 80% of the whole share of all the kinds of vehicles. As far as we know, there is no dedicated traffic assignment (TA) model for the MTSDM in Vietnam at present. This leads to a fact that all major UTP projects depend on TA models, which are not originally designed for the traffic system in Vietnam and thus are based on different standards. Although they can probably be customized to deal with mixed traffic systems, the accuracy of their applications to the traffic system in Vietnam is still in doubt.

In order to help traffic managers evaluate UTP projects reliably, we investigates a new TA model, that strongly supports mixed traffic systems, especially for those dominated by motorcycles. Such a model should be particularly meaningful for cities like Hanoi in terms of saving public investment and coping with the present traffic problems.

1.2 Problems and Methods

TAM involves a number of scientific fields in both mathematics and computer science, such as discrete optimization, constrained continuous programming problems, and computer visualization, etc. In order to sketch a general picture of the workflow, we briefly introduce major problems and the corresponding methods in the following subsections. The details of these problems and methods will be given in the later chapters.

1.2.1 Routing Problems

Problems of finding the best paths in terms of a given number of objectives in a graph (or transportation network) are called routing problems. They are a part of discrete optimization with a wide range of applications in both academic research and real problems. Depending on the given objectives and constraints, routing problems can be divided into several categories. For instance, the shortest path (SP) problem is a classical problem, that finds a shortest path between two nodes in a given graph. A generalized version of the SP problem is the k shortest paths (KSP) problem where the first $k \geq 1$ shortest paths are determined, i.e. determining a set of the shortest path, the second shortest path, ..., the k^{th} shortest path, between two nodes. When k equals the number of possible paths, the problem is called the ranking shortest path problem. If the paths are required to be loop-less, the problem is named as the k shortest loop-less paths (KSLP) problem.

If the dissimilarity between each pair of paths is taken into account, then the problem is the dissimilar shortest loop-less paths (DSLP) problem. The dissimilarity between two paths is determined by the total length of the common links and the total length of the different links between them. The DSLP problem has an important application in transport of hazardous materials where vehicles are not allowed to travel in the same path since the radiation materials could be accumulated through time and have negative impacts on people living along the path. In this case, transport companies must investigate a set of alternative paths which are short in length and dissimilar from each other.
When the links in the graph have various objectives (properties), e.g. length, traveling times at free flow and at congested flow, drivers may choose their paths according to not only one but many objectives. This problem is called the multi-objective shortest path (MOSP) problem. An example of this problem is in traffic systems where traffic congestion happens frequently, drivers may select paths according to both their lengths and the probability of congestion on the paths.
A number of algorithms were proposed to deal with routing problems, for instance Dijkstra's algorithm and the A-Star algorithm for the SP problem, Yen's algorithm, Martins' algorithm, and Eppstein's algorithm for the KSP problem, etc.

1.2.2 Traffic Assignment Modeling

Forecasting traffic flows on the links of a transportation network with given traffic demands includes three major steps. The first step is to investigate the characteristics of the traffic system as well as the behaviors of drivers. Based on that information, a number of assumptions are proposed. The second step is to formulate all the assumptions as a complete mathematical programming problem, that can be solved by a suitable method. The last step is to evaluate the model by comparing the optimal solution of the mathematical programming problem with real traffic data.
For instance, the all-or-nothing (AON) model is based on the assumption that drivers choose a shortest path from a source node to a destination node with respect to only the lengths of possible paths. The model, however, does not take the possible traffic delay time into account. This is considered as the biggest disadvantage of the AON model.
Another well known model is the user equilibrium (UE) model which is based on the first principle of Wandrop [67] that drivers choose the best path in terms of traveling time, i.e. the traveling times of all the used paths between two nodes are the same and less than those of any unused possible path. Due to the similarity between the principle and real driving behaviors, a number of UE-based models have been proposed, such as the stochastic user equilibrium and the dynamic user equilibrium models. These models are included in most of modern traffic planning

software. Since UE models are formulated as constrained continuous programming problems, they can be solved efficiently by the Frank-Wolfe algorithm [27], that was significantly improved by LeBlanc et al. [46] with a grouping technique.
Developing TA models for standard traffic systems, which contain mainly one kind of vehicle or many kinds with dedicated lanes for each, is significantly simpler than those for mixed traffic systems where vehicles of various kinds travel together on the same lanes. It results from the complexity of the characteristics of mixed traffic flows and the differences in routing behaviors of drivers.

1.2.3 Map Visualization

Working on TAM requires visualization tools, which can simulate all map objects on computers, e.g. nodes, links, and intersections, such that it is convenient to manage the properties of objects and to comprehend the computational results of TA models. In the scope of this thesis we investigate a software, named TranOpt Plus, strongly supporting map visualization. The current version has three major features: map editing, dynamic routing, and traffic assignment modeling.
One of the most challenging tasks in map visualization is to find a proper data structure for representing large maps, that may contain millions of objects. A proper data structure should not consume too much storage and it must provide quick access to map objects. Displaying a large number of objects in a short time is also a challenge. In order to avoid technical difficulty in computer graphic, we employ the Qt—a cross-platform application framework for building software, that can run on various kinds of hardware platform, e.g. Windows, Linux, OS X, and Unix-like. Qt provides various graphical libraries which can handle efficiently large graphs with millions of objects. We also use other open-source graphics libraries, such as the MapGraphics library for importing map images and the QCustomPlot library for data visualization.
Since we aim at working on real street maps of cities worldwide, it is important to possess a feature for data importing, so that we can utilize available map resources, such as OpenStreetMap or Google map.

1.3 Outline and Contributions

The thesis is divided into 8 chapters. While this chapter provides a brief introduction on the problems and the methods, the next chapter, i.e. Chapter 2, gives the background knowledge needed in later chapters. Chapter 3 introduces the classical shortest path (SP) problem where only a shortest path is determined. The chapter also gives the details of two well known algorithms for the SP problem: Dijkstra's and A-Star algorithms. They are used widely in a number of algorithms proposed for the KSP problem, that is investigated in Chapter 4. Chapter 5 investigates

traffic assignment modeling with emphasis on the user equilibrium model. After introducing mathematical formulations and methods for solving the UE model, a new UE model for mixed traffic systems is proposed along with a mathematical method for solving the new model. In Chapter 6 the new model is examined on the traffic system in Hanoi with collected data including the results of the new online survey on traffic behaviors in Vietnam. Chapter 7 introduces a new software, named TranOpt Plus, that has three main features: graph editor, dynamic routing, and traffic assignment modeling. The last chapter, i.e. Chapter 8, is for conclusions, discussions, as well as the perspective of further works. While Appendix A provides the list of the map instances used in all experiments in the thesis, Appendix B gives the details of computational results.

Our major contributions in this doctoral thesis are summarized as follows.

Two heuristics for KSP problems: We propose a new heuristic, named HELF, that is based on Eppstein's algorithm using loop-filters for the KSLP problem. The HELF dominates the other examined methods in terms of running time.

A new TA model for mixed traffic systems: The new user equilibrium traffic assignment model, named GUEM, supports strongly for mixed traffic systems, especially mixed traffic systems dominated by motorcycles. In the new model, the traffic demand and the potential paths between each origin-destination pair of zones are separated according to vehicle kinds. The model ensures that there exists a user equilibrium for each kind of vehicle. Moreover, it can be solved efficiently by the Frank-Wolfe algorithm with an improvement of LeBlanc et al.

A case study in Hanoi Vietnam: In order to run the new model GUEM on the traffic system in Hanoi, we do not only collect the available data resources, but also launch an online survey on traffic behavior of drivers living in urban areas in Vietnam. The survey has 316 participants by September 15, 2015. New formulas for estimating the parameters of the BPR traveling time function, applying on roads in Hanoi, are investigated with promising computational results. Finally, the GUEM model is examined on the traffic system in Hanoi with all the collected data and the BPR function using new calibrated parameters. The computational results proves the model to be highly accurate. This opens a large number of further applications in traffic planning particularly in Hanoi and generally in other cities with mixed traffic systems.

The TranOpt Plus software: The software, programmed in C++ since 2013, has three major features, namely map editing, dynamic routing, and traffic assignment modeling. Various routing algorithms and traffic assignment models, including the new routing heuristics HELF, HELSF, and the new model GUEM,

are implemented and embedded into TranOpt Plus. Moreover, the software provides a framework for further map-based applications, especially for those in traffic planning.

Chapter 2

Preliminaries and Terminologies

This chapter provides background knowledge and terminologies needed in the later chapters. In the first section, i.e. Section 2.1, we introduce basic definitions of sets and vector spaces. Section 2.2 discusses complexity theory, that is important for evaluating algorithms. Section 2.3 introduces briefly graph theory, while Section 2.4 investigates priority queues and its efficient implementations: heaps. Optimization problems are introduced in Section 2.5 with emphases on the first-order optimality conditions and the Frank-Wolfe algorithm. The last section, i.e. Section 2.6, is spent on traffic assignment modeling, where a number of popular models are reviewed. More details of the topics covered in this chapter can be found in the following materials.

1 T.H. Cormen, *Introduction to algorithms*. MIT press, 2009, [15].

2 B. Korte and J. Vygen, *Combinatorial Optimization: Theory and Algorithms*, fifth edition, Springer, 2011, [45].

3 J. Nocedal and S.J. Wright, *Numerical Optimization*, second edition, Springer, 2006, [56].

4 Y. Sheffi, *Urban Transportation Networks: Equilibrium Analysis with Mathematical Programming Methods*. Prentice-Hall, Inc., Massachusetts Institute of Technology, 1985, [60].

2.1 Sets and Vector Spaces

A **set** is a collection of distinguishable elements, also called components, e.g. the set of natural numbers $\mathbb{N} = \{0, 1, 2, \ldots\}$. If an element e is in a set D, it is written as $e \in D$. The number of elements of D is denoted as $|D|$. A set without any element is the empty set, denoted $\emptyset$. A set C is a **subset** of a set D, denoted $C \subseteq D$, if every element of C is also an element of D, i.e., $e \in C$ implies $e \in D$.

A **vector** is a collection of ordered elements. We denote vector A of size n as $A = (a_1, a_2, \ldots, a_n)$. The n-dimensional **vector space** $\mathbb{R}^n$, where $n \geq 1$, is the set of vectors of n real numbers, i.e.

$$\mathbb{R}^n = \{(r_1, r_2, \ldots, r_n) \mid r_i \in \mathbb{R}, i = 1, 2, \ldots, n\}.$$

A set of points $P \subset \mathbb{R}^n$ is a **convex set** if, for all x, y in P and all α in the interval $[0, 1]$, the point $\alpha x + (1 - \alpha)y$ is also in P.

2.2 Complexity Theory

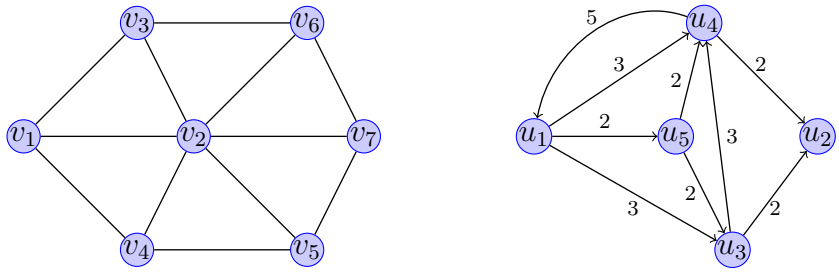
In mathematics and computer science, an **algorithm** is a set of rules or operations to be performed for solving a problem. In order to evaluate an algorithm in terms of running time, a function is investigated to express how fast the running time of the algorithm grows as the size of the input increases. This function is called the **time complexity** of the algorithm or complexity for short. The complexity function of an algorithm is based on the number of basic operations needed to implement the algorithm, e.g. plus, divide, multiply, assignment, comparison, and jump. For the purpose of estimating the worst case, or an upper bound, of the running time of an algorithm, the **big-O notation** is used. We have $f(n) = O(g(n))$ if there exist two positive constants $\alpha \in \mathbb{R}^+$ and $n_0 \in \mathbb{N}$, such that $f(n) \leq \alpha g(n)$ for all $n \geq n_0$. A **polynomial time algorithm** is an algorithm with the time complexity function $f = O(g)$ for some polynomial function g.
The **space complexity** of an algorithm is a function for estimating the growth of the memory space needed by the algorithm for performing on computers. The big-O notation is also used to estimate the worst case of space complexities.

2.3 Graphs

A graph is a mathematical concept with a wide range of applications, e.g. routing problems, the traveling salesman, and logistic problems, etc. In this section we introduce not only basic definitions in graph theory but also the graph representation and trees—a special type of graph.

2.3.1 Definitions

A **graph** is a set of two basic finite sets: the set of nodes V and the set of links E. It is normally denoted as $G(V, E)$. A **node** of a graph is also called vertex or end-point. Each node can have one or many attributes, e.g. name, capacity, and location, etc. A **link**, also called an edge or an arc, is a connection between two nodes.



(a) Undirected graph (b) Directed graph

Figure 2.1: Examples of graphs.

Given a graph $\overline{G}(\overline{V}, \overline{E})$, we say that $\overline{G}$ is a **subgraph** of a graph $G(V, E)$ if $\overline{V} \subseteq V$ and $\overline{E} \subseteq E$. A node $v_j \in V$ is an **adjacent node** of a node $v_i \in V$ if there is a link connecting them. The **degree** of a node v_i, denoted as $deg(v_i)$, is the number of links incident to v_i. The **maximum degree** of a graph is the maximum degree of its nodes.
A link with an associated direction is a **directed link**. The directed link from a node v_i to a node v_j is denoted as (v_i, v_j), e_{ij} or simply (i, j) without any misunderstanding. Each link has its attributes, e.g. the source node, the destination node, the length, the flow, and the capacity. The length (or so-called weight) of a link (v_i, v_j) is denoted as $w(v_i, v_j)$ or w_{ij}.

A graph is said to be **connected** if for any pair of nodes of the graph, there exists a path connecting them. Graphs without any directed links are called **undirected graphs**, otherwise they are **directed graphs** (or digraphs). Figure 2.1 (a) shows an example of an undirected, connected graph, while Fig. 2.1 (b) shows an example of a directed graph where the length of each link is given.

A **path**, say p, from a source node s to a destination node t is a sequence of nodes, e.g. $p = (v_1, v_2, \ldots, v_{k-1}, v_k)$, where $v_1 = s$, $v_k = t$, and $(v_i, v_{i+1}) \in E$ for all $i = 1, \ldots, k-1$. The length of the path p is defined as the total length of its links, i.e. $W(p) = \sum_{e \in p} w(e)$. A path p from s to t is called a **shortest path** if its length is not greater than that of any other possible path from s to t, i.e. $p \in P_{st}$ and $W(p) \leq W(q)$ for all $q \in P_{st}$. A **cycle** (or **loop**) is a path whose the destination node is also the source node, e.g. $(s, v_2, v_3, \ldots, v_l, s)$. A cycle with a negative length is called **negative cycle** (or absorbent cycle). Paths without cycles, i.e. no repeated nodes, are called **loop-less paths**. The **number of links** on a path p is denoted as $|p|$. A path p is **finite** if $|p|$ is finite, otherwise, p is an **infinite path**. The set of all possible paths from s to t is denoted as $P_{s,t}$.

Given two paths $p_1 = (e_1, e_2, \ldots, e_h)$ and $p_2 = (l_1, l_2, \ldots, l_k)$, the **similarity**, denoted as $S(p_1, p_2)$ and the **dissimilarity**, denoted as $D(p_1, p_2)$, between p_1 and p_2 are

defined as

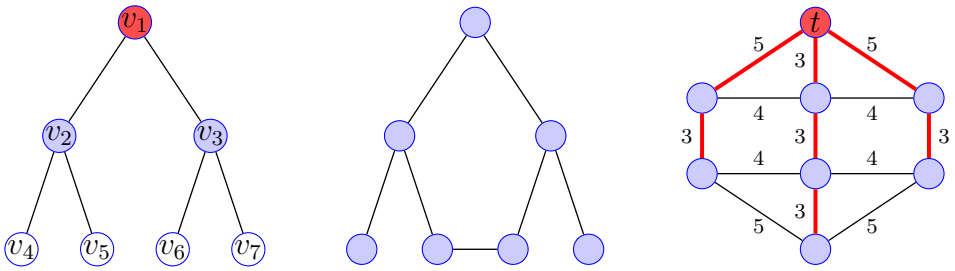
$$S(p_1, p_2) = \frac{1}{2}\left(\frac{W(p_1 \cap p_2)}{W(p_1)} + \frac{W(p_1 \cap p_2)}{W(p_2)}\right) \text{ and } D(p_1, p_2) = 1 - S(p_1, p_2),$$

where $W(p_1 \cap p_2)$ is the total length of the common links of p_1 and p_2. When the destination node of p_1 is the same as the source node of p_2, i.e. $e_h = l_1$, the operation concatenating the two paths, named **link join operation** or concatenating operation, is defined as

$$p_1 \oplus p_2 = (e_1, e_2, \ldots, e_h, l_2, l_3, \ldots, l_k).$$

2.3.2 Trees

A **tree** is an undirected, connected graph without any cycles. There exist a unique loop-less path connecting two nodes in a tree. A **rooted tree** is a tree where one node is selected as a **root node**. In a rooted tree, if (u, v) is the last link on the path from the rooted node to the node v, then u is called the **parent node** of v and v is a **child node** of u. Each node except the rooted node has a unique parent node. Nodes with the same parent are **sibling nodes**. Nodes without children are called **leaf nodes**, or leaves for short. A tree where each node has at most two children is a **binary tree**.



(a) (b) (c)

Figure 2.2: Examples of (a) a tree, (b) not a tree, and (c) a shortest path tree of a graph.

Figure 2.2 (a) shows an example of a binary tree with the root node v_1 and 4 leaves v_4, v_5, v_6, and v_7. The node v_1 is the parent node of v_2 and v_3, in other words, v_2 and v_3 are sibling nodes with the same parent v_1. In turn, the node v_2 is the parent node of the leaves v_4 and v_5, the node v_3 is the parent node of v_6 and v_7. Lets denote the rooted tree in Fig. 2.2 (a) as H, then $H.top()$ refers to the root node of H, i.e. v_1. The set of all the children of a node v on H is GETCHILDREN(H, v), e.g. GETCHILDREN(H, v_1)= $\{v_2, v_3\}$. The graph shown in Fig. 2.2 (b) is not a tree since it contains a cycle.

Given an undirected, connected graph $G(V, E)$, a **shortest path tree** (SPT) to a node t of the graph G, denoted as SPTT(t), is a tree and a subgraph of G with the same set of nodes V, such that the unique path from any other node to t in the tree is a shortest path from that node to t in G. For instance, Fig. 2.2 (c) shows a graph with 8 nodes, the subgraph with all the nodes and all the red, bold links is a shortest path tree to the node t of the graph.

2.3.3 Graph Representation

There are two main approaches for graph representation. In the first approach, named **adjacency matrix**, information of the links of a graph $G(V, E)$ is stored in a $|V| \times |V|$ matrix, i.e., the information of a link (i, j) is stored in the element $A[i, j]$ of the adjacency matrix A. The second approach, called **adjacency list**, stores the graph's information in $|V|$ lists. Each list consists of all the in-going links (and/or out-going links) of a node.
Table 2.1 shows the comparison in terms of time and space complexities of the both approaches to represent a graph $G(V, E)$ where k is the maximum degree of the graph. The first four rows indicate the time complexities of the approaches, while the last row shows the space complexities. The first approach, i.e. the adjacency matrix, has advantages in the first three operations, i.e. adding, deleting, and searching a link, however, the second approach has advantage in the fourth operation, i.e. counting adjacent nodes, and in space complexity when the graph G is sparse, i.e. when the number of edges $|E|$ is much less than $|V|^2$.

Table 2.1: Comparison of two approaches for graph representing.

Operations	**Adjacency matrix**	**Adjacency list**
Adding a new link	$O(1)$	$O(1)$
Deleting a link	$O(1)$	$O(k)$
Searching a link	$O(1)$	$O(k)$
Counting adjacent nodes	$O(\|V\|)$	$O(k)$
Memory consumption	$O(\|V\|^2)$	$O(\|E\|)$

2.4 Priority Queues and Heaps

Many algorithms require a data structure, that can process a large number of elements regarding their comparable keys. One of the most frequently used data structures is a heap, that is an efficient implementation of a priority queue. This section briefly summarizes the basic facts of them.

2.4.1 Priority Queues

A **priority queue** is an abstract data structure for maintaining a number of elements where each of them has a comparable key, e.g. a real number. A priority queue supports three basic operations: inserting new element, indicating and extracting an element with the optimal key. There are two types of priority queues: **min priority queues** (MinPQ) and **max priority queue** (MaxPQ) where the optimal keys are the smallest and the largest keys, respectively. Since every MaxPQ can be simulated by a MinPQ, for convenience we only present MinPQs, and if no further explanation is given, all the forthcoming priority queues are MinPQs. The three basic operations of a (min) priority queue, say H, are described as follows.

- INSERT(H, u) either inserts the element u into H or updates the key of u if u is already in H.
- MINIMUM(H) indicates an element with the minimum key. If H is a heap—a special kind of priority queue mentioned in the next subsection—the operation is also denoted as $H.top()$.
- EXTRACTMIN(H) returns an element with the minimum key and removes it from the queue.

2.4.2 Heaps

A **heap data structure** is an efficient implementation of a priority queue. It is usually implemented in an array where each element is assigned to a position in the array according to certain rules.

For instance, a (min) **binary heap** is a heap data structure, which can be visualized as a binary tree, such that the key of a node is not greater than the keys of its children (if exist). The element with the minimum key is assigned to the root node of the tree. Figure 2.3 shows the array and the visualization of a binary heap with 7 elements. The element at the k^{th} position of the array is the parent of the two elements at $(2k)^{th}$ and $(2k+1)^{th}$ positions where $1 \leq k \leq 3$. Heaps have all the basic operations of a priority queue, i.e. inserting new element, indicating and extracting an element with the minimum key. Depending on how these basic operations are implemented, heaps can be categorized into different types. Table 2.2 shows a review of three popular types of heaps with respect to the time complexities of the basic operations and to their space complexities in

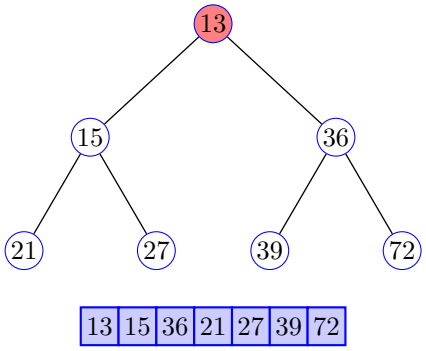


Figure 2.3: An example of a binary heap.

Table 2.2: A review of three popular heaps.

Name	Finding min	Extracting min	Inserting
Binary heap	$O(1)$	$O(\log n)$	$O(\log n)$
Binomial heap	$O(\log n)$	$O(\log n)$	$O(\log n)$
Fibonacci heap	$O(1)$	$O(\log n)$	$O(1)$

implementation, where n is the number of elements. The table indicates that the Fibonacci heap is the best one in terms of time complexity of the inserting operation. However, the binary heap is most popular heap data structure used in applications because of its simpleness. Since heaps are the most efficient implementations of priority queues, it is common that priority queues imply heaps.

2.5 Optimization Problems

In this section a basic introduction of optimization problems (OPs) is given in four subsections. In Subsection 2.5.1 we give notations and mathematical formulations of OPs. The next two subsections focus on constrained continuous optimization problems, that are mentioned in Chapter 5 to prove the existence of an equilibrium of traffic assignment models. Subsection 2.5.2 is about the first-order optimality conditions for both constrained and unconstrained problems, while Subsection 2.5.3 presents the Frank-Wolfe algorithm—a well known algorithm for solving large-scale OPs. Subsection 2.5.4 introduces combinatorial optimization—an area in discrete optimization with a large number of applications.

2.5.1 Introduction

An **optimization problem** (OP) is to minimize or maximize a given objective function over a set of variables. In order to make a problem clear and well defined, a mathematical formulation is investigated. We use the following notations for the forthcoming OPs.

- $x = (x_1, x_2, \ldots, x_n)$ is the **variable vector**.
- $z : \mathbb{R}^n \to \mathbb{R}$ is the **objective function** to be minimized or maximized of a problem.
- $g_i : \mathbb{R}^n \to \mathbb{R}$ and $h_j : \mathbb{R}^n \to \mathbb{R}$ are **constraint functions**, that define the constraints of the problem. These constraints are normally given as $g_i(x) \geq 0$ or $h_j(x) = 0$. The set of vectors, that satisfy all the constraints, is the **feasible region** of the problem.

The general mathematical formulation of an OP is

$$\begin{aligned} &\text{Minimize } z(x) \\ &\text{Subject to } g_i(x) \geq 0 \qquad \forall i \in I \\ &\qquad\qquad h_j(x) = 0 \qquad \forall j \in J, \end{aligned}$$

where I and J are the finite sets of indices of inequality constraints and equality constraints, respectively. Since a maximization problem can be transformed into a minimization problem by replacing the objective function $z(x)$ by $-z(x)$, for convenience we only use the minimization formulation to describe optimization problems. Problems whose variables vary continuously in $\mathbb{R}$, i.e. $x \in \mathbb{R}^n$, are called **continuous optimization problems**. However there are also problems whose variables are in sets of non-continuous values, such as a set of integers, a set of natural numbers, or a set of permutations of events, etc. Such problems belong to the concept of **discrete optimization**. Especially, if all the variables of a discrete optimization problem receive only integer values, i.e. $x \in \mathbb{Z}^n$, the problem is called **integer programming problem** (IP).
When the objective function and all the constraint functions of a problem are linear, the problem is called **linear programming problem** or **linear program** (LP) for short. The canonical formulation of an LP is as follows.

$$\begin{aligned} &\text{Minimize } c^T x \\ &\text{Subject to } Ax \leq b \\ &\qquad\qquad x \geq 0, \end{aligned}$$

where $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$, and $b \in \mathbb{R}^m$.
Given an OP over a feasible region ω, a vector $x \in \omega$ is a **local solution** of the OP if there exists a neighborhood E of x, such that $f(x) \leq f(y)$ for all $y \in \omega \cap E$. An OP may have one or many local solutions. If $z(x) \leq z(y)$ for all $y \in \omega$, then x is a **global solution** of the problem.
A function z, defined on a convex set of points ω, is **convex** if

$$z\left(\alpha x_1 + (1-\alpha)x_2\right) \leq \alpha z(x_1) + (1-\alpha)z(x_2) \quad \text{for all } 0 \leq \alpha \leq 1, \text{ and } x_1, x_2 \in \omega.$$

Minimization problems where the objective function is convex and the feasible region is a convex set, are called **convex minimization problems** (CMP). An important property of a CMP is that any local solution must be a global solution. Figure 2.4 (a) shows an example of a minimization problem with non-convex objective function on the range $[0, 5]$ having two local solutions y_1 and y_2. In Fig. 2.4 (b), a convex minimization problem has only one local solution x_2 and it is also the global solution.

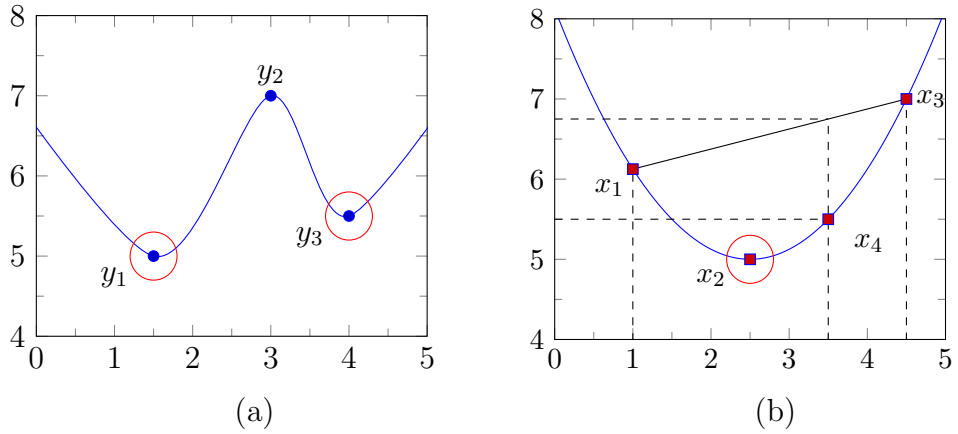


Figure 2.4: Examples (a) minimization problem with two local solutions and (b) a convex minimization problem with the unique local solution.

2.5.2 The First-Order Optimality Conditions

Given an OP problem without constraints, i.e. minimize $z(x)$ where $x \in \mathbb{R}^n$, the most well known optimality conditions are the **first-order conditions** which state that if the function z has a local minimum at x^* then the gradient of z with respect to x equals zero at x^*, i.e.

$$\nabla_x z(x^*) = \left(\frac{\partial z(x^*)}{\partial x_1}, \frac{\partial z(x^*)}{\partial x_2}, \ldots, \frac{\partial z(x^*)}{\partial x_n} \right) = 0.$$

The first-order conditions are more complicated when constraints are added to the problem, i.e.

$$\begin{aligned} &\text{Minimize } z(x) && \\ &\text{Subject to } g_i(x) \geq 0 && \forall i \in I \\ &\qquad\quad h_j(x) = 0 && \forall j \in J. \end{aligned}$$

The first-order conditions for constrained optimization problems are well known as **Karush-Kuhn-Tucker (KKT) conditions**, that were discovered by W. Karush and published by H.W. Kuhn and A.W. Tucker. The KKT conditions state that if all the functions g_i and h_j are continuously differentiable at a local minimum point x^*, and a number of regularity conditions (mentioned below) are satisfied then there exist μ_i and λ_j, such that

$$\sum_{i \in I} \mu_i \nabla g_i(x^*) + \sum_{j \in J} \lambda_j \nabla h_j(x^*) = \nabla z(x^*) \qquad \text{(KKT)}$$

$$\mu_i g_i(x^*) = 0, \qquad \forall i \in I \qquad (2.1)$$

$$\mu_i \geq 0, \qquad \forall i \in I. \qquad (2.2)$$

Constraints (2.1) are called **complementary slackness**, and μ_i are called **dual variables**. A number of **regularity conditions** have been proposed, two of them are described as follows.

Linearity constraints: when $g_i(x)$ and $h_j(x)$ are **affine functions**, i.e., they are composed of a linear function and a constant, e.g. $g(x) = Ax + c$.

Linear independence: At x^* the gradients of the functions h_i and those of the active constraints g_i at x^*, i.e. $g_i(x^*) = 0$, are linearly independent.

When $g_i(x)$ and $h_j(x)$ are linear functions and the objective function $z(x)$ is strictly convex, the problem is called a **linear constrained convex minimization** (LCCM) problem. In an LCCM problem, the first mentioned regularity condition, i.e. the linearity constraints, is satisfied, therefore if the KKT conditions are satisfied at a point, then that point is a global minimum of the problem. The general formulation of an LCCM problem is described as the following program.

$$\text{Minimize } z(x) \tag{GLCC}$$

$$\text{Subject to } \sum_{i=1}^{n} l_{ij} x_i = b_j \qquad \forall j \in J \tag{2.3}$$

$$x_i \geq 0 \qquad \forall i = 1, 2, \ldots, n. \tag{2.4}$$

If x^* is the minimum point of program GLCC, applying the KKT conditions at x^* with $g_i(x) = x_i$ and $h_j(x) = \sum_{i=1}^{n} l_{ij} x_i - b_j$ we have

$$\mu_i + \sum_{j \in J} (\lambda_j l_{ij}) = \frac{\partial z(x^*)}{\partial x_i} \qquad \forall i = 1, 2, \ldots, n$$

$$\mu_i x_i^* = 0 \qquad \forall i = 1, 2, \ldots, n$$

$$\mu_i \geq 0 \qquad \forall i = 1, 2, \ldots, n.$$

These equalities and inequalities can be rewitten as

$$x_i^* \left(\frac{\partial z(x^*)}{\partial x_i} - \sum_{j \in J} (\lambda_j l_{ij}) \right) = 0 \qquad \forall i = 1, 2, \ldots, n, \tag{2.5}$$

$$\frac{\partial z(x^*)}{\partial x_i} - \sum_{j \in J} (\lambda_j l_{ij}) \geq 0 \qquad \forall i = 1, 2, \ldots, n. \tag{2.6}$$

In summary, at the global minimum point of program GLCC, constraints (2.3), (2.4), and the KKT conditions (2.5), (2.6) are satisfied.

2.5.3 The Frank-Wolfe Algorithm

In the previous subsections LCCM problems and the general formulation GLCC for them, as well as the KKT conditions have been presented, this subsection introduces a well known algorithm proposed by Frank and Wolfe [27]. The algorithm is known under different names, such as the **Frank-Wolfe algorithm**, the conditional gradient algorithm, the reduced gradient algorithm, and the convex combination algorithm. The Frank-Wolfe algorithm can be used to solve a general constrained convex minimization problem, however, in this thesis we only consider LCCM problems formulated as GLCC.

The idea of the algorithm is to find a better feasible point in a neighborhood of the current feasible point. After a number of iterations, a sequence of feasible points approaching the global solution is obtained. Let us assume that the current feasible point is x_1. In order to determine a better feasible point x_2 in a neighborhood of x_1, a reducing direction is investigated by using Taylor's expansion. Suppose that y is a point in the neighborhood of x_1, according to Taylor's expansion we have

$$z(y) \approx z(x_1) + \nabla z(x_1)(y - x_1).$$

This means that

$$z(y) \approx \nabla z(x_1)y + z(x_1) - \nabla z(x_1)x_1.$$

Because $z(x_1) - \nabla z(x_1)x_1$ is constant, a better solution in the neighborhood of x_1 can be obtained by solving the minimum program with respect to the variable y as follows.

$$\text{Minimize} \quad f(y) = \nabla z(x_1)y, \tag{2.7}$$

where y satisfies constraints (2.3) and (2.4) of GLCC. Since 2.7 is a linear program, it can be easily solved by an existing algorithm, e.g. the simplex algorithm. Let y_1 be the optimal solution of 2.7. The vector $d = y_1 - x_1$ is a **reducing direction**. Because all the constraints of GLCC are linear and x_1, y_1 are feasible points, $x_1 + \alpha d$ for any $\alpha \in [0, 1]$ is also a feasible point of GLCC. This leads to a subproblem, named **one-dimensional minimization** problem as follows

$$\begin{aligned} \text{Minimize} \quad & z(x_1 + \alpha d) \\ \text{Subject to} \quad & \alpha \in [0, 1]. \end{aligned} \tag{2.8}$$

There are a number of efficient algorithms for 2.8, such as golden section, Fibonacci search, and polynomial interpolation methods. If α^* is the optimal solution of 2.8, then $x_2 = x_1 + d\alpha^*$ is the best feasible solution in the neighborhood of x_1. Repeating the same steps will return a sequence of efficient solutions $\{x_1, x_2, \ldots, x_k\}$ approaching to the global solution x^* of GLCC.

Moreover, according to Taylor's expansion we have

$$z(x^*) \approx z(x_k) + \nabla z(x_k)(x^* - x_k) + 0.5\nabla^2 z(x_k)(x^* - x_k)^2. \tag{2.9}$$

Because z is a convex function, we have $\nabla^2 z(x_k)(x^* - x_k)^2 \geq 0$. This and (2.9) imply that

$$z(x^*) \geq z(x_k) + \nabla z(x_k)(x^* - x_k). \tag{2.10}$$

Since y_k is the minimum of $\nabla z(x_k)y$, $\nabla z(x_k)x^* \geq \nabla z(x_k)y_k$ holds. From (2.10) we have

$$z(x^*) \geq z(x_k) + \nabla z(x_k)(y_k - x_k).$$

This means $z(x_k) + \nabla z(x_k)(y_k - x_k)$ is a lower bound on $z(x^*)$. It holds that

$$z(x_k) \geq z(x^*) \geq z(x_k) + \nabla z(x_k)(y_k - x_k) \quad \text{for all } k \geq 1. \tag{2.11}$$

Inequalities (2.11) imply that the Frank-Wolfe algorithm can be stopped at the iteration k if $\nabla z(x_k)(y_k - x_k)$ is less than a given value.

In summary, the FWA starts from a feasible point, say x_0, and determines a sequence of feasible points, that approach the global solution. If x_i is the current feasible point where $0 \leq i$, then the next feasible point, i.e. x_{i+1} is determined by two major steps as follows.

- **Step 1** (reduction direction): solving the program 2.7 and get the optimal solution y_i, i.e., $y_i = \arg\min \nabla z(x_i)y$. The vector $d = y_i - x_i$ is a reducing direction.

- **Step 2** (one-dimension search): investigating the optimal solution α^* of the program 2.8, i.e., $\alpha^* = \arg \min_{0 \leq \alpha \leq 1} z(x_i + \alpha d)$. The next feasible point is determined as $x_{i+1} = x_i + d\alpha^*$.

2.5.4 Combinatorial Optimization

Combinatorial optimization is an area in discrete optimization where an optimal solution must be identified from a finite set of solutions. Combinatorial optimization covers a wide range of real problems, such as routing, network design, the traveling salesman, and facility location problems. Most of combinatorial optimization problems can be formulated as graph problems, that can normally be well defined as integer programs (IP) and be solved by an IP solver.
An example of combinatorial optimization problem is the **shortest path** (SP) problem, that is to find a possible path with the minimum length on a given graph $G(V, E)$ from a given source node s to a given destination node t. The SP problem

can be formulated as an IP as follows

$$\text{Minimize} \sum_{(i,j)\in E} w_{ij}x_{ij} \tag{2.12}$$

$$\text{Subject to} \sum_j x_{ij} - \sum_k x_{ki} = \begin{cases} 1 & \text{if } i = s \\ -1 & \text{if } i = t \\ 0 & \text{otherwise} \end{cases} \quad \forall i \in V \tag{2.13}$$

$$x_{ij} \in \{0, 1\} \qquad \text{for all } (i, j) \in E, \tag{2.14}$$

where w_{ij} is the length of the link (i, j) and the each variables x_{ij} receive either the value 1, if (i, j) is on the determined shortest path from s to t, or 0 in other cases. Constraints (2.13) are so-called the **flow constraints**. The optimal solution of the SP problem can be found by solving its IP formulation using an IP solver, however, there are also dedicated efficient algorithms for the problem, such as Dijkstra's algorithm and A-Star algorithm. The details of these algorithms will be discussed in Chapter 3.

2.6 Traffic Assignment Modeling

Traffic assignment modeling (TAM) is the problem of forecasting traffic flows on the links (roads) of a given transportation network with given travel demands between each pair of population zones. It plays a key role in urban traffic planning. In this section we introduce shortly basic knowledge of TAM and it will be investigated deeply in Chapter 5. In Subsection 2.6.1 a number of definitions and terminologies for traffic systems are presented including a review of traveling time functions. Subsection 2.6.2 gives an overview of popular traffic assignment models with an emphasis on the user equilibrium model, that is the most well known model being used in traffic planning.

2.6.1 Introduction

The **traffic flow** on a link corresponds to the average number of vehicles passing a certain point on the link within a specified time interval, e.g. 12 cars pass the entrance point of the link within every 10 seconds. A **free flow** on a link is a possible flow when vehicles do not interact with each other, such that they can run as freely as without other vehicles. Traffic flows are measured in a **flow unit**, that usually corresponds to one vehicle of the most common kind of vehicles in the traffic system. For instance, in a traffic system containing mostly cars, the traffic flow unit should be the **passenger car unit** (PCU). Similarly, the **motorcycle unit** (MCU) should be the flow unit for traffic systems containing mostly motorcycles. In the

case of **mixed traffic systems** (MTS), i.e. traffic systems consist of various kinds of vehicles traveling together without dedicated lanes for each kind, it is necessary to estimate the **equivalent value** of each kind of vehicle in the flow unit corresponding to a given kind of vehicle. For instance, in an MTS consisting of cars and motorcycles where MCU is chosen to be the flow unit, in order to estimate the flows on the links, the equivalent value of a car in MCU must be investigated, e.g. one car equals 3.67 MCU. The equivalent value of a vehicle is normally depended on the physical characteristics of the vehicle, such as the size, the speed, the acceleration, and the safety distance to others.

The **link capacity** of a link is the maximum traffic flow on the link, such that some conditions are satisfied. For instance, the **steady capacity** of a link is the maximum steady-state flow on the link, i.e., the capacity of the point on the link with the minimum capacity. This point is usually the end point (at an intersection) or a bottle-neck point on the link. The **practical capacity** of a link is defined as the maximum traffic flow, that can go through the link, such that there is no dense traffic or congestion.

The traveling time on a link depends on many factors, e.g. the capacity of the link, the traffic flow, and geographical shape of the link, etc. Given a link, the relationship between traveling time and the traffic flow on the link is called under different names, e.g. **traveling time function**, travel cost function, and speed-flow equation. Branston [8] gave a good review of traveling time functions by 1976 summarized shortly in Table 2.3 with an addition of the **conical volume-delay function** proposed by Spiess [62]. Each of proposed functions is based on some certain observations and data. Thus, they may be accurate for estimating traveling time on some specific kinds of links where the data was collected and the observations are made, but may not really accurate on links with different characteristics.

Table 2.3: Overview of a number of popular traveling time functions.

Type	**Authors**	**Comment**
N-line	Irwin, Dodd, Cube [41]	Simple, but hard to identify without data
Curvilinear	Smock [61], Soltman Overgaard [57]	Expensive in computation
Logarithmic exponential	Mosher [54]	Not suitable for iterative assignments
BPR	BPR (USA) [9] Steenbrink [63]	Simple, easily and quickly integrable. Suitable for UE models
Conical	Spiess [62]	Simple, easily and quickly integrable. Suitable for UE models

The function, that is most accepted in general, is the **BPR function**, which has been developed by the Bureau of Public Roads (USA) [9]—one of the former sections of the Federal Highway Administration. The BPR function computes the traveling time on a link as

$$t(x) = T_0\Big(1 + \rho\,(\frac{x}{C_p})^{\beta}\Big), \tag{2.15}$$

where ρ, β are the parameters suggested by BPR engineers to be 0.15 and 4 without explanation, T_0 is the traveling time at free flow, and C_p is the practical capacity of the link.
In TAM, each of traffic assignment (TA) model has its own assumptions about routing behaviors of drivers or about transportation networks. The principles proposed by Wardrop [67] are widely used in TAM as basic traffic assumptions.

Principle 2.6.1. *(**Wandrop's first principle**) The journey times in all routes actually used are equal and less than those which would be experienced by a single vehicle on any unused route. Each user non-cooperatively seeks to minimize his cost of transportation.*

Principle 2.6.2. *(**Wandrop's second principle**) At equilibrium the average journey time is minimum. This implies that each user behaves cooperatively in choosing his own route to ensure the most efficient use of the whole system.*

2.6.2 A Review on Popular TA Models

In the literature of traffic planning, many TA models have been proposed. Each is based on different certain assumptions. A number of models can be applied to various kinds of traffic systems, some others are developed for a specific traffic system. In this subsection some popular TA models, that are widely used in traffic planning, are briefly summarized.
One of the simplest models is the **all-or-nothing** (AON) model, which is based on the assumption that drivers choose a shortest length path to travel without considering other factors. This assumption is reasonable in sparse and uncongested transportation networks where the traveling time on a link is approximately proportional to the distance of the link. However, in traffic systems where traffic congestion occurs frequently, a shortest path in terms of length may take more traveling time than other paths with less traffic flow do. In such traffic systems, the AON model does not describe exactly the routing behavior of drivers. The AON model can be solved by assigning all the traffic demands of each **origin-destination** (O-D) pair to a shortest path connecting them.
In the **incremental model**, the traffic demands are assigned on separated steps. On each step, a part of the traffic demands is assigned to the network based on the AON model. However, drivers in each step choose the best path in term of traveling

time instead of length of the path. This means drivers choose the path with shortest traveling time according to the current traffic flows on the links in the current step. After each step, the traveling time on each link is updated according to the current traffic flow on the link. The more traffic demands are assigned on a path, the more traveling time it takes on that path, thus drivers would not choose a path with a large traffic flow. This model does not yield an equilibrium, since it depends on the order of O-D pairs to be assigned, and also on the amount of demand to assign in each step. Thus, it is difficult to evaluate this model, however, the incremental model is closer to the real routing behavior than the AON model since it takes the traffic flows on the links into account while the AON does not.

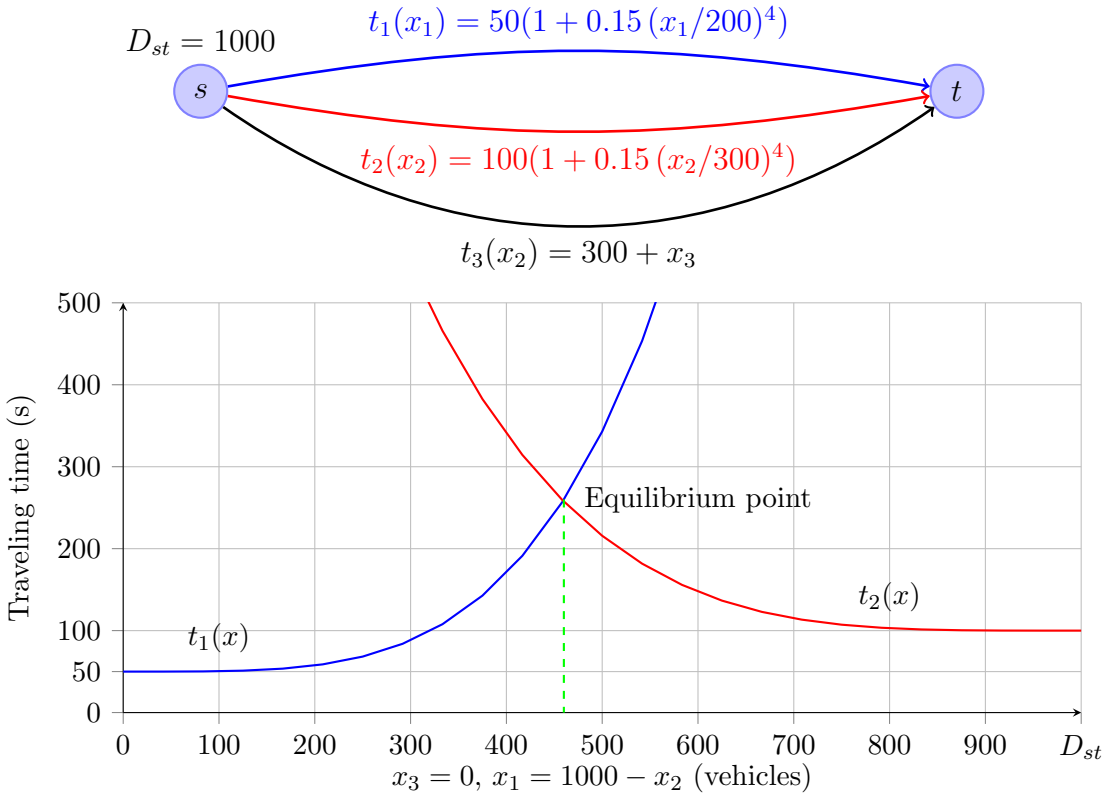


Figure 2.5: An example of the user equilibrium model.

Traffic assignment models satisfying the first principle of Wardrop are called **user equilibrium** (UE) models since they approach to an equilibrium by drivers, i.e., all drivers travel on a shortest traveling time path and they are not willing to change their paths. This assumption is close to the real traffic routing behavior, thus it is used widely in modern TA models. There are a number of models, developed from the UE model, such as the stochastic user equilibrium model and the dynamic user equilibrium model.

Figure 2.5 shows an example of the UE model on a transportation network containing only cars. There are three possible paths from s to t with the traveling time $t_1(x_1)$,

$t_2(x_2)$, and $t_3(x_3)$ where x_1, x_2, and x_3 are the number of cars on the first, the second, and the third paths, respectively. The number of traffic demands between s and t is $D_{st} = 1000$ PCU, i.e. there are in average 1000 cars traveling from the origin s to the destination t every hour. Drivers choose one of the three possible paths to travel, such that their traveling time is minimum. The optimal solution is $x_1 \approx 460$ PCU, $x_2 \approx 540$ PCU, and $x_3 = 0$ PCU. The traveling times on the first path and second path are equal (around 260 seconds) while the traveling time on the third path is 300 seconds. No drivers choose the third path to travel since its traveling time is larger than those of other paths even when its traffic flow is free. Because all the drivers (either on the first path or on the second path) have to spend the same amount of traveling time from s to t and that is less than those on the third path, they are not willing to change their path to another possible path. This means the system has an equilibrium according to the drivers.

Traffic assignment models satisfying the second principle of Wardrop are called **system optimal** (SO) models. All SO models reach to a system equilibrium when all the drivers cooperate with each other, so that the total traveling time of all the drivers in the system is minimum. This assumption does not reflect the real traffic behaviors in many cases, however, it can be applied to traffic systems where drivers are willing to follow instructions of a traffic controlling center. The model is also used to estimate a lower bound on the total traveling time of drivers on the system assigned by other models.

Chapter 3

Shortest Path Problem

This chapter presents the classical shortest path (SP) problem, that has a wide range of applications in both research and real-world problems. Various algorithms for the SP problem have been proposed. Among them, Dijkstra's and A-Star algorithms are most investigated due not only to their efficiency but also to their ability to be implemented from two directions. Algorithms for the SP problem mentioned in this chapter are also reused frequently in a number of algorithms for the k shortest paths problem presented in Chapter 4.
The chapter is divided into 6 sections. Section 3.1 gives a general introduction on the SP problem including a review of popular algorithms for it. Section 3.2 and Section 3.3 present algorithms following the one-directional and the bidirectional approaches, respectively. A comparison between the implementations of various algorithms on real city maps is also presented in Section 3.4. Section 3.5 introduces routing methods on large maps and the last section, i.e. Section 3.6, are conclusions.

3.1 Introduction

Given a graph $G(V, E)$, the SP problem is to find a path in G with the shortest length (weight) from a given source node s to a given destination node t, i.e.

$$\begin{aligned} \text{Minimize} \quad & W(p) \\ \text{Subject to} \quad & p \in P_{st}, \end{aligned}$$

where P_{st} is the set of all possible paths from s to t and $W(p)$ is the length of the path p. As mentioned in Subsection 2.5.4, the SP problem can be formulated as an IP and solved by an IP solver, however, in this thesis we investigate some dedicated algorithms for it.
If the graph is connected, then there is always a finite shortest path between any pair of distinct nodes. This is stated in Theorem 3.1.1.

Theorem 3.1.1. *Given a connected graph $G(V, E)$, there is a finite shortest path from a source node s to a destination node t if and only if there are no negative cycles in the graph G.*

Proof. Firstly, we prove that if there is a finite shortest path from s to t, say p^*, then there are no negative cycles in G. Indeed, for a proof by contradiction, we assume that there is a negative cycle $c = (v_1, v_2, \ldots, v_l, v_1)$, i.e. $W(c) < 0$. Because G is connected, there is a path from s to v_1, say p_1, and a path from v_1 to t, say p_2. The path $p = p_1 \oplus p_2$ is also a possible path from s to t, thus $W(p) \geq W(p^*)$. Let

$$k = \left\lceil \frac{W(p) - W(p^*)}{|W(c)|} \right\rceil,$$

where $\lceil x \rceil$ gives the next bigger integer of x. The path defined as

$$\bar{p} = p_1 \oplus (c\oplus)^h \oplus p_2$$

is a path from s to t including $h > k$ times the cycle c. We have

$$W(\bar{p}) = W(p_1) + W(p_2) + kW(c) < W(p^*).$$

We can create a sequence of paths from s to t with decreasing lengths, and all of those are less than the length of p^*. This conflicts with the assumption that p^* is the path with smallest weight. Hence, there is no negative cycle in G.
Secondly, if there are no negative cycles in G, we prove that the k^{th} shortest path is finite, where $k \geq 1$. Indeed, if there are no cycles in G, then the number of nodes of any path from s to t does not exceed $|V|$. This means that all the possible paths are finite. If G contains (positive) cycles, let $c = (v_1, v_2, \ldots, v_l, v_1)$ is a cycle in G, then we have a list of k finite paths $D = \{q_i = p_1 \oplus (c\oplus)^i \oplus p_2 \mid i = 1, 2, \ldots, k\}$ where p_1 is a path from s to v_1 and p_2 is a path from v_1 to t. We have

$$W(q_1) < W(q_2) < \ldots < W(q_k),$$

thus the length of the k^{th} shortest path, say p_k^*, should not exceed $W(q_k)$. If p_k^* is infinite, then there must be a (positive) cycle, say $\bar{c}$ appear on p^k unlimited number of times. This leads to $W(p_k^*) = +\infty$. This conflicts with the statement that the length of p_k^* does not exceed $W(q_k)$. □

Principle 3.1.2. *(**Optimality Principle**) If v is a node on a shortest path p from s to t, then the subpath of p from v to t is a shortest path from v to t. In other words, a subpath of a shortest path is also a shortest subpath.*

Proof. We have $p = RootP \oplus SubP_1$ where $RootP$ and $SubP_1$ are the subpaths of p from s to v and from v to t, respectively. For a proof by contradiction, we assume

that there exist a path $SubP_2 \in P_{vt}$, such that $W(SubP_2) < W(SubP_1)$. Then the path $\bar{p} = RootP \oplus SubP_2$ is also a path from s to t and

$$W(\bar{p}) = W(RootP) + W(SubP_2) < W(RootP) + W(SubP_1) = W(p).$$

This means p is not a shortest path from s to t. □

In the lectures series of H. Bast [3] a number of popular algorithms were introduced, while in [70] Zhan gave a review on the implementations of algorithms on real transportation networks. Table 3.1 summaries a number of existing algorithms for the SP problem, where $|V|$ is the number of nodes, $|E|$ is the number of links and N (in the scaling algorithm proposed by Gabow) is the largest graph parameter, that is said to be smaller than $|V|$, see [29] for more details.

Table 3.1: Review of popular algorithms for the shortest path problem.

Algorithm name	**Complexity**	**Author, Year**
Bellman-Ford-Moore	$O(\|V\|\|E\|)$	Bellman [6], 1956; Moore [53],1959; Ford [26], 1962
	$O(\|V\|^2 \log \|V\|)$	Dantzig 1958, Dantzig 1960, Minty (Pollack & Wiebenson 1960), Whiting & Hillier 1960
Floyd-Warshall	$O(\|V\|^3)$	Roy 1959; Floyd [25] 1962, Warshall [68] 1962
Dijkstra with list	$O(\|V\|^2)$	Dijkstra [17] 1959
A-Star	$O(\|V\|)$	Hart [37] 1968
Threshold algorithm	$O(\|V\|\|E\|)$	Glover et al. [32] 1984
Dijkstra with Fibonacci heap	$O(\|E\| + \|V\| \log \|V\|)$	Fredman & Tarjan [28] 1987
Scaling algorithm	$O(\|E\| \log_{1+\|E\|/\|V\|} \|V\|)$	Gabow [29] 1983
Topological ordering	$O(\|V\|\|E\|)$	Goldberg and Radzik [35] 1993

3.2 One-Directional Search

The **one-directional search** is an approach, searching from the source node (or destination node), and then expanding the visited area around the source (or destination) node until the destination (or source) node is visited. In this section, we present the two most well known algorithms following this approach, namely Dijkstra's algorithm and A-Star algorithm. These algorithms cover a wide range of applications in both academic research and real-world problems. An interesting

characteristic of these algorithms is that they can be implemented from one direction or from both directions. The bidirectional implementations of them are introduced later in the next section.

3.2.1 Dijkstra's Algorithm

Dijkstra's algorithm was proposed by Dijkstra [17] in 1959 and has been improved by various researchers. It is one of the best exact algorithms for the SP problem in terms of running time and is also used as the standard algorithm for evaluating other algorithms for the SP problem.
The idea of the algorithm is to update the distances from the source node to other nodes via their adjacent nodes using the labeling technique. The reached nodes are labeled as "visiting nodes" and stored in a priority queue with respect to their current distances from the source node. Let us denote $g(s, v)$, or $g(v)$ for short if there is no misunderstanding, is the current distance from the source node s to the node v. In each iteration the node, say u, with the minimum current distance in the priority queue is extracted from the queue and labeled as "visited node". The distance of a node, say v, adjacent to u is updated if it is greater than the sum of the the distance of u and the length of the link (u, v), i.e. if $g(v) > g(u) + w(u, v)$. The algorithm terminates when the destination node is visited or when the priority queue is empty. If the priority queue is empty while the destination node is not yet visited, then the destination node is not connected to the source node.

Figure 3.1 illustrates Dijkstra's algorithm to find the shortest path from the node v_1 to the node v_2. In the initial step, the current distances from v_1 to other nodes are set to infinite, but the distance from v_1 to v_1 is set to 0. The source node v_1 is then added to the priority queue H. In Step 1, the node v_1 is extracted from the list H and marked as visited node (in red). Then, the distances of its adjacent nodes, i.e. v_3, v_4, and v_5, are updated. Because $g(v_1, v_3) = \infty > g(v_1, v_1) + w(v_1, v_3) = 3$, the current distance from v_1 to v_3 is updated to 3, i.e. $g(v_1, v_3) = 3$. The node v_3 is then marked as visiting node (in light blue) and added to the priority queue. Repeating the updating procedure for the node v_4 and v_5 we have $g(v_1, v_4) = 3$, $g(v_1, v_5) = 1$. After Step 1, the priority queue H has three visiting nodes v_3, v_4, and v_5 with their current distances from v_1 as follows: $g(v_1, v_3) = 3$, $g(v_1, v_4) = 3$, and $g(v_1, v_5) = 1$. The node v_5 is the node with minimum distance from the source node v_1.
In Step 2, since the visiting node on the priority queue with the smallest current distance is v_5, it is extracted from the priority queue and marked as visited node. The remaining steps, i.e. Step 3, Step 4, and Step 5, repeat the updating procedure as in Step 1. The algorithm terminates at Step 5 when the destination node v_2 is visited. The shortest path tree from v_1 to all visited nodes is marked in red. Based on this tree, we can track backward to find the full shortest path from the source node v_1 to not only the destination node v_2 but all the visited nodes. For example,

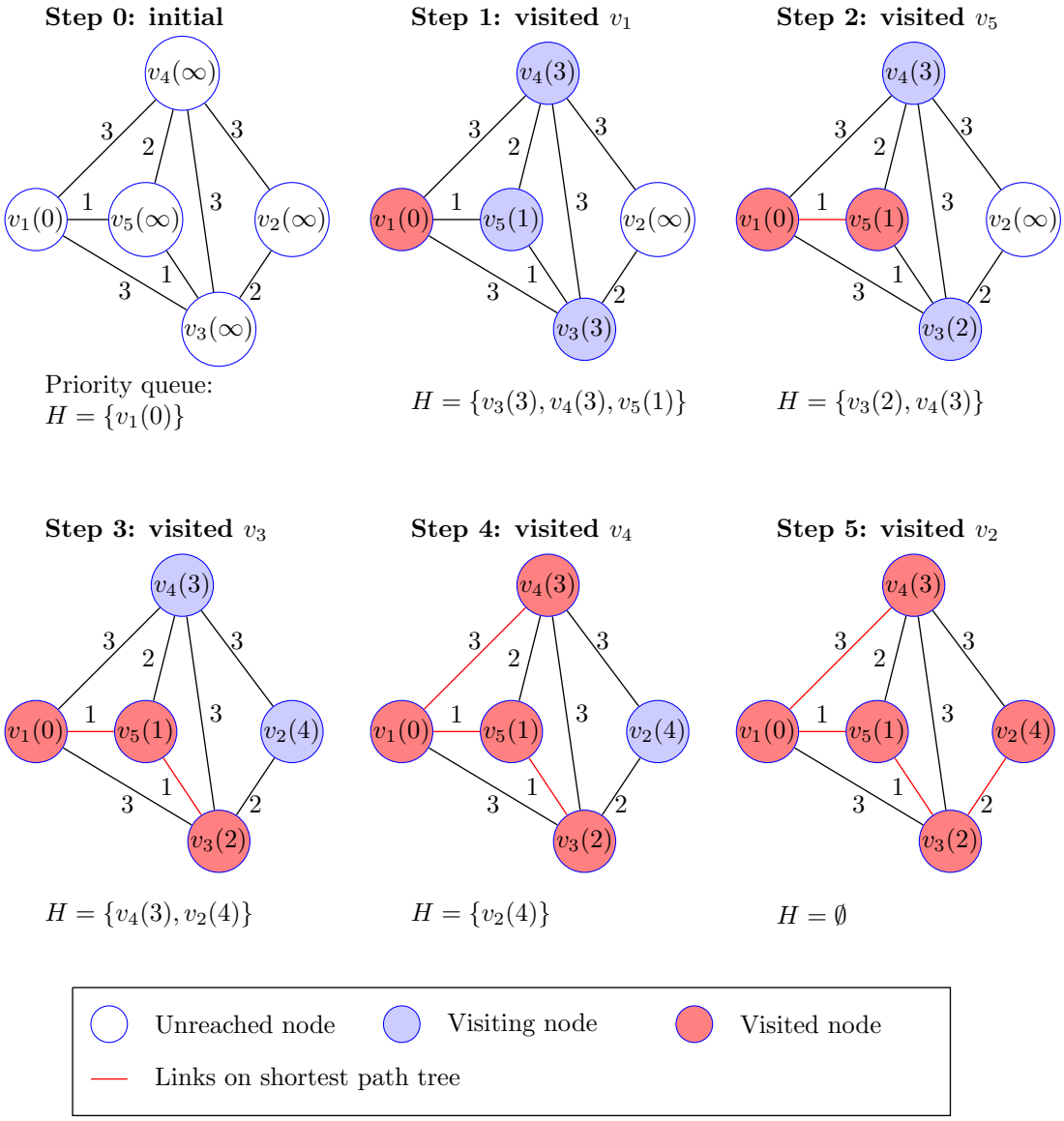


Figure 3.1: Illustration of Dijkstra's algorithm.

the shortest path from v_1 to v_2 is (v_1, v_5, v_3, v_2), the shortest path from v_1 to v_3 is (v_1, v_5, v_3).

Dijkstra's algorithm can also be used to find shortest paths from one source node to many destination nodes on a given graph. In this case, the algorithm terminates when the priority queue is empty or when all the destination nodes are visited. Dijkstra's algorithm from a source node s to a destination node t is described in Alg. 3.1 where H is a priority queue of nodes with respect to the current distances from the source node. Recall that the function $\textsc{Insert}(H, v)$ either adds the node s into the priority queue H if s is not on H or updates the position of node v (already in H) as its distance changed. The function $\textsc{ExtractMin}(H)$ returns the element in H with smallest current distance, and remove it from H.
The previous node of the node v on the shortest path from the source node to v is stored in $Pre(v)$. By tracking backward the previous nodes of the visited nodes on the graph, the shortest path from the source node to any visited node can be explicitly recovered. This tracking procedure, namely $\textsc{RecoverPath}(s, t, Pre[])$, is expressed in Alg. 3.2.

The time complexity of Dijkstra's algorithm depends significantly on the data structure for graph representing and on the kind of the priority queue of visiting nodes. The best known complexity of Dijkstra's algorithm, archived by using Fibonacci heap, is $O(|E| + |V| \log |V|)$, see [28]. The advantages of Dijkstra's algorithm are its simpleness and its easy implementation, therefore it is used widely in both academic research and real-world applications. It is also known as the representative algorithm in the labeling approach for the shortest path problem. However, the algorithm has an advantage that it have to visit all the nodes in the neighborhood of the source node including nodes locating further from the destination node than the source node does.

Algorithm 3.1 Dijkstra's algorithm for the one-to-one SP problem.

```
1: procedure DIJKSTRA(G(V, E), s, t)
2:     S ← ∅                                          ▷ Set of visited nodes
3:     H ← ∅                                          ▷ Priority queue of nodes regarding g()
4:     g(v) ← +∞, Pre(v) ← NULL for all v ∈ V
5:     g(s) ← 0
6:     INSERT(H, s)
7:     while H ≠ ∅ do
8:         u ← EXTRACTMIN(H)                          ▷ Take out the smallest element
9:         if u = t then
10:            Break                                  ▷ Stop loop
11:        end if
12:        S ← S ∪ {u}
13:        for all v ∈ V s.t. v ∉ S, and (u, v) ∈ E do   ▷ Adjacent nodes of u
14:            if g(v) > g(u) + w(u, v) then
15:                g(v) ← g(u) + w(u, v)
16:                Pre(v) ← u
17:                INSERT(H, v)
18:            end if
19:        end for
20:    end while
21:    Return RECOVERPATH(s, t, Pre[])
22: end procedure
```

Algorithm 3.2 Recover shortest path from the array of previous nodes.

```
1: procedure RECOVERPATH(s, t, Pre[])
2:     p ← ∅                                          ▷ Empty path
3:     if Pre[t] = NULL then                          ▷ Two nodes are not connected
4:         Return p                                   ▷ Return empty path
5:     end if
6:     i ← t
7:     while i ≠ s do
8:         p ← {(Pre[i], i)} ⊕ p
9:         i ← Pre[i]
10:    end while
11:    Return p
12: end procedure
```

3.2.2 A-Star Algorithm

In 1968 Hart et al. [37] proposed an algorithm, named **A-Star** (or A^*), that is a generalized version of Dijkstra's algorithm. The common idea of the two algorithms is the labeling technique and updating current distances from the source node s to the adjacent nodes of the visited node. Both of the algorithms terminate when the destination node t is visited or when the queue of visiting nodes is empty. The difference between the algorithms is in the order of nodes to visit. In each iteration, Dijkstra's algorithm chooses a node v to visit with respect to only the current distance from the source node s to v, i.e. $g(s, v)$. The algorithm does not consider the potential distance from v to the destination node t. Thus, some nodes staying very far from the destination node may be visited before some closer nodes. This is said to be the disadvantage of Dijkstra's algorithm. In order to overcome that difficulty, A-Star algorithm requires a **potential function**, denoted $h(v, t)$ or $h(v)$ for short, to evaluate the potential distance from a node v to the destination node t. The function returns a lower bound on the shortest distance from a node to the destination node. The potential function is useful for knowing whether a node is close to or far away from the destination node. A popular lower bound of the shortest length between two nodes is the Euclidean distance—the length of the straight line between the nodes. The Euclidean is surely a lower bound on the shortest length of a path from v to t. Note that the value of the potential function at the destination node must be zero, i.e. $h(t, t) = 0$.
A-Star algorithm chooses nodes to visit according to the sum of the current distance from the source node s and the lower bound distance to the destination node, i.e. nodes are visited orderly according to the sum $f(v) = g(s, v) + h(v, t)$. By choosing nodes to visit with respect to the value of $f(v)$, A-Star algorithm may not need to visit nodes, that are very far from the destination node, thus the destination node is possibly reached after visiting fewer nodes than those of Dijkstra's algorithm.

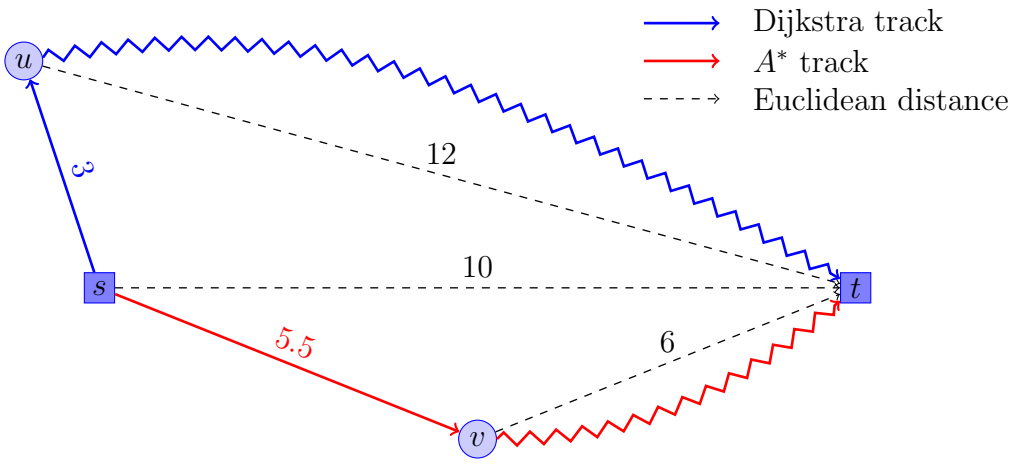


Figure 3.2: Tracking nodes in A-Star algorithm and in Dijkstra's algorithm.

Figure 3.2 shows an example of the difference in the order of nodes to visit between Dijkstra's algorithm and A-Star algorithm using Euclidean distance as a lower bound on the length of the shortest length of a path connecting two nodes. Both of the algorithms start at the source node s and then update the distance from s to the adjacent nodes u and v, i.e. $g(s,u) = 3$ and $g(s,v) = 5.5$. Because $g(s,u) < g(s,v)$, Dijkstra's algorithm chooses the node u to be the next visited node. However, it can easily be seen that the node u is further from the destination node than s. In other words, the direction from s to u goes away from t, and Dijkstra's algorithm cannot take this fact into account. A-Star algorithm considers also the Euclidean distance from all the nodes to the destination node, i.e. $h(u,t) = 12$ and $h(v,t) = 6$. We have $f(u) = g(s,u) + h(u,t) = 15$ and $f(v) = g(s,v) + h(v,t) = 11.5$, thus the node v is chosen to be the next visited node, and it is actually closer to t than u.

Algorithm 3.3 A-Star algorithm for the one-to-one SP problem.

```
1: procedure ASTAR(G(V,E), s, t, h(v,t))
2:     S ← ∅                                   ▷ Set of visited nodes
3:     H ← ∅                                   ▷ Priority queue of nodes regarding f()
4:     for v ∈ V do
5:         g(v) ← +∞, Pre(v) ← NULL
6:         f(v) ← +∞
7:     end for
8:     g(s) ← 0, f(s) ← h(s,t)
9:     INSERT(H, s)
10:    while H ≠ ∅ do
11:        u ← EXTRACTMIN(H)                   ▷ Take out the smallest element
12:        if u = t then
13:            Break                           ▷ Stop loop
14:        end if
15:        S ← S ∪ {u}
16:        for all v ∈ V s. t. v ∉ S and (u,v) ∈ E do
17:            if f(v) > f(u) − h(u,t) + w(u,v) + h(v,t) then
18:                f(v) ← f(u) − h(u,t) + w(u,v) + h(v,t)
19:                Pre(v) ← u
20:                INSERT(H, v)
21:            end if
22:        end for
23:    end while
24:    Return RECOVERPATH(s, t, Pre[])
25: end procedure
```

A-Star algorithm to find a shortest path from a source node s to a destination node t is described in Alg. 3.3, where $h(v,t)$ is a potential function and other denotations are the same as those in Alg. 3.1.

It can be easily seen that A-Star algorithm considers more "the future", whereas Dijkstra's algorithm only focuses on the present. A-Star algorithm is obviously the same as Dijkstra's algorithm when its potential function is equivalent to zero, i.e. $h(x,t) = 0$. Therefore, in theory the complexity of A-Star algorithm equals those of Dijkstra's algorithm. However, its average running time for querying a number of pairs of nodes is normally better than those of Dijkstra's algorithm. The running time of A-Star algorithm depends considerably on the potential function $h(x)$, i.e. the better lower bound the potential function $h(x,t)$ can give, the faster the algorithm reaches to the destination node. According to this feature, various algorithms based on A-Star algorithm have been developed by applying different potential functions, e.g., routing services in real transportation networks using landmark technique or shortcut technique.

3.3 Bidirectional Search

A number of algorithms, e.g. Dijkstra's algorithm and A-Star algorithm, visit the source node (or destination node) first and then enlarge the visited area until the destination node (or source node) is reached. In other words, those algorithms only search from one direction and do nothing on the other direction. The idea of the **bidirectional search** approach is to search in both directions simultaneously, i.e. from the source node, called **forward search**, and from the destination node, called **backward search**. When the sets of the visited nodes in the both directions meet each other and a number of stopping conditions are satisfied, the bidirectional search terminates. Algorithms following the bidirectional search have their own stopping conditions.
For instance, the bidirectional version of Dijkstra's algorithm, so-called **bidirectional Dijkstra**, terminates when there exists a node visited from both searching directions. The stopping conditions of bidirectional search using A-Star algorithm, namely **bidirectional A-Star algorithm**, is more complex since having a visited node in both directions does not guarantee that the shortest path is found. The stopping conditions of bidirectional A-Star are introduced in [34].

Figure 3.3 shows an illustration of the bidirectional Dijkstra to find a shortest path from the source node s to the destination node t. The green ball illustrates the visited area in the forward search, while the red ball illustrates the visited area in the backward search. The balls are enlarged simultaneously until they meet each other at the node v_4, i.e. v_4 is visited in both directions. The shortest path from s to t is the combination of the shortest path from s to v_4, i.e. $p_1 = (s, v_{11}, v_4)$, and the shortest path from v_4 to t, i.e. $p_2 = (v_4, t)$. In other words, the shortest path from s to t is

$$p = p_1 \oplus p_2 = (s, v_{11}, v_4, t).$$

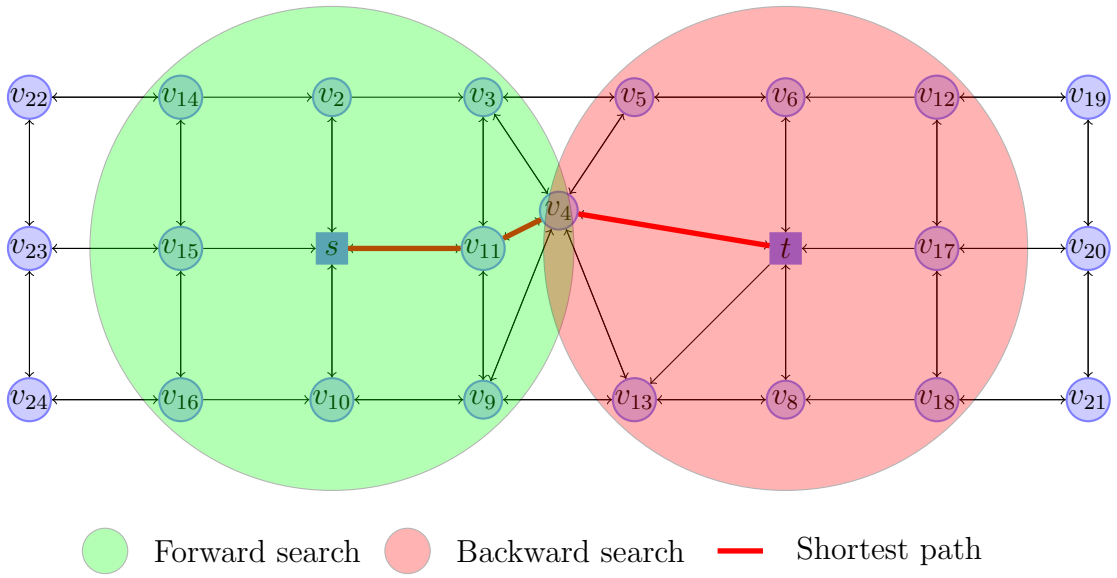


Figure 3.3: An illustration of the bidirectional version of Dijkstra's algorithm.

Note that, in the forward search, the set of adjacent nodes of a node v is defined as $\{y \in V \mid (v, y) \in E\}$, while in the backward search from the destination node, the set of adjacent nodes of v is $\{y \in V \mid (y, v) \in E\}$.
The general structure of algorithms following the bidirectional search is described in Alg. 3.4 where the function RecoverShortestPath() returns the shortest path based on the information from both the searching directions. In each direction there is a set of the visited nodes and a priority queue of the visiting nodes. H_f and S_f are the priority queue of the visiting nodes and the set of the visited nodes in the forward search, respectively. Similarly, H_b and S_b are the priority queue of the visiting nodes and the set of the visited nodes in the backward search, respectively. The stopping conditions are checked outside of the searching directions, however, they can also be checked inside each of the directions. Both the searches can be implemented simultaneously in two independent threads, but they must be able to share some common data to other thread to check the stopping conditions.

Algorithm 3.5 describes the bidirectional Dijkstra where the stopping condition is checked inside each searching direction. The common data of both directions are the sets of visited nodes, i.e. the forward search can access to the set of the visited nodes in the backward search and reverse. In each searching direction, whenever a node is labeled as visited, e.g. node v, it will be checked if it was visited in the other searching direction or not. If v was visited in the other searching direction, the algorithm will terminate. The determined shortest path of the algorithm from the source node s to the destination t is $p = p_1 \oplus p_2$, where p_1 is the determined shortest path from s to v in the forward search and p_2 is the determined shortest path from v to t in the backward search.

Algorithm 3.4 General structure of algorithms following the bidirectional search.

```
1: procedure BidirectionalSearch(G(V, E), s, t)
2:     S_f ← ∅                          ▷ Set of visited nodes in forward search
3:     H_f ← ∅                          ▷ Priority queue in the forward search
4:     S_b ← ∅                          ▷ Set of visited nodes in backward search
5:     H_b ← ∅                          ▷ Priority queue in the backward search
6:     Initializing in the forward search and the backward search
7:     Insert(H_f, s), Insert(H_b, t)
8:     while Stop condition is not satisfied do
   /*Steps in forward search*/
9:         u ← ExtractMin(H_f)
10:        S_f ← S_f ∪ {u}
11:        for all v ∉ S_f and (u, v) ∈ E do
12:            if Value of node v is out of date then
13:                Update value of v
14:                Pre_f(v) ← u
15:                Insert(H_f, v)
16:            end if
17:        end for
   /*Steps in backward search*/
18:        ū ← ExtractMin(H_b)
19:        S_b ← S_b ∪ {ū}
20:        for all v̄ ∉ S_b and (v̄, ū) ∈ E do
21:            if Value of node v̄ is out of date then
22:                Update value of v̄
23:                Pre_b(v̄) ← ū
24:                Insert(H_b, v̄)
25:            end if
26:        end for
27:    end while
28:    Return RecoverShortestPath()
29: end procedure
```

Algorithm 3.5 Bidirectional Dijkstra's algorithm for point-to-point SP problem.

```
1: procedure BIDIJKSTRA(G(V, E), source, dest)
2:     S_f ← ∅, H_f ← ∅
3:     S_b ← ∅, H_b ← ∅
4:     for v ∈ V do
5:         g_f(v) ← +∞                          ▷ Distance from s
6:         g_b(v) ← +∞                          ▷ Distance to t.
7:     end for
8:     INSERT(H_f, s), INSERT(H_b, t)
   /*Steps in forward search*/
9:     while H_f ≠ ∅ do
10:        u ← EXTRACTMIN(H_f)                  ▷ Element with smallest g()
11:        if u ∈ S_b then Stop                 ▷ Visited in both directions
12:        H_f ← H_f \ {u}; S_f ← S_f ∪ {u}
13:        for all v ∉ S_f that (u, v) ∈ E do
14:            if g_f(v) > g_f(u) + w(u, v) then
15:                g_f(v) ← g_f(u) + w(u, v)
16:                Pre_f(v) ← u
17:                INSERT(H_f, v)
18:            end if
19:        end for
20:    end while
   /*Steps in backward search*/
21:    while H_b ≠ ∅ do
22:        ū ← EXTRACTMIN(H_b)
23:        if ū ∈ S_f then Stop                 ▷ Visited in both directions
24:        H_b ← H_b \ {ū}; S_b ← S_b ∪ {ū}
25:        for all v̄ ∉ S_b that (v̄, ū) ∈ E do
26:            if g_b(v̄) > g_b(ū) + w(v̄, ū) then
27:                g_b(v̄) ← g_b(ū) + w(v̄, ū)
28:                Pre_b(v̄) ← ū
29:                INSERT(H_b, v̄)
30:            end if
31:        end for
32:    end while
33:    Return RECOVERSHORTESTPATH()
34: end procedure
```

3.4 Comparison and Discussion

Table 3.2 shows the comparison of the running times of Dijkstra's and A-Star algorithms following both the one-directional and bidirectional search approaches. The column labeled "Maps" indicates the map instances of cities in Germany, Vietnam, USA, Thailand, the Philippines, Taiwan, and Cambodia. The remaining columns are categorized into two groups: one for the running times of Dijkstra's algorithm and the other for A-Star algorithm. In turn, each group has two columns: one labeled "1 Direction" shows the running times of the algorithm following one-directional search approach and the other labeled "Bidirectional" shows the running times of the algorithm following the bidirectional search approach. The last row labeled "Average" are the average running times of the algorithms on all the map instances. The running times are in millisecond.

Table 3.2: Comparison of Dijkstra's algorithm and A-Star algorithm following the one-directional and the bidirectional approaches.

Maps	**Dijkstra** (ms)		**A-Star** (ms)	
	1 Direction	**Bidirectional**	**1 Direction**	**Bidirectional**
HD-DE1k	0.2667	0.5333	0.2000	0.2000
HP-VN2k	0.4000	0.3000	0.1000	0.2000
BH-VN4k	1.6000	0.6000	0.6667	0.3333
NY-USA5k	1.6000	0.9000	1.1000	0.6000
VT-VN5k	0.8000	0.6000	0.3333	0.2667
MH-DE6k	1.2000	0.7333	0.8667	0.7333
DN-VN8k	1.4000	1.0000	0.3333	0.3333
HN-VN9k	1.8000	0.8000	0.4000	0.8667
PP-CB9k	3.0000	1.9333	0.9333	1.2000
MNL-PP12k	3.4000	2.6000	0.8000	1.2667
TP-TW21k	5.8667	3.1333	3.2000	1.7333
BK-TL22k	7.6667	5.3333	2.0667	2.4000
HCM-VN24k	10.8667	7.1333	2.7333	2.8667
Average	**3.0667**	**1.9692**	**1.0564**	**1.0000**

In the experiments, a number of pairs of nodes were randomly selected on each map instance, and they were used to test all the algorithms, i.e., all the algorithms ran on the same O-D pairs of nodes. In implementing A-Star algorithm we used the Euclidean distance between two nodes as the lower bound on the length of the shortest path connecting the nodes. The time for calculating lower bounds is defined as the preprocessing time of A-Star algorithm, that is not mentioned on Table 3.2. The results of the bidirectional search approach can be considerably improved by using the parallel computing technique, however, we did not use the technique. Instead,

each loop in back search is implemented after one loop in forward search. The map instances are approximately ordered by the total number of nodes and links, i.e. the maps at lower rows are larger than the maps at higher rows. For example, the map in the last row, named HCM-VN24k, is the map of Ho Chi Minh City in Vietnam, that has approximately 24 thousands of nodes. The instance named HD-DE1k is a small map of Heidelberg city in Germany with about one thousand nodes.
It can be seen that , in most of the experiments Dijkstra's algorithm following the bidirectional approach has better running times than original algorithm, i.e. following one-directional search, while the bidirectional A-Star, in some cases, has larger running time than those of A-Star algorithm. This is the results from the stopping conditions in the bidirectional A-Star. As we mentioned above, having a node visited in both searching directions does not guarantee that bidirectional A-Star terminates. Thus, the bidirectional A-Star has to take additional time for checking all the stopping conditions. A-Star algorithm generally has better running times than those of Dijkstra's algorithm, in both one-directional and bidirectional approaches. This is reasonable since A-Star algorithm allows more information from the preprocessing to be received, while in Dijkstra's algorithm no preprocessing step is required. The difference increases when the size of the maps increases. The average running time of Dijkstra's algorithm on this map is 10.8667 milliseconds while the average running time of A-Star algorithm is 2.7333 milliseconds (decrease by nearly 75%).
In conclusion, the bidirectional search approach is generally better than the one-directional search approach, in terms of running time, especially when apply to large city maps or in complex maps. The running times of A-Star algorithm is significantly better than those of Dijkstra's algorithm, however, it requires a preprocessing step and a potential function, thus A-Star algorithm is suitable in real applications where the preprocessing step can be done once and the query time should be short.

3.5 Routing in Large Maps

Routing services in real-world applications are increasingly important. While the size of maps can be very large, e.g. millions of nodes, the query times are expected to be very short. For instance, in transport, when drivers update the position continuously during the trip via the Global Positioning System (GPS) signal, they might lose the way in some moments and the router should be able to find out an alternative route immediately. In order to reduce the query times, most algorithms proposed for routing on large maps require preprocessing steps. The steps may take considerable running time varies from several minutes to even several days according to the algorithms. The memory space required in the preprocessing step can be less than a Gigabyte or larger than hundreds Terabytes.

Table 3.3: Review of speed-up techniques on the map of Western Europe. Source: Bast et al. [4].

	Preprocessing		**Queries**	
	Hour	**Gb**	**Avg. scan**	**Time (μs)**
Dijkstra	–	0.4	9300000	2550000.00
Bidirectional Dijkstra	–	0.4	4800000	1350000.00
CRP	1 : 00	0.9	2766	1650.00
Arc Flags	0 : 20	0.6	2646	408.00
CH	0 : 05	0.4	280	110.00
CHASE	0 : 30	0.6	28	5.76
HLC	0 : 50	1.8	–	2.55
TNR	0 : 20	2.5	–	1.25
TNR+AF	1 : 45	5.4	–	0.99
HL	0 : 37	18.8	–	0.56
HL$-\infty$	60 : 00	17.7	–	0.25
Table lookup	145 : 30	1208358.7	–	0.06

In [4], Bast et al. conducted a survey on recent advances in routing algorithms on real transportation networks. Most of them are further developed from the proposed algorithms for the SP problem, however, the data structures and the preprocessing play significantly important roles. Although the preprocessing steps may take considerable time and space, the query times are very short, e.g. milliseconds for routing in the whole world map. Bast et al. have collected a number of recent speedup techniques in routing algorithms for real transportation networks.
Their computational results in the transportation network of Western Europe are showed in Table 3.3. Most of the experiments were run on a single core of an Intel X5680 3.33 GHz CPU. The first column indicates the name of the algorithm or speedup techniques including Dijkstra's algorithm and the bidirectional Dijkstra. The remaining columns are divided into two groups: one labeled "Preprocessing" contains the information of the preparing step, and the other labeled "Queries" contains the information of queries. In turn, each group has two columns. In preprocessing group, there are two columns namely "Hour" and "Gb" indicating preprocessing time in hour and the memory space for storing the information in Gigabyte, respectively. In the queries group, the column labeled "Avg. scan" indicates the average number of scanned nodes for each query, while the column labeled "Time (μs)" indicates the average query time in microsecond (μs). It can be seen that the two versions of Dijkstra's algorithm do not require a preprocessing step, whereas the others take considerable time for that step. However, the average query times of Dijkstra's algorithm and the bidirectional Dijkstra are significantly larger

than those of the other algorithms with the preprocessing step.

Table 3.4: A comparison of some algorithms for routing in San Francisco Bay. Source: Goldberg [33].

	Preprocessing		**Queries**		
	Hour	**Gb**	**Avg. scan**	**Max scan**	**Time (ms)**
Bidirectional Dijkstra	0.0	0.5	10 255 356	27 166 866	7633.9
A-Star + Landmark	1.6	2.3	250381	3584377	393.4
Reaches + Shortcuts	11.3	1.8	14684	24618	17.4
Reaches + Shortcuts + Landmark	12.9	3.6	1595	7450	3.7

In another study, Goldberg [33] also presented a short overview of a number of popular routing algorithms on the map of San Francisco Bay with 330024 nodes and 793681 links. Table 3.4 shows the computational results of the examined algorithms including the maximum number of scanned nodes, i.e. number of visited nodes in the worst case. All implementations are in a personal laptop with 2 GB of RAM and dual-core 2.0 GHz processor. The meanings of columns are the same with those of Table 3.3.

3.6 Conclusions

The shortest path problem is an important problem with a large number of applications in both academic and real-world problems. Various algorithms have been proposed for the problem. Dijkstra's and A-Star algorithms are the two most popular algorithms for the problem in academic research. While Dijkstra's algorithm is popularly used in researching and teaching, the other is used widely in real-world applications. A-Star algorithm is obviously identical to Dijkstra's algorithm when it uses a potential function equal to zero, i.e. $h(x, t) = 0$. Both Dijkstra's algorithm and A-Star algorithm can be implemented in bi-directional search approach, i.e. forward search from the source node and backward search from the destination node, simultaneously.

Our experiments on real maps of cities in some countries show that, the bidirectional versions of Dijkstra's algorithm and A-Star algorithm are more efficient than the original versions following one-directional approach, in terms of running times. A big advantage of the bidirectional approach is that it visits fewer nodes than those in the one-directional approach. Moreover, both searching directions can be implemented simultaneously in two different threads. With a preprocessing step and a suitable potential function, the running times of A-Star algorithm on real maps are generally better than those of Dijkstra's algorithm.

In order to deal with routing on large maps, e.g. maps of countries, continents or the whole world, a number of methods have been proposed based on existing algorithms for the classical shortest path problem. Most of them require a preprocessing step that may take several hours or even many days and a large memory space. Due to the preprocessing step, the query times are cut down dramatically. In addition, there are also a large number of speed up techniques for routing in real maps, e.g. highway hierarchies and goal directed, but in the scope of this thesis, we do not go into details. We focus more on routing algorithms without preprocessing steps that can be applied efficiently on small or medium graphs with thousands of nodes.

Chapter 4

K Shortest Paths Problem

The ***k* shortest paths** (KSP) problem is an extension of the classical shortest path problem. Instead of finding a path with the shortest length, the KSP problem aims at finding the $k \geq 1$ best paths, in terms of one or a given number of objectives, e.g. length, traveling time, and dissimilarity between paths. Although the problem was proposed in the 1960s, it still receives significant attention from various researchers in recent decades, since it plays an increasingly important role in real-world applications, such as transportation logistic, drivers guiding services, and traffic planning. According to the constraints and the objectives, the problem can be categorized into different problems. In this chapter, four of these problems are investigated, namely the k shortest non-loop-less paths (KSNLP), the k shortest loop-less paths (KSLP), the dissimilar shortest loop-less paths (DSLP), and the multi-objective shortest paths (MOSP) problems.

The contributions of this chapter include two new heuristics based on Eppstein's algorithm: one using loop filters (HELF) for the KSLP problem and the other using loop-and-similarity filters (HELSF) for the DSLP problem. The new heuristics are tested and compared with a number of popular algorithms on real transportation networks.

This chapter is divided into 8 sections. In Section 4.1, the background is introduced, while Section 4.2, Section 4.3, and Section 4.4 present three popular approaches for the KSP problem with representative algorithms for each approach. Eppstein's algorithm is emphasized since it is one of the best algorithms in terms of time complexity for the KSP problem. In Section 4.5 the new heuristic, named HELF, for the KSLP problem is presented with the computational results. Based on the HELF another new heuristic, named HELSF, is proposed for the DSLP problem in Section 4.6. Section 4.7 investigates the MOSP problem with a particular case, named the bi-objective shortest paths (BOSP) problem, where only two objectives are considered simultaneously. In the last section, i.e. Section 4.8, conclusions and discussion are given.

4.1 Introduction

In a large number of modern routing applications users may require many alternative paths. For instance, in navigation systems drivers look for the most suitable path according to their objectives. It is a fact that the shortest-length path is always on the list of priorities, however, the shortest-length path, sometimes, takes more traveling time than others because of the crowded traffic flow on it. There may be other drivers taking the air pollution situation and the road safety into their consideration. In this case, finding a set of $k > 1$ best paths, in terms of considering objectives, is meaningful for drivers. According to the constraints and the objectives, the KSP problem can be categorized into some groups of problems as follows.

KSNLP: The ***k* shortest non-loop-less paths** problem aims at finding the first k shortest paths between a given pair of nodes on a given graph. The paths can either contain or not contain loops (cycles). The best running time in theory belongs to the algorithm proposed by Eppstein [20]. The time complexity of Eppstein's algorithm for finding k shortest non-loop-less paths from one source to one destination is $O(|E| + |V| \log |V| + k|V|)$ where $|V|$ and $|E|$ are the number of nodes and links on the graph, respectively.

KSLP: The ***k* shortest loop-less paths** problem is restricted to the condition that the paths must be loop-less, i.e. they do not contain any loops. If two nodes in the given graph are connected, there is certainly at least one loop-less path connecting them, however, the number of loop-less paths may be dominated by the number of paths containing loops. Therefore, finding a shortest loop-less paths should be more difficult than finding non-loop-less paths, theoretically. A well known algorithm for the KSLP problem was proposed by Yen [69]. The time complexity of Yen's algorithm (using Dijkstra's algorithm) is $O(k|V|(|E| + |V| \log |V|))$.

DSLP: The **dissimilar shortest loop-less paths** problem aims at finding a set of k shortest loop-less paths such that each path is dissimilar from each other, i.e, the minimum dissimilarity between two paths is not less than a given value.

MOSP: The **multi-objective shortest paths** problem is the problem of finding a set of k non-dominated paths (defined later) from a source node to a destination node in terms of a given number of objectives.

The first three problems mentioned above, i.e. the KSNLP problem, the KSLP problem, and the DSLP problem, are one-objective problems. They all select paths based on their lengths, however, each of them has different constraints. The KSNLP problem and the KSLP problem have the constraints of either containing loops or not, while the DSLP problem has the constraint of the dissimilarity between

two paths. Even though, they share a number of common characteristics, thus some approaches can be applied to deal with all of them. In the last mentioned problem, i.e. the MOSP problem, various objectives are taken into consideration simultaneously. The approaches for this problem are much different from the those for the one-objective KSP problems.
In the previous chapter, Theorem 3.1.1 states that there exists a finite shortest path between a given connected pairs of node if and only if the graph has no negative cycles. This statement is held for the k^{th} shortest path, i.e. there exists a finite k^{th} shortest path between two connected nodes on a given graph if and only the given graph does not contain any negative cycles, therefore in the rest of this thesis only graphs without negative cycles are in consideration. As in the SP problem, there is also an **optimality principle for the KSP problem**, that is stated in the following principle.

Principle 4.1.1. *Given a connected graph $G(V, E)$, if v is a node on the finite k^{th} shortest path, say p^k, from s to t, then the subpath of p^k from v to t is a h^{th} shortest path from v to t, where $h \leq k$.*

Proof. For a proof by contradiction, we assume that $h > k$, i.e. there are $h-1$ paths, say $q^1, q^2, \dots q^{h-1}$, from v to t that are shorter than the subpath of p^k from v to t, say q^{tail}. Let q^{head} denotes the subpath of p^k from source node s to v, so that $p^k = q^{head} \oplus q^{tail}$. The set $D = \{l^i = q^{head} \oplus q^i \mid i = 1, 2, \dots h-1\}$ consists of $h-1$ paths from s to t, for each $i = 1, 2, \dots h-1$ we have

$$W(l_i) = W(q^{head}) + W(q^i) < W(q^{head}) + W(q^{tail}) = W(q^k).$$

This means that the lengths of $h-1$ paths on D are smaller than the length of p^k, thus p^k cannot be the k^{th} shortest path where $h > k$. □

Table 4.1 gives a review on a number of proposed algorithms for one-objective KSP problems. They can be grouped into three major approaches. The first one is the labeling approach where the extended version of Dijkstra's algorithm is a representative algorithm. The second one is the path-removing approach. The idea of this approach is that the $(h+1)^{th}$ shortest path is a shortest path in the new graph generated from the original graph by removing the first h shortest paths. A representative algorithm following this approach is Martins' algorithm. The last approach is the path-deviating where a number of path candidates are generated from the previously determined path and put into a priority queue with respect to the lengths of the candidates. Algorithms in this approach have to ensure that the next shortest path is one of the candidates being stored in the priority queue. Yen's algorithm and Eppstein's algorithm are representative algorithms following this approach. While the former algorithm of them, i.e. Yen's algorithm, can find loop-less paths, the later one can determine only non-loop-less paths.

Table 4.1: Some works on k shortest paths problems.

Year	Author	Commment
1971	J.Y. Yen [69]	Finding the k shortest loop-less paths. It is one of the most well known algorithm for the KSP problem.
1984	E.Q.V. Martins [48]	An algorithm in paths removing approach. The next shortest path is found after removing previous shortest paths.
1993	A. Aggarwal et al. [1]	Finding a minimum weight K-link path in graphs with Monge property and applications.
1997	D. Eppstein [20]	Finding the k shortest non-loop-less paths. It is a popular algorithm in path-deviating approach for KSP problem.
1999	V.M. Jiménez et al. [43]	Computing the k shortest paths: A new algorithm and an experimental comparison.
1999	M.B. Pascoal et al. [58]	Labeling approach as an extension of Dijkstra to find k shortest paths.
2003	V.M. Jiménez et al. [44]	A lazy version of Eppstein k shortest paths algorithm.
2011	H. Aljazzar et al. [2]	K^*: A heuristic search algorithm for finding the k shortest paths.

4.2 Labeling Approach

As for the shortest path problem (SP), the **labeling approach for the KSP problem** also uses labeling techniques to update the status of nodes. In order to illustrate this approach, we introduce an extension of Dijkstra's algorithm, named Extension-Dijkstra, for the KSNLP problem.
Suppose that a graph $G = (V, E)$, a source node $s \in V$, a destination node $t \in V$, and a natural number $k \geq 1$ are given. Unlike in Dijkstra's algorithm for the SP problem where only the best current path from the source node s to a reached node v is stored, the Extension-Dijkstra stores the current k best paths from s to every reached node.
The pseudo code of Extension-Dijkstra for finding k shortest paths from s to t is described in Alg. 4.1 where D is the set of shortest paths from s to t and H is the priority queue of the generated paths. Each path $p = (v_1, v_2, \ldots, v_l)$ is im-

Algorithm 4.1 An extension of Dijkstra's algorithm for the KSP problem.

```
1: procedure KSP-DIJKSTRA(G(V, E), s, t, k)
2:     D ← ∅                                          ▷ Set of found paths
3:     H ← ∅                                          ▷ Priority queue of paths based on length
4:     Count(v) ← 0 for all v ∈ V,                    ▷ Number of determined paths from s to v
5:     Count(s) ← −1
6:     id ← 1                                         ▷ First path with id = 1
7:     DestNd(id) ← s, RootList(s) ← {id}
8:     W(id) ← 0
9:     INSERT(H, id)
10:    while Count(t) < k and H ≠ ∅ do
11:        p ← EXTRACTMIN(H)                          ▷ Path with the smallest length
12:        u ← DestNd(p)                              ▷ Destination node of the path
13:        Count(u) ← Count(u) + 1
14:        if u = t then
15:            NewPath ← explicit path of p           ▷ Next shortest path
16:            D ← D ∪ NewPath
17:        end if
18:        if Count(u) ≤ k then
19:            for node v adjacent to u do
20:                id ← id + 1                        ▷ New path from s to v
21:                DestNd(id) ← v                     ▷ Destination node of the new path
22:                Root(id) ← p                       ▷ Root path of the new path
23:                W(id) ← W(p) + w(u, v)
24:                RootList(v) ← RootList(v) ∪ {p}
25:                INSERT(H, id)
26:            end for
27:        end if
28:    end while
29:    return D
30: end procedure
```

plicitly expressed by its root path $Root(p) = (v_1, v_2, \ldots, v_{l-1})$ and its destination node $DestNd(p) = v_l$. The root path of paths with only one node is a null path (a path without nodes). The notation $Count(v)$ indicates the number of selected shortest paths from the source node s to node v. According to Principle 4.1.1, in the Extension-Dijkstra algorithm, only the first k shortest paths from source node to each reached node are stored, thus $Count(v) \leq k$ for all $v \in V$. The set of the root paths of all determined paths to v is $RootList(v)$, that is used to recover all the full paths from s to v. Each path generated by the algorithm has an identity number, denoted id, and the length $W(id)$.
In the initial step of Alg. 4.1, the first path, with the identity number $id = 1$, is investigated. It has only the node s and its length is 0. The path is put into the priority queue of the generated paths H. The algorithm terminates either when the number of selected paths from s to the destination node t equals k—when there are at least k possible loop-less paths from s to t—or when the priority queue H is empty, i.e. all possible loop-less paths are determined. At a reached node the next determined path from the source node s to it is always equal or greater than the previously determined path, i.e. paths are generated in non-decreasing order. More details and the proof for the correctness of the algorithm can be found in the work of Pascoal et al. [58].

The advantages of Extension-Dijkstra are its simpleness and easy implementation. In addition, it can be more efficient when dealing with the KSNLP problem from one source to many destinations. However, the algorithm takes considerable time and space to store the generated paths from the source node to all the nodes that have been reached. This disadvantage makes Extension-Dijkstra not efficient to implement on large graphs with thousands of nodes.

4.3 Path-Removing Approach

The **path-removing approach** is based on Principle 4.3.1, in other words, the next shortest path is determined after removing the previously determined paths from the graph, e.g. the second shortest path is determined after removing the first determined shortest path from the graph, the third shortest path is found after removing the first and the second determined shortest paths, etc.

Principle 4.3.1. *Given a graph $G(V, E)$, a source node s, and a destination node t. The k^{th} shortest path from s to t, if exists, is a shortest path in the graph after removing the $k-1$ previously determined shortest paths from s to t.*

Each algorithm, following the path-removing approach, has its own methods for removing paths from a given graph. The general structure of the algorithms following this approach are described in Alg. 4.2 where s is the source node, t is

Algorithm 4.2 The path-removing approach for the KSP problem.

```
1: procedure KSP-RMPaths(G(V, E), s, t, k)
2:     D ← ∅                                          ▷ Array of found paths
3:     for i ← 0, k − 1 do
4:         p ← ShortestPath(G, s, t)                  ▷ Find a shortest path
5:         if p ≠ NULL then
6:             D[i] ← p
7:             RemovePath(G(V, E), p)
8:         else
9:             Stop                                   ▷ All paths found
10:        end if
11:    end for
12:    Return D
13: end procedure
```

the destination node, and k is the number of paths to determine. The function RemovePath$(G(V,E),p)$ removes path p from the graph $G(V,E)$. In each iteration, a shortest path of the current graph is determined by one of the existing algorithms for the SP problem, e.g. Dijkstra's algorithm or A-Star algorithm. If the function ShortestPath() returns an nonempty path, it is added to the set of the determined shortest paths and then removed from the graph. The algorithm terminates when k paths have been found—when k iterations have been implemented—or when no more paths between the source node s to the destination node t are found, i.e. the function ShortestPath() return an empty path. The time complexity of algorithm in this approach is $O(k[O_{SP}+O_{RmP}])$ where O_{SP} is the complexity of the shortest path algorithm used in the algorithm and O_{RmP} is the complexity of the method for removing a path.

A well known algorithm following this approach is proposed by Martins in 1984 [48]. In Martins' algorithm, a certain path is removed from a graph by adding virtual nodes and virtual links, called **virtual objects**, to the graph, as well as removing a number of links from the graph. The **virtual node** of a node v_i, denoted $\bar{v}_i$, has the attributes of its original node v_i, and is linked to v_i. A **virtual link** of a link is created by replacing one or both the nodes of the link by their corresponding virtual nodes, e.g. $(v_1,\bar{v}_2)$ and $(\bar{v}_1,\bar{v}_2)$ are virtual links of the link (v_1,v_2). A virtual link has all the attributes of its original link.

In order to remove a path $p=(v_1,v_2,\ldots,v_l)$ from a graph $G(V,E)$, Martins' algorithm transforms the graph as follows.

- For each intermediate node on p, i.e. v_i where $2\leq i\leq l-1$, create a virtual node $\bar{v}_i$ linked to the node, and add it into the graph, i.e.

$$V=V\cup\{\bar{v}_i \mid 2\leq i\leq l-1\}.$$

- Add the virtual link $(\bar{v}_i, \bar{v}_{i+1})$, where $2 \leq i < l-1$, to the graph, i.e.

$$E = E \cup \{(\bar{v}_i, \bar{v}_{i+1}) \mid 2 \leq i < l-1\}.$$

- For each link not in p coming to an intermediate node of p, say $(u, v_i) \notin p$ where $2 \leq i \leq l-1$, replace it by its virtual link $(u, \bar{v}_i)$, i.e.

$$E = E \setminus \{(u, v_i) \mid 2 \leq i \leq l-1\} \cup \{(u, \bar{v}_i) \mid 2 \leq i \leq l-1\}$$

  where $(u, v_i) \in E$ and $(u, v_j) \notin p$. This step replaces the header of each link not in p coming to an intermediate node on p by its virtual node. After this step, we have

$$E^+(\bar{v}_i) = \{u \in V \mid (u, v_i) \in E \text{ and } u \neq v_{i-1}\} \cup \{\bar{v}_{i-1}\},$$

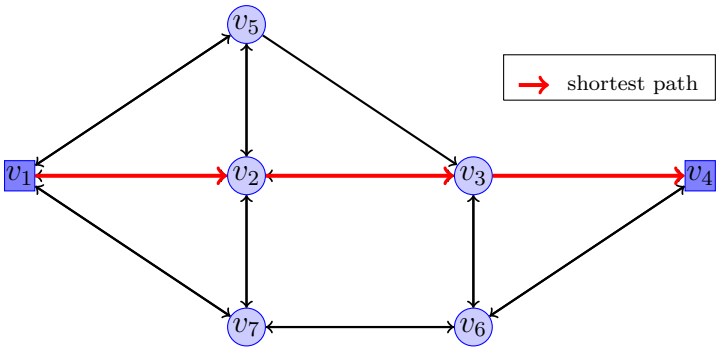
  and $E^+(v_i) = \{v_{i-1}\}$.

- Replace the last link on p by its virtual link i.e.

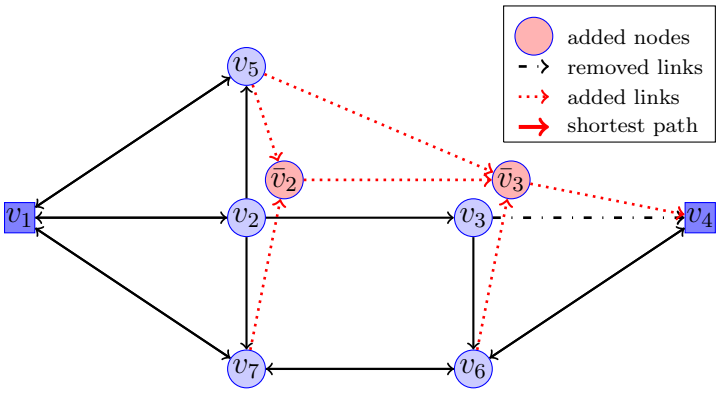
$$E = E \setminus \{(v_{l-1}, v_l)\} \cup \{(\bar{v}_{l-1}, v_l)\}.$$

Each possible path on the new graph refers to a path on the original graph, and each path in the original path except p is linked to a path in the new graph. A path on the original graph can be recovered from its linked path on the new graph by replacing virtual nodes, if exist, by their original nodes.
Figure 4.1 shows an illustration of Martins' algorithm for finding two shortest paths from v_1 to v_4. In Step 1, Fig. 4.1 (a), the first shortest path on the original graph G is $p_1 = (v_1, v_2, v_3, v_4)$. The path p_1 can be determined by any available algorithm for the SP problem. The path is then removed by transforming the graph G in Step 2, Fig. 4.1 (b). In Step 3, the shortest path on the new graph is $\bar{p} = (v_1, v_5, \bar{v}_3, v_4)$. It refers to the second shortest path $p_2 = (v_1, v_5, v_3, v_4)$ on the original graph G.

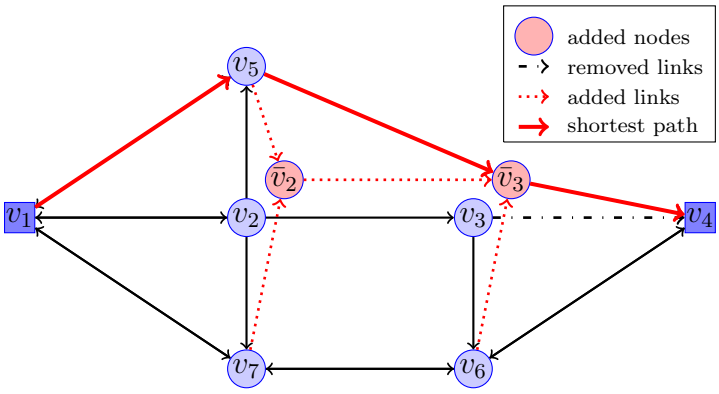
Martins' algorithm for removing the path $p = (v_1, v_2, \ldots, v_l)$ from the graph $G(V, E)$ is described in Alg. 4.3 where the function RemoveLink$(G, (u, v))$ removes the link (u, v) from the graph G, and the function NewLink(u, v, w) creates a new link from u to v with length w. There are two important remarks on the algorithm. First, the paths determined may contain loops, therefore, in order to find loop-less paths, the loop-less condition must be checked for every found path. Second, in every iteration, a number of virtual nodes are added to the graph, this makes the size of the graph bigger. When k is large, the size of the graph after a number of iterations may be significantly larger than those of the original graph. Thus, the running times of the algorithm for finding a shortest path in later iterations are considerably greater than those in the former iterations.



(a) Step 1: Find the shortest path $p_1 = (v_1, v_2, v_3, v_4)$



(b) Step 2: Remove p_1 from $G(V, E)$



(c) Step 3: Find the shortest path $\bar{p}_2 = (v_1, v_5, \bar{v}_3, v_4)$ that refers to the original path $p_2 = (v_1, v_5, v_3, v_4)$

Figure 4.1: An illustration of Martins' algorithm.

Algorithm 4.3 Martins' algorithm for removing a path from a graph.

```
1: procedure MARTIN-REMOVEPATH(G(V, E), p)
2:     if v_{h-1} = s then
3:         REMOVELINK(G, (s, t))
4:         Stop
5:     end if
6:     for all i ← 2, h − 1 do
7:         V = V ∪ {v̄_2} where v̄_i is the virtual node corresponding to the node v_i
8:         for (v, v_i) ∈ E s.t. v ≠ v_{i−1} do
9:             E ← E ∪ {NEWLINK(v, v̄_i, w(v, v_i))}
10:        end for
11:    end for
12:    for all i ← 2, h − 2 do
13:        E ← E ∪ {NEWLINK(v̄_i, v̄_{i+1}, w(v_i, v_{i+1})) }
14:    end for
15:    E ← E ∪ { NEWLINK(v̄_{h−1}, t, w(v_{h−1}, t))}
16:    REMOVELINK(G, (v_{h−1}, t))
17: end procedure
```

Let Max_d be the maximum degree of the graph and L be the maximum number of nodes on the determined paths, then the time complexity for removing a path from the graph is $O(L.Max_d)$. The complexity of Martins' algorithm using Dijkstra's algorithm for finding a shortest path in each iteration is $O(k[L \cdot Max_d + \overline{m} + \bar{n}\log(\bar{n})])$, where $\overline{m} = |E| + kL$ and $\bar{n} = |V| + kL$.

4.4 Path-Deviating Approach

In this section, the **path-deviating approach** for the KSP problem and two well known algorithms following the approach, namely Yen's algorithm and Eppstein's algorithm, are presented. The idea of the path-deviating approach is to create path candidates from the previously determined path. These candidates are inserted into a priority queue regarding their lengths. Each algorithm following this approach has its own method for generating candidates such that the next shortest path is surely one of them.

Algorithm 4.4 describes the general steps of algorithms following the path-deviating approach for finding k shortest paths from the source node s to the destination node t on the graph $G(V, E)$. At the beginning, the shortest path determined by an existing algorithm for the SP problem, e.g. Dijkstra's algorithm or A-Star algorithm, is inserted into the priority queue of path candidates H. In each iteration, the path in the priority queue with minimum length is selected to be the next shortest path. It is also removed from the list, whereas the new candidates generated from it are

Algorithm 4.4 The path-deviating approach for the KSP problem.

```
1: procedure KSP-DevApp(G(V, E), s, t, k)
2:     D ← ∅                                          ▷ Array of found paths
3:     H ← ∅                                  ▷ Priority queue of path candidates
4:     p^0 ← ShortestPath(G, s, t)
5:     if p^0 =NULL then
6:         Stop                                  ▷ Two nodes are not connected
7:     end if
8:     Insert(H, p^0)
9:     for i ← 0, k − 1 do                                      ▷ k iterations
10:        p ← ExtractMin(H)
11:        D[i] ← p
12:        for q ∈ DeviationPaths(G, p, D) do
13:            Insert(H, q)
14:        end for
15:        if |H| = 0 then                                  ▷ All paths found
16:            Stop
17:        end if
18:    end for
19:    Return D
20: end procedure
```

inserted into the queue. Depending on the method for generating candidates, the selected paths may contain loops or not contain any loops.

4.4.1 Yen's Algorithm

In 1971, Yen [69] proposed an algorithm following the path-deviating approach for finding the k shortest loop-less paths. The algorithm is called **Yen's algorithm**. In order to generate candidates from a given path, say $p = (v_1, v_2, \ldots, v_l)$, Yen's algorithm turns at each node on p, i.e. v_i, where $1 \leq i \leq l$. The turning node is called the **spur node**. The candidate deviated from the spur node v_i has two subpaths. The first subpath is the **root path**, denoted as R_i, containing the first $i-1$ links on p, i.e. $R_i = (v_1, v_2, \ldots, v_i)$. The second part, named **spur path**, is a shortest path from v_i to t that is not a subpath of any previously generated paths that have the root path R_i.

In Fig. 4.2, the path $p = (v_1, v_2, v_3, v_4)$ is the shortest path from v_1 to v_4. The path $\bar{p}_1 = (v_1, v_5, v_6, v_4)$ is the deviated path of p turning at v_1. The root path of $\bar{p}_1$ is a null path and the spur path of $\bar{p}_1$ is (v_1, v_5, v_6, v_4). The deviated path of p turning at v_2 is $\bar{p}_2 = (v_1, v_2, v_5, v_6, v_4)$ where $R_2 = (v_1, v_2)$ is the root path, and (v_2, v_5, v_6, v_4) is the spur path. There is no deviated paths of p turning at v_3 or v_4.

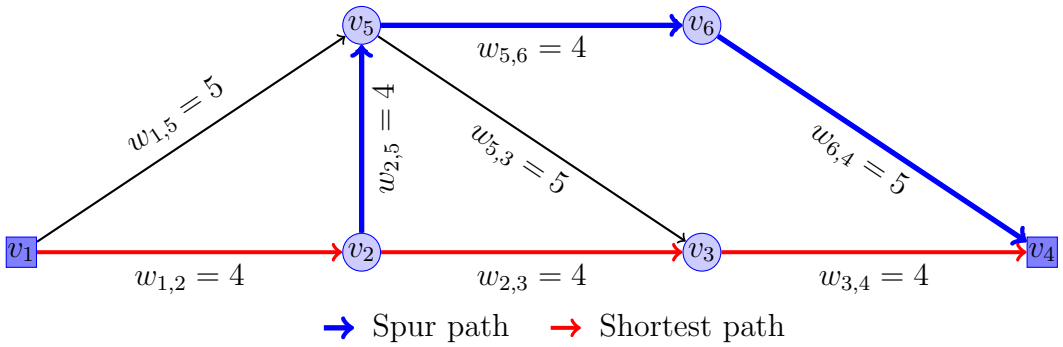


Figure 4.2: An example of a deviated path in Yen's algorithm.

Generating the candidate, turning at the spur node v_i of a path $p = (v_1, v_2, \ldots, v_l)$, follows 5 steps.

- **Step 1:** For each generated path that has the first $i-1$ links as the root path $R_i = (v_1, v_2, \ldots, v_i)$, remove its $(i)^{th}$ link from the graph, e.g. if the path $(v_1, v_2, \ldots, v_i, u_1, u_2, \ldots u_h)$ is already generated, then remove the link (u_1, u_2) from the graph.
- **Step 2:** Remove all the nodes v_j, where $1 \leq j \leq i-1$, from the graph. As a sequence, all the links from/to v_j are also removed.
- **Step 3:** Find a shortest path, say R_i^{spur} from v_i to the destination node t on the new graph. It plays as the spur path of the new candidate.
- **Step 4:** Concatenate the root path and the spur path to generate the new candidate path, i.e. $p_{new} = R_i \oplus R_i^{spur}$.
- **Step 5:** Recover the original graph. This step prepares for generating the next candidate.

Step 1 ensures that the new candidate is not similar to any path that has been generated, thus each path is generated at most once. Whereas, Step 2 makes sure that the spur path of the new candidate does not contain any node of the root path, as a result the new candidate is loop-less. Step 3 and Step 4 create a new loop-less path candidate by combining the root subpath and the determined shortest path from the spur node to the destination node.
Yen's algorithm for generating candidates from a given path p is expressed in Alg. 4.5 where D is the set of all the generated paths. The function SubPath(p, i, j) returns the subgraph between the i^{th} node and the j^{th} node of the path p. The function GetLink(p, j) returns the j^{th} link on the path p, while the function GetNode(p, j) returns the j^{th} node on the path p. The function RemoveLink(G, q) removes the link q from the graph G, while the function RemoveNode(G, u) removes the node u and its related links, i.e. links come to or links from u, from the graph G.

The function RECOVER(G) returns the original graph after removing nodes and links. The algorithm terminates when all the candidates turning from the nodes on the path p have been generated. If no valid candidates have been generated, the algorithm returns an empty set.
One of the greatest advantages of Yen's algorithm is that the determined paths are loop-less. The time complexity of the algorithm depends significantly on the algorithm for finding a shortest path from every spur node to the destination node. If the complexity of the utilized algorithm for finding a shortest path is $O(f(G))$, then the complexity of Yen's algorithm is $O(k|V|f(G))$, e.g. the complexity is $O\big(k|V|(|E||V|\log(|V|)\big)$ when Dijkstra's algorithm is used.

Algorithm 4.5 Yen's algorithm for generating candidates from a path.

```
1: procedure YEN-DEVIATIONPATHS(G(V,E), p, D)
   Note: D is the set of the previously generated paths.
2:     C ← ∅                                   ▷ Set of deviated paths from p
3:     for j ← 1, p.size() do
4:         spurNode ← GETNODE(p, j)
5:         rootPath ← SUBPATH(p, 1, j)
   /*Step 1*/
6:         for all q ∈ D s.t. SUBPATH(q, 1, j) ≡ rootPath do
7:             REMOVELINK(G, GETLINK(q, j))
8:         end for
   /*Step 2*/
9:         for u ← 1, j − 1 do
10:            REMOVENODE(G, GETNODE(rootPath, u))
11:        end for
   /*Step 3*/
12:        spurPath ← SHORTESTPATH(G, spurNode, t)
   /*Step 4*/
13:        if spurPath ≠ NULL then
14:            newPath ← rootPath ⊕ spurPath        ▷ New path from s to t
15:            INSERT(C, newPath)
16:            INSERT(D, newPath)
17:        end if
   /*Step 5*/
18:        RECOVER(G)                           ▷ Recove to the original graph
19:    end for
20:    Return C
21: end procedure
```

4.4.2 Eppstein's Algorithm

In [20] Eppstein proposed an algorithm, following the path-deviating approach, that can determine the k shortest non-loop-less paths in time $O(|E| + |V|lg(|V|) + k)$. The algorithm, called **Eppstein's algorithm** (EA), is one of the most prominent algorithms for the KSP problem in terms of time complexity, however, the paths, determined by the algorithm, may contain loop. In this subsection, we present the original version of Eppstein's algorithm, that is lately used to develop new heuristics for the KSLP problem in Section 4.5 and for the DSLP problem in Section 4.6.

Let T_t be a shortest path tree to the destination t on a given graph $G(V, E)$. The links on T_t are called **basic links**, and the links not in T_t are called **side tracks**. Side tracks with the same source node are **sibling side tracks**. Figure. 4.3 shows an example of a shortest path tree to the node t (or v_4). The basic links on T_t are marked in red, while the side tracks are dotted lines. The two side tracks (v_1, v_5) and (v_1, v_7) are siblings turning from the node v_1.
Note that, there may be a number of shortest path trees to a node on a given graph, however in Eppstein's algorithm only one of them is needed. It is easy to determine such a shortest path tree by Dijkstra's algorithm, mentioned in Chapter 3, from all the nodes to one destination node.

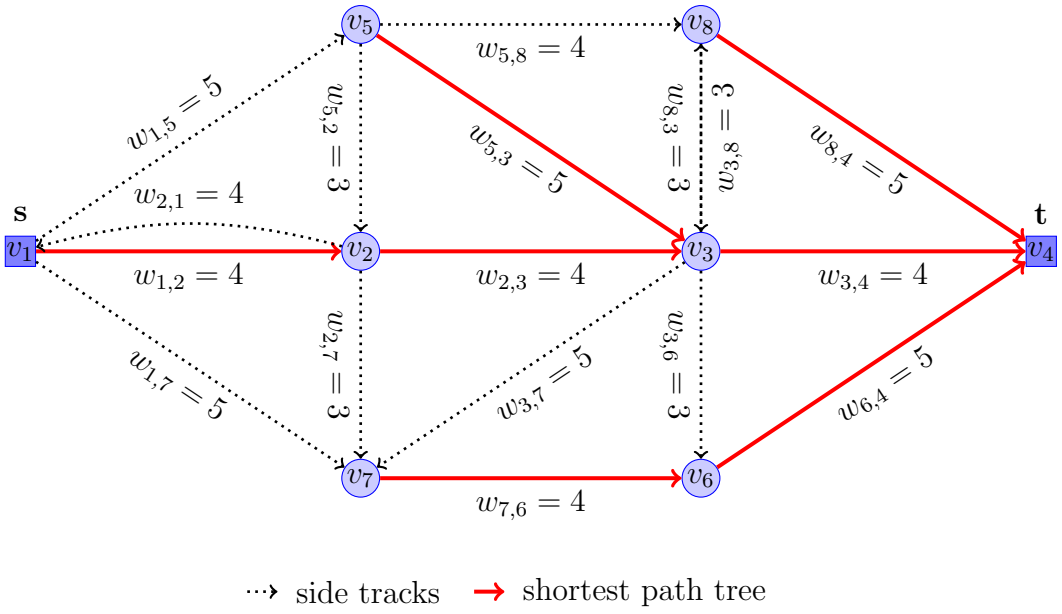


Figure 4.3: An example of a shortest tree to the node t.

Each possible path p from the source node s to the destination node t is a combination of its side tracks and its basic links staying on T_t. The basic links on p connect the side tracks, thus they can be determined if the sequence of the side tracks on p and the shortest path tree T_t are available. This means that path p can

be recovered from the sequence of its side tracks and T_t. For instance, in Fig. 4.3 the path $p = (v_1, v_5, v_3, v_8, v_4)$ has two side tracks $(v_1, v_5), (v_3, v_8)$ and two basic links $(v_5, v_3), (v_8, v_4)$. The first basic link (v_5, v_3) connects the destination node of the first side track to the source node of the second side track. The second basic link connects the destination node of the second side track, i.e. v_8, to the destination node t.

Proposition 4.4.1. *Given a graph $G(V, E)$, a source node s, a destination node t, and a shortest path tree T_t to t. Every possible path from s to t can be uniquely expressed as a sequence of side tracks.*

Proof. Assume that $p = (s = v_1, v_2, \ldots, v_l = t)$ is a path from s to t. The sequence of the side tracks on p is $S = \{S_1, S_2, \ldots, S_h\}$. The function $\mathrm{BP}(u, v)$ returns the unique path connecting u and v on the shortest path tree T_t. For convenience let $l.src$ and $l.dest$ denote the source node and the destination node of a link l, respectively. The path p can be recovered from the sequence of side tracks S as follows.

$$p = BP(s, S_1.src) \oplus S_1 \oplus MiddlePath \oplus BP(S_h.dest, t), \tag{4.1}$$

where $MiddlePath = \bigoplus_{i=1}^{h-1} [BP(S_i.src, S_{i+1}.dest) \oplus S_{i+1}]$. This means p can be implicitly expressed as the sequence of its side tracks and that sequence is unique. In deed, if $\bar{S}$ is a different sequence of side tracks from S. By recovering the full path from $\bar{S}$ as (4.1), we should get a new path different from p. □

Let $d(v)$ be the distance from node v to the destination node, i.e. the shortest length of a path from v to t. For each link $(u, v) \in E$ the **increasing cost** of (u, v) is defined as $\delta(u, v) = w(u, v) + d(v) - d(u)$. In other words, $\delta(u, v)$ is the increasing length of the shortest path from u to the destination via the link (u, v) to the length of a shortest path from u to the destination node that may not go through (u, v). Thus, the increasing cost of a link is always greater or equal to zero, and the increasing cost of a link on the shortest path tree is always zero, i.e.

- $\delta(v, r) \geq 0$ for all $(v, r) \in E$,
- $\delta(v, r) = 0$ if link (v, r) is on the shortest path tree T_t.

Figure 4.4 shows the increasing costs of all the links of the example graph in Fig. 4.3, where t is the destination node and the tree consisting of all the nodes and all the red links is a shortest path tree. The increasing costs of basic links are zero. The increasing cost of the side track (v_5, v_8) is zero but it is not on the shortest path tree since there are two possible paths from v_5 to t with the shortest length: $p_1 = (v_5, v_3, t)$ and $p_2(v_5, v_8, t)$. The increasing costs of other side tracks are greater than zero.

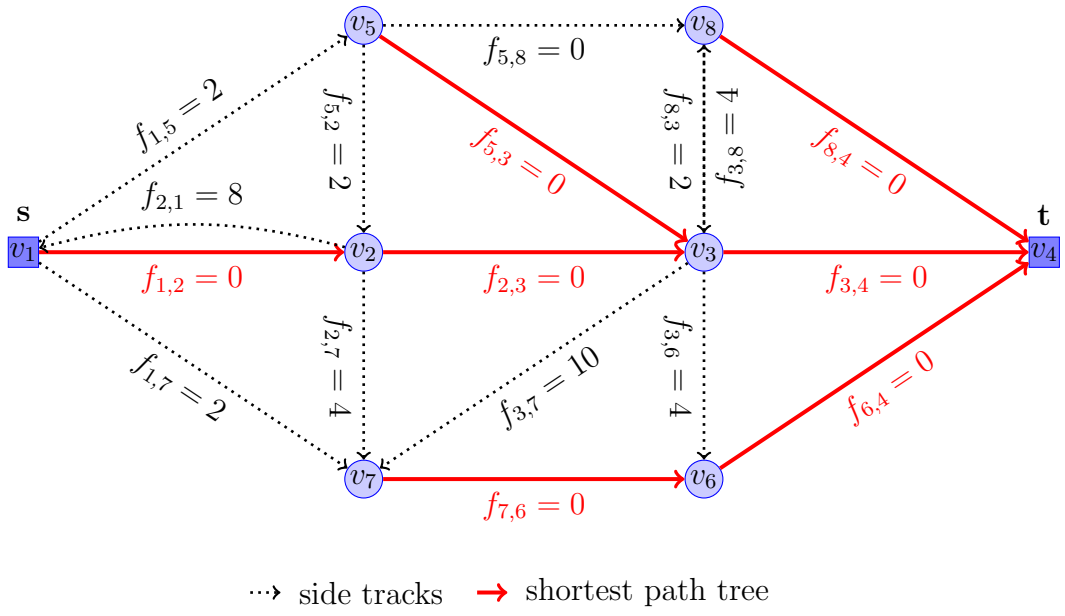


Figure 4.4: The increasing costs of links.

The increasing cost of a path is defined as the sum of the increasing costs of all the links on the path, i.e. $\delta(p) = \sum_{e \in p} \delta(e)$. Since the increasing cost of a basic link is zero, the increasing cost of a path equals the sum of the increasing costs of all the side tracks on the path. The meaning of the increasing cost of a path is that it indicates the greater amount of the path's length to the shortest length. This statement is expressed in the following proposition.

Proposition 4.4.2. *The length of a path equals the sum of the shortest length (length of a shortest path) and the increasing cost of the path.*

Proof. Assume that path $p = (s = v_1, v_2, \ldots, v_l = t)$, we have

$$\begin{aligned}\delta(p) =& \sum_{i=1}^{l-1} \delta(v_i, v_{i+1}) = \sum_{i=1}^{l-1} (w(v_i, v_{i+1}) + d(v_{i+1}) - d(v_i)) \\ =& d(t) - d(s) + \sum_{i=1}^{l-1} w(v_i, v_{i+1}) = d(t) - d(s) + w(p).\end{aligned}$$

Sine $d(t) = 0$, we have $w(p) = \delta(p) + d(s)$ where $d(s)$ is the length of a shortest path from s to t. □

Due to Proposition 4.4.2, the lengths of paths can be compared to each other with respect to their increasing costs. The idea of Eppstein's algorithm is to generate orderly path candidates from a previously determined path regarding their increasing costs. Each candidate generated from a parent path is called a deviated path.

Definition 4.4.3. *Given a path $p = (v_1, v_2, \ldots, v_h)$, a path $q = (u_1, u_2, \ldots, u_l)$ is a **deviated path** of p if there exists $i \in \mathbb{N}$ where $1 \leq i < min\{l, h\}$ such that*

- $v_j = u_j$ *for all* $j = 1, 2, \ldots, i$;
- (u_i, u_{i+1}) *is a side track;*
- SubPath(p, i, h) *is a shortest path from* v_i *to* t;
- SubPath$(q, i+1, l)$ *is a shortest path from* u_{i+1} *to* t.

*The path q is called the deviated path from p via the side track (u_x, u_{x+1}), and p is the **parent path** of q.*

Paths, deviated from a parent path via sibling side tracks, are called **sibling deviated paths**. For instance, in Fig. 4.4, (v_1, v_5) and (v_1, v_7) are two sibling side tracks, thus the paths $p_1 = (v_1, v_5, v_3, v_4)$ and $p_2 = (v_1, v_7, v_6, v_4)$ are two sibling deviated paths of the path $p = (v_1, v_2, v_3, v_4)$ via the sibling side tracks (v_1, v_5) and (v_1, v_7), respectively. The path $p_3 = (v_1, v_5, v_8, v_4)$ is the deviated path of p_1 via the side track (v_5, v_8), however, it is not a deviated path of p.

Lemma 4.4.4. *If q is the deviated path from p via a side track e, then the length of q is equal to the sum of the length of p and the increasing cost of the side track e, i.e. $d(q) = d(p) + \varepsilon(e)$.*

Lemma 4.4.4 can be easily proved by the definitions of side track and deviated path. According to Lemma 4.4.4, the deviated paths from the same parent path can be ordered with respect to the increasing costs of the side tracks that they are deviated via, i.e. the last side tracks of paths. In order to rank the side tracks coming out from the nodes on the path $p = (v_1, v_2, \ldots, v_l)$, at each node v_i where $1 \leq i \leq l$, Eppstein's algorithm (EA) builds a **local heap**, denoted as $LHeap(v_i)$, containing all the sibling side tracks coming out from v_i. In order to compare the deviated paths from different nodes on p, a **global heap** at each node v_i, denoted as $GHeap(v_i)$, is constructed by inserting the head element of the local heap at v_i, i.e. $LHeap(v_i).top()$, to the global heap at the previous node on p, i.e. $GHeap(v_{i-1})$. In other words, the global heap at a node v consists of all the best side tracks coming out from all the nodes on the shortest path from v to the destination node.
For instance, Fig. 4.5 illustrates the local-heaps and the global-heaps of the nodes on the path $p = (s = v_1, v_2, v_3, v_4 = t)$, that is shown in Fig. 4.4. Each of the heaps (local or global) is a binary heap. They are built as following steps. Firstly, the local heaps at v_1, v_2, and v_3 are built. Each of them has a head element that is the side track with the minimum increasing cost coming from the corresponding node. All these head elements are then used to create the global heaps. The global heaps are built backward, i.e. the global heap at the node closest to the destination, i.e.

v_3, is built at first. It contains only the head element of the local heap at v_3, i.e. the side track $e_{3,6}$ with increasing cost 4. The global heap at v_2 is created by inserting the header element of the local heap at v_2 into the global heap at v_3. It contains two side tracks with the same increasing cost 4. Finally, the global heap at v_1 is built by inserting the head element of the local heap at v_1 into the global heap at v_2.

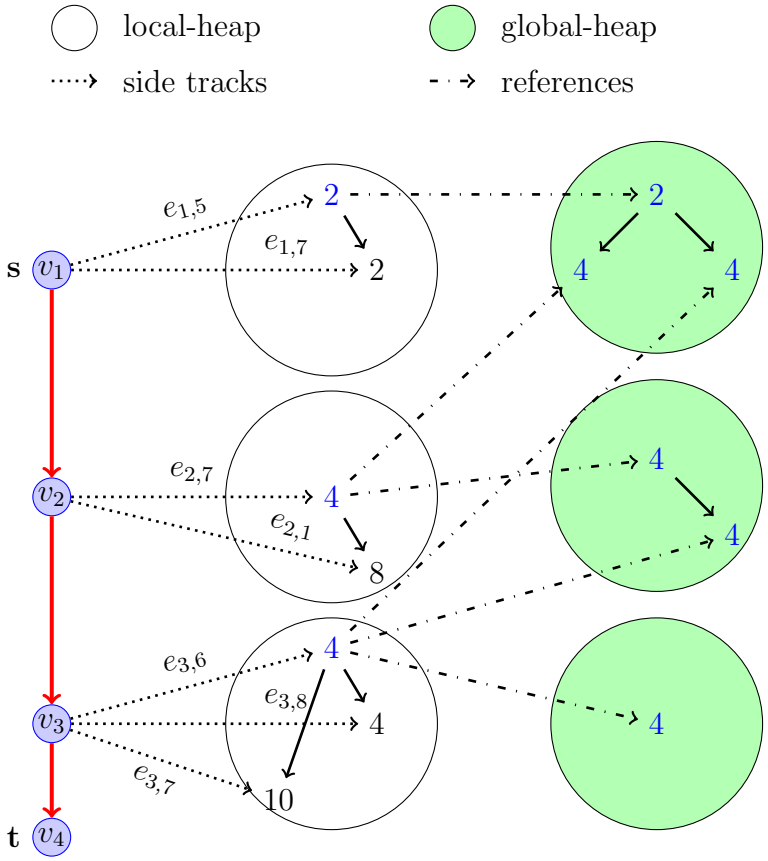


Figure 4.5: Building local and global heaps at nodes on a shortest path.

Eppstein's algorithm follows the general structure in Alg. 4.4 of the path-deviating approach for the KSP problem. However, it is a fact that, each path may have a large number of deviated paths, thus generating all the path candidates deviated from the previously determined path in each iteration could lead to memory leak and slow down the operating speed of the algorithm. In addition, it does not make sense if we generate more than k deviated paths from a parent path, thus in each iteration only a number of the best path candidates are generated. There may be various strategies of selecting the best candidates. They depend on the data structures and also on the characteristics of the graph.

In the original version of EA, the set of potential path candidates from a path, say p, has at most 5 elements. It consists of the best deviated path from p and at most 4 best local and global sibling candidates of p that are not generated. Each sibling candidate of p has the same parent path of p, i.e. they differ from p over the last

side track. The last side tracks of these sibling candidate are children of the last side track of p in the local or global heaps.
Assume that the path p is implicitly expressed by the sequence of its side tracks as $p = (S_1, S_2, \ldots, S_h)$, i.e. p is deviated from its parent $p_{parent} = (S_1, S_2, \ldots, S_{h-1})$ via S_h. The candidates from p are generated in the following steps.

- **Step 1:** generate the **best deviated path of** p by adding the side track with the minimum increasing cost coming out from the nodes on the subpath of p between the destination node of the last side track, i.e. $S_h.dest()$, and the destination node. The added side track is the head element of the global heap at the destination node of the last side track, i.e. $GHeap(S_h.dest()).top()$. The new candidate is

  $$newCandidate = (S_1, S_2, \ldots, S_h, S),$$

  where $S = GHeap(S_h.dest()).top()$.

- **Step 2:** generate the **local sibling candidates of** p from p_{parent}. Each sibling is deviated from p_{parent} via a side track that is a child of S_h on the local heap at $S_h.source()$, i.e.

  $$newCandidate = (S_1, S_2, \ldots, S_{h-1}, S),$$

  where $S \in \textsc{GetChildren}(Lheap(S_h.source()), S_h)$.

- **Step 3:** if S_h is in the global heap at $S_h.source()$, generate the **global sibling candidates** of p from p_{parent} via side tracks that are the children of S_h on the global heap at $S_h.source()$, i.e.

  $$newCandidate = (S_1, S_2, \ldots, S_{h-1}, S),$$

  where $S \in \textsc{GetChildren}(GHeap(S_h.source()), S_h)$.

Eppstein's algorithm (EA) for generating candidates from path p is described in Alg. 4.6 where the function $\textsc{SideTrack}(p)$ returns the sequence of the side tracks of p. The function $\textsc{GetFullPath}(S)$ returns the full path that is implicitly expressed by the sequence of side tracks S. If p is a path on the shortest path tree, i.e. p has no side track, only the best deviated path from p via the head element of the global heap at the source node s is generated. Otherwise, the candidates are generated as three steps mentioned above.
The advantage of EA is to quickly generate deviated paths from a given path. There are a number of improvements for EA, e.g. in [44] Jiménez et al. proposed the lazy version of EA algorithm that can reduce the time for building global heaps. However, EA cannot guarantee that the found paths are loop-less.

Algorithm 4.6 Eppstein's algorithm to generate candidates from a path.

```
1: procedure EPPSTEIN-DEVIATIONPATHS(G(V, E), s, t, p, GHeap[], LHeap[])
2:     C ← ∅                                          ▷ Set of deviated paths from p
3:     if SIDETRACKS(p)= NULL then                    ▷ p is the shortest path
4:         NewCandidate ← GHeap(s).Minimum()          ▷ Path with 1 side track
5:         C ← C ∪ {GETFULLPATH(NewCandidate)}
6:         Stop
7:     end if
8:     (S_1, ..., S_h) ← SIDETRACKS(p)                ▷ Sequence of side tracks presenting p

   /*Step 1: Best candidate deviated from p*/
9:     S_{h+1} ← GHeap(S_h.dest()).Minimum()
10:    NewCandidate ← (S_1, ..., S_h, S_{h+1})
11:    C ← C ∪ {GETFULLPATH(NewCandidate)}
12:    TurnNd ← S_h.source()                          ▷ Turnning node

   /*Step 2: The next siblings of p*/
13:    for all s̄ ∈ GETCHILDREN(LHeap(TurnNd), S_h) do
14:        NewCandidate ← (S_1, ..., S_{h−1}, s̄)
15:        C ← C ∪ {GETFULLPATH(NewCandidate)}
16:    end for

   /*Step 3: Candidates on the global heap*/
17:    if S_h ∈ GHeap(TurnNd) then
18:        for all s̄ ∈ GETCHILDREN(GHeap(TurnNd), S_h) do
19:            NewCandidate ← (S_1, ..., S_{h−1}, s̄)
20:            C ← C ∪ {GETFULLPATH(NewCandidate)}
21:        end for
22:    end if
23:    Return C
24: end procedure
```

4.5 New Heuristics for the KSLP Problem

In this section, we investigate heuristics for the KSLP problem based on algorithms which can find orderly shortest paths but they cannot ensure that the found paths are loop-less, e.g. Martins' algorithm and Eppstein's algorithm. In order to reduce the number of visited paths by Eppstein's algorithm, we investigate loop filters that can determine parent paths whose all the deviated paths contain loops. The new heuristic based on EA using the loop filters for the KSLP problem dominates other examined algorithms in terms of running time on real city maps.

4.5.1 Heuristic Approach

A heuristic is a method for problem solving that does not ensure finding the optimal solution but in many cases it is efficient enough to find a satisfactory solution. For some difficult problems where finding the optimal solution would be impossible or would take a large amount of time, some heuristics can be applied to speed up the process of finding a good-enough solution. One of the most popular heuristic is so-called **trial-and-error**, i.e. trying a feasible solution if it is the optimal solution or it satisfies all the stopping conditions, if neither of them is true, the next feasible solution is examined.
As shown in Section 4.4, among algorithms proposed for the KSP problem, Yen's algorithm is the only one that ensures the found paths are loop-less. The paths determined by the other algorithms, i.e. Martins' algorithm and Eppstein's algorithm, may contain loops. The idea of the trial-and-error heuristic, based on those algorithms, is to check the loop-less condition for every path determined. If a path contains loops, it is removed, otherwise it is selected to be the next shortest loop-less path. The heuristic proceeds further until k loop-less paths are obtained or all possible paths have been visited. Such heuristics using Martins' and Eppstein's algorithms are named the **heuristic based on Martins' algorithm** (HMA) and the **heuristic based on Eppstein's algorithm** (HEA), respectively.

Algorithm 4.7 describes the general structure of the HMA and the HEA for the KSP problem. The structure follows all the steps as those in the original algorithms, i.e. Martins' algorithm an EA, but it sets a limit on the number of iterations to ensure that the heuristics will terminate within a given time without any memory leak. The maximal number of iterations is normally related to k, e.g. $Imax = 1000k$. Some other stopping conditions may be used according to the kind of graphs or the strategies of the heuristics, such as the upper bound on running time or maximum size of the heaps of visited paths, etc. In each iteration, the function NextShortestPath(s, t) returns the next shortest path (may contain loop) determined by the original algorithms. The path is then checked if it is loop-less

or not. It is inserted into the set of found path if it is loop-less, otherwise it is eliminated and the heuristic goes to the next iteration.

Algorithm 4.7 Structure of heuristics for the KSLP problem.

```
1: procedure HEURISTIC-KSLP(G(V, E), s, t, k)
2:     D ← ∅                                  ▷ Set of found shortest loop-less paths
3:     Imax ← maximum number of iterations
4:     i ← 0
5:     while |D| < k and i < Imax do
6:         p ← NEXTSHORTESTPATH(s, t)
7:         if p is loop-less then
8:             D ← D ∪ {p}
9:         end if
10:        i ← i + 1
11:    end while
12:    Return D
13: end procedure
```

Table 4.2: Comparison between Yen's algorithm, the HMA, and the HEA for finding 10 shortest loop-less paths.

Maps	**Yen**			**Martins (HMA)**			**Eppstein (HEA)**		
	Vs	Fs	Time	Vs	Fs	Time	Vs	Fs	Time
HD-De1k	10	10	0.0311	29.27	10	0.0749	437.33	10.00	0.0090
HD-VN2k	10	10	0.0286	29.20	10	0.0672	1486.80	9.80	0.0206
BH-VN4k	10	10	0.1022	34.00	10	0.1775	1648.73	9.40	0.0297
NY-USA5k	10	10	0.1021	15.43	10	0.0664	1437.29	9.43	0.0423
VT-VN5k	10	10	0.1334	36.67	10	0.1889	2895.27	9.27	0.0438
MH-DE6k	10	10	0.1237	29.93	10	0.1885	1411.80	9.40	0.0445
DN-VN8k	10	10	0.1213	20.60	10	0.1415	748.07	9.40	0.0610
HN-VN9k	10	10	0.1394	19.60	10	0.1462	1348.93	9.40	0.0503
PP-CB9k	10	10	0.3411	13.13	10	0.1201	12.33	10.00	0.0645
MNL-PP12k	10	10	0.4783	18.33	10	0.2223	858.73	9.73	0.1000
TP-TW21k	10	10	0.7248	20.25	10	0.4489	20.92	10.00	0.1023
BK-TL22k	10	10	0.9907	25.47	10	0.6782	55.33	10.00	0.1071
HCM-VN24k	10	10	1.7383	17.60	10	0.5347	44.53	10.00	0.2015
Average	**10**	**10**	**0.3888**	**23.81**	**10**	**0.2350**	**954.31**	**9.68**	**0.0674**

Table 4.2 shows the computational results of Yen's algorithm, the HMA, and the HEA for finding 10 shortest loop-less paths on the real maps of various cities. The first column labeled "City maps" indicates the map instances using for the experiments. The rest of the columns are grouped into three blocks: one for Yen's algo-

rithm, one for the HEA, and the other for the HMA. In turn, each block has three columns: one labeled "Vs" shows the average number of paths visited, one labeled "Fs" shows the average number of paths found, and the last one labeled "Time" indicates the average running time. The maximum number of iteration is set to 1000 times k, i.e. $Imax = 10000$. All the experiments were implemented in a personal desktop computer with an Intel (R) dual Core at 2.20 GHz processor and 2 GB of RAM.

It can be seen that the running times of Yen's algorithms are greater than those of the two heuristics, i.e. HMA and HEA. The number of paths visited by the HEA is significantly greater than those by the HMA. In some experiments, the HEA stops before 10 loop-less paths are determined, whereas the HMA is able to determine all the 10 shortest loop-less paths. However, the HMA has better running times than those of the HMA in all the cases. This is reasonable since the HEA is based on Eppstein's algorithm that can generate a large number of shortest non-loop-less paths in a shorter time than those of Martins' algorithm. Other computational results with different values of k can be found in Appendix B.

4.5.2 Loop Filters for Eppstein's Algorithm

As shown in Table 4.2, the HEA dominates the HMA in terms of running time, however, in order to find 10 shortest loop-less paths it has to visits a large number of paths, and in some cases, the maximum number of iterations is reached before 10 shortest loop-less paths are determined. In order to reduce the number of the visited paths, we introduce loop filters that can predetermine paths from which only the candidates with loops are generated.

In Eppstein's algorithm, the sequence of the side tracks of a deviated path is created by adding a side track to the sequence of the side tracks of the parent path. Thus, all the deviated paths have a common subpath of the parent path from the source node to the destination node of the last side track on the parent path. If this common subpath contains a loop, then all the deviated paths contains a loop. Such parent paths are called **bad-parent paths**. The idea of **loop filters** is to predetermine such bad-parent paths. The detected bad-parent paths are removed from the list of considering paths without generating candidates. Thus, the number of path candidates should probably decrease. Theorem 4.5.1 and Theorem 4.5.2 shows two cases of bad-parent paths.

Theorem 4.5.1. *Given a graph $G(V, E)$ and a path $p \in P_{s,t}$—expressed as the sequence of its side tracks $(S_1, S_2, \ldots, S_h)$— if there exists $l \in \{1, 2, \ldots, h-1\}$ such that the subpath p_1 of p from s to the destination node of S_l contains loops, then all the candidates generated from p in Step 2 and Step 3 of Alg. 4.6 also contain loops.*

Proof. Assume that q is a candidate generated from p in Step 2 of Alg. 4.6, and q can be expressed as the sequence of its side tracks as $(S_1, S_2, \ldots, S_{h-1}, S)$. The

explicit expression of q is

$$\begin{aligned} q =& BP(s, S_1.src) \oplus S_1 \oplus \left(\bigoplus_{i=1}^{h-1} [BP(S_i.src, S_{i+1}.dest) \oplus S_{i+1}] \right) \oplus BP(S.dest, t) \\ =& p_1 \oplus \left(\bigoplus_{i=l}^{h-1} [BP(S_i.src, S_{i+1}.dest) \oplus S_{i+1}] \right) \oplus BP(S.dest, t). \end{aligned}$$

This means that p_1 is also a subpath of q, therefore, if p_1 contains a loop, then q also contains a loop. □

Theorem 4.5.2. *Given a graph $G(V, E)$ and a path $p \in P_{s,t}$—expressed as the sequence of its side tracks $(S_1, S_2, \ldots, S_h)$—if the subpath p_1 of p from s to the destination node of the last side track, i.e. $S_h.dest()$, contains loops, then the candidate generated from p in Step 1 of Alg. 4.6 also contains loops.*

Proof. By recovering the explicit expression of the candidate q generated from p in Step 1 of Alg. 4.6 as in the proof of the Theorem 4.5.1, we can prove that p_1 is also the subpath of q. Thus, if p_1 contains a loop, then q also contains a loop. □

The new heuristic, named **heuristic based on Eppstein's algorithm using loop filters** (HELF), applies the loop filters based on Theorem 4.5.1 and Theorem 4.5.2 into Alg. 4.6. The HELF is described in Alg. 4.8. While the loop filter before Step 1 applies directly Theorem 4.5.2, the loop filter before Step 2 and Step 3 applies Theorem 4.5.1 with $l = h-1$. It is possible to use only one of the loop filters, however, in our implementation both the filters are used.

4.5.3 Computational Results

In the first evaluation of the loop filters for Eppstein's algorithm, the computational results of the HELF and those of the HEA are compared in terms of average number of visited paths, average number of found paths and running time. The second evaluation focuses on the comparison of the running times of the HEA, the HMA, Yen's algorithm and the HELF.

In our experiments, all the methods were programmed in C++ and ran under the Windows platform on a personal desktop computer with an Intel (R) dual Core at 2.20 GHz processor, and 2 GB of RAM. The algorithms ran on the real maps of cities in Vietnam, Germany, USA, Thailand, The Philippine, Cambodia and Taiwan. For each map instance, all the method ran on the same pairs of randomly selected nodes on the map.

Table 4.3 shows the comparison between the computational results of the HEA and the HELF on finding 10 shortest loop-less paths on the real maps. The meanings of the columns are the same with those of Table 4.2. The computational results

Algorithm 4.8 The HELF for generating path candidates.

procedure HELF-DeviationPaths($G(V,E), s, t, p, GHeap[], LHeap[]$)
  $C \leftarrow \emptyset$ ▷ Set of deviated paths from p
  if SideTracks(p)= $NULL$ **then** ▷ p is the shortest path
    $NewCandidate \leftarrow GHeap(s).Minimum()$ ▷ Path with 1 side track
    $C \leftarrow C \cup \{$GetFullPath($NewCandidate$)$\}$
    Stop
  end if
  $(S_1, \ldots, S_h) \leftarrow$SideTracks($p$) ▷ Sequence of side tracks presenting p

*/*Step 1: Best candidate path deviated from p*/*
  if SubPath($p, s, S_h.dest()$) is loop-less **then** ▷ First filter
    $S_{h+1} \leftarrow GHeap(S_h.dest()).Minimum()$
    $NewCandidate \leftarrow (S_1, \ldots, S_h, S_{h+1})$
    $C \leftarrow C \cup \{$GetFullPath($NewCandidate$)$\}$
  end if

*/*Step 2: The next siblings of p*/*
  if SubPath($p, s, S_{h-1}.dest()$) is loop-less **then** ▷ Second filter
    $TurnNd \leftarrow S_h.source()$ ▷ Turning node
    for all $\bar{s} \in$ GetChildren($LHeap(TurnNd), S_h$) **do**
      $NewCandidate \leftarrow (S_1, \ldots, S_{h-1}, \bar{s})$
      $C \leftarrow C \cup \{$GetFullPath($NewCandidate$)$\}$
    end for

*/*Step 3: Candidates on the global heap*/*
    if $S_h \in GHeap(TurnNd)$ **then**
      for all $\bar{s} \in$ GetChildren($GHeap(TurnNd), S_h$) **do**
        $NewCandidate \leftarrow (S_1, \ldots, S_{h-1}, \bar{s})$
        $C \leftarrow C \cup \{$GetFullPath($NewCandidate$)$\}$
      end for
    end if
  end if
  Return C
end procedure

Table 4.3: Comparison between the HEA and the HELF on finding 10 shortest loop-less paths.

City maps	Eppstein (HEA)			HELF		
	Vs	Fs	Time	Vs	Fs	Time
HD-DE1k	437.33	10.00	0.0090	75.33	10.00	0.0106
HP-VN2k	1486.80	9.80	0.0206	91.50	10.00	0.0131
BH-VN4k	1648.73	9.40	0.0297	152.87	10.00	0.0197
NY-USA5k	1437.29	9.43	0.0423	20.71	10.00	0.0181
VT-VN5k	2895.27	9.27	0.0438	169.13	10.00	0.0237
MH-DE6k	1411.80	9.40	0.0445	63.87	10.00	0.0217
DN-VN8k	748.07	9.40	0.0610	37.80	10.00	0.0334
HN-VN9k	1348.93	9.40	0.0503	37.87	10.00	0.0389
PP-CB9k	12.33	10.00	0.0645	11.87	10.00	0.0393
MNL-PP12k	858.73	9.73	0.1000	38.00	10.00	0.0785
TP-TW21k	20.92	10.00	0.1023	18.75	10.00	0.1057
BK-TL22k	55.33	10.00	0.1071	25.87	10.00	0.0859
HCM-VN24k	44.53	10.00	0.2015	20.33	10.00	0.1351
Average	**954.31**	**9.68**	**0.0674**	**58.76**	**10.00**	**0.0480**

Table 4.4: Reduced times due to applying the loop filters to Eppstein's algorithm.

Maps	k=5	k=10	k=20	k=30	k=40	k=50	k=60
	Diff.(%)	Diff.(%)	Diff.(%)	Diff.(%)	Diff.(%)	Diff.(%)	Diff.(%)
HD-DE1k	−14.67	+17.78	−83.75	−77.66	−90.63	−88.75	−83.95
HP-VN2k	−28.93	−36.41	−77.80	−79.15	−88.13	−88.24	−90.93
BH-VN4k	−12.17	−33.86	−75.35	−75.46	−77.94	−83.70	−82.36
NY-USA5k	−9.42	−57.09	−57.64	−66.53	−68.44	−64.08	−80.28
VT-VN5k	−12.93	−45.97	−75.36	−80.11	−79.56	−74.43	−86.90
MH-DE6k	−32.47	−51.35	−47.23	−66.74	−73.65	−75.10	−81.24
DN-VN8k	+0.96	−45.25	−51.85	−66.05	−61.34	−68.76	−85.13
HN-VN9k	−9.38	−22.55	−6.10	−33.28	−51.14	−62.18	−67.45
PP-CB9k	−16.15	−39.05	−15.15	−38.70	−29.18	−4.57	−45.27
MNL-PP12k	−27.05	−21.47	−14.57	−39.05	−38.12	−45.75	−67.17
TP-TW21k	−7.31	+3.34	−19.23	−0.78	−29.48	−5.06	−40.25
BK-TL22k	−13.37	−19.85	−9.32	−17.03	−16.35	−37.25	−68.25
HCM-VN24k	−6.73	−32.98	−35.70	+1.54	−2.35	−3.02	−42.53
Average	**-14.59**	**-29.59**	**-43.77**	**-49.15**	**-54.33**	**-53.91**	**-70.90**

of the HEA showed on Table 4.2 are reused, thus we can indirectly compare the computational results of the HMA with the computational results of the HELF. It can be easily seen on Table 4.3 that the average number of visited paths by the HELF is significantly smaller than those by the HEA. As a result, the HELF can

find all the 10 shortest loop-less paths in all the maps that the HEA cannot due to the limitation on the number of iterations. In most of cases, the HELF has better running time than those of the HEA, but there are two exceptions on the maps namely "HD-DE1k" of Heidelberg, Germany and "TP-TW21k" of Taipet, Taiwan. The average running times of the HELF in those cases are lightly greater than the average running times of the HEA since the contribution of reducing the number of visited paths cannot compensate the additional spending time for the loop filters. This fact may happen in some rare cases, e.g. when the source node is close to the destination node or when the number of visited paths with loops is significantly small in comparison to the number of visited paths without loop. However, those cases are not popular in our experiments, and the benefit of using the loop filters is generally enough to compensate the additional computing time.

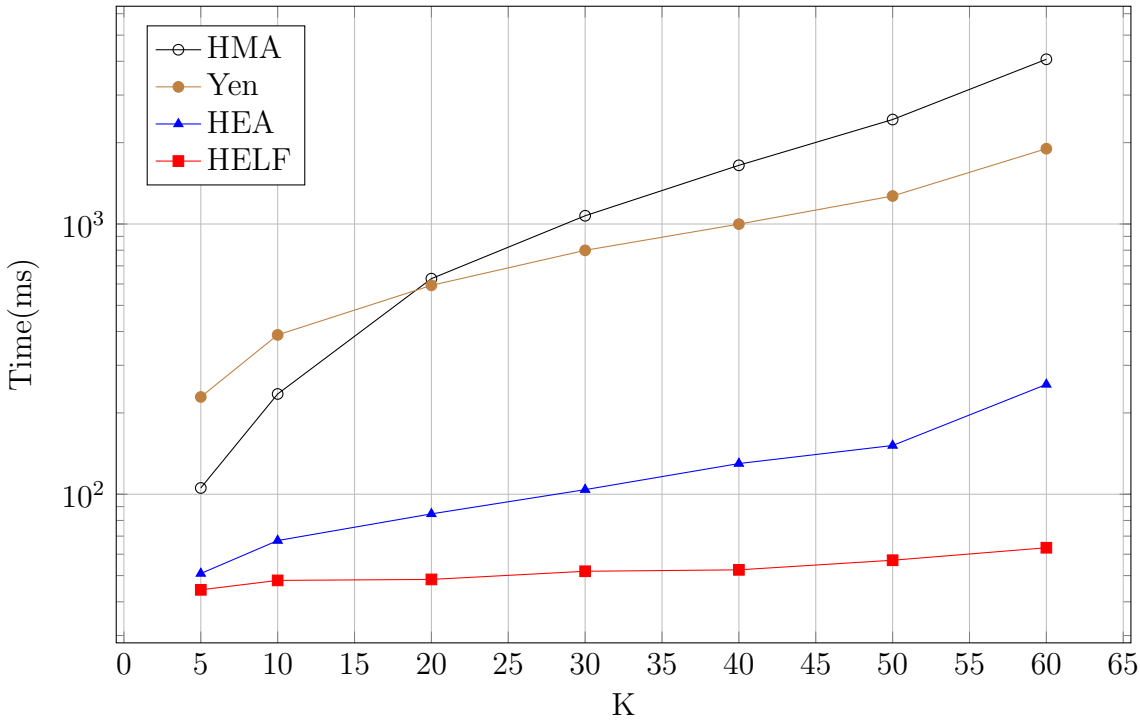


Figure 4.6: Comparison of running times of the HMA, Yen's algorithm, the HEA, and the HELF.

Table 4.4 shows the values of the percent deviation of the running time of the HELF and the running time of the HEA, i.e. $100(T_{HELF} - T_{HEA})/T_{HEA}$, where T_{HELF} and T_{HEA} are the running times of the HELF and the HEA, respectively. There are 7 columns for 7 values of k in the range $[5, 60]$. The results show that, in most of the cases, the running time of the HELF is better than those of the HEA. There are 4 cases where the running times of the HELF are lightly greater than the running times of the HEA, however, when k is greater than 30, the HELF dominates the

HEA in all the cases. This implies that using the loop filters in the HELF has a significantly effective. The effectiveness of the loop filters is greater when k increases. This fact is indicated in the last row of Table 4.4 that shows the average differences of running time between the two methods. The difference is -14.59% with $k = 5$ and up to -70.90% with $k = 60$.
Figure 4.6 illustrates the average running times of the HMA, Yen's algorithm, the HEA and the HELF with different values of k. In all the cases, the HELF dominates others. In addition, when k increases, the running time of the HELF grows slowly while those of the others grow rapidly. More details about the computational results can be found in Appendix B.

4.6 Dissimilar Shortest Paths Problem

The **dissimilar shortest loop-less paths** (DSLP) problem is an extension of the KSP problem where the paths are not only loop-less but also dissimilar from each other. An application of the DSLP problem is in hazardous material transport where managers have to make a routing plan such that the probability of accidents on each area is equal to those on other areas. Similarly, in radiation materials transportation if a large number of vehicles travel in similar paths, it could be dangerous for people living around the paths since the radiation can be accumulated vehicle by vehicle. The DSLP problem has been studied by various researchers, e.g. Erkut et al. [22], Dell'Olmo et al. [16], and Marti et al. [47]. In this section, we present some approaches for the DSLP problem including an approach using the HELF presented in Section 4.5.
Given a graph $G(V, E)$, two nodes s, t, number of paths $k \geq 1$, and the lower bound on minimum dissimilarity between two paths $0 < \varepsilon \leq 1$. There are two major approaches for finding k shortest loop-less paths from s to t such that the dissimilarity between two found paths is not less than ε. The first approach, called subset selecting, is to find a set P of $h > k$ shortest loop-less paths, and then select the best k dissimilar paths in P. The second approach, named on-line approach, determines the best dissimilar shortest paths one by one.

4.6.1 Subset Selecting Approach

Algorithms following the **subset selecting approach** consist of two main steps. The first step is to determine a set P of $h > k$ shortest loop-less paths by an algorithm for the KSLP problem, whereas the second step is to select the best k dissimilar paths from the h predetermined paths in the first step. Some methods have been proposed for selecting the set of the k best dissimilar paths from set P, such as the max-min method or the p-dispersion method.
Assume that $P = \{p_1, p_2, \ldots, p_h\}$ is a set of paths, the **max-min method** for

finding the best subset of k dissimilar paths in P is described in Alg. 4.9 where ε is the lower bound on the dissimilarity between two paths and the function $\textsc{Diss}(p, q)$ returns the dissimilarity between the paths p and q. In each iteration, one path in P is selected and added to the set D of the determined paths. The selected path is a path that the minimum dissimilarity between it and other paths on P is maximal. If the maximum value is less than the lower bound on dissimilarity ε, then the method terminates.

Algorithm 4.9 The max-min method for the DSLP problem.

1: **procedure** DSP-MaxMin(P, k, ε)
2: $D \leftarrow p^1$ ▷ p^1 is the shortest path
3: $P \leftarrow P \setminus \{p^1\}$
4: **while** $|D| < k$ **do**
5: $d^{opt} \leftarrow \underset{p^i \in P}{\text{Max}} \underset{p^j \in D}{\text{Min}} \textsc{Diss}(p^j, p^i)$ ▷ Optimal value
6: **if** $d^{opt} \geq \varepsilon$ **then**
7: $p^* \leftarrow \arg\min\{\underset{p^i \in P}{\text{Max}} \underset{p^j \in D}{\text{Min}} \textsc{Diss}(p^j, p^i)\ \}$ ▷ Optimal path on P
8: $D \leftarrow D \cup \{p*\}$
9: $P \leftarrow P \setminus \{p*\}$
10: **else**
11: Stop
12: **end if**
13: **end while**
14: Return D
15: **end procedure**

Another method for selecting the best k dissimilar paths from a given set of paths P is the **p-dispersion method** that is introduced by Erkurt et al. in [22]. The advantage of this method in comparison to the max-min method is that, it can be solved efficiently by a number of existing algorithms for the p-dispersion problem, see [21].
In summary, the subset selecting is an approach for the DSLP problem with a large number of efficient existing algorithms. However, the disadvantage of this approach is that, it cannot guarantee to find out k satisfactory paths if the set of the predetermined shortest loop-less paths is not large enough. There is no upper bound on the number of shortest loop-less paths should be predetermined to gain k dissimilar paths with a given lower bound of dissimilarity between two paths.

4.6.2 On-line Approach

In the **on-line approach**, paths are determined one after another. The next dissimilar path is determined according to the information of all the previously determined

paths. Three well known methods in this approach are the gateway method, the penalty method, and the heuristic based on ranking algorithms (HRA).

Gateway method: In this method, a set of k distinct nodes, so-called gates, must be predetermined. The dissimilar shortest paths are determined as the shortest paths containing one of the predetermined nodes. The method requires the knowledge of the "gates". However, it cannot guarantee that the optimality on the length of paths as well as the dissimilarity between determined paths.

Penalty method: The idea of this method is to add penalty costs to the links on the previously selected paths so that the next shortest path on the updated graph should be dissimilar from the previously selected paths. There are various strategies for adding penalty costs regarding the characteristics of the considering maps. The advantage of this method is the high probability of finding dissimilar paths. However, the length of the next selected path may be considerably greater than those of the previously selected paths. Algorithm 4.10 describes the penalty method for finding k shortest dissimilar paths from s to t where $PCost$ is the penalty factor, ε is the lower bound on dissimilarity between two paths, and $Imax$ is the maximum number of iterations.

Algorithm 4.10 Penalty method for the DSLP problem.

1: **procedure** DSP-PENALTY($G(V, E), s, t, k, PCost, \varepsilon$)
2: $D \leftarrow \emptyset$ ▷ Set of dissimilar paths
3: $Imax \leftarrow$ maximum number of iterations
4: $i \leftarrow 0$ ▷ Number of iteration
5: **while** $|D| < k$ and $i < Imax$ **do**
6: $i \leftarrow i + 1$
7: $p \leftarrow$ SHORTESTPATH(G, s, t)
/*Add penalty cost*/
8: **for all** e on p **do**
9: $w(e) \leftarrow w(e)(1 + PCost)$ ▷ Add penalty cost
10: **end for**
11: **if** $\underset{q \in D}{\text{Min}} \{\text{DISS}(p, q)\} \geq \varepsilon$ **then**
12: $D \leftarrow D \cup \{p\}$
13: **end if**
14: **end while**
15: Return D
16: **end procedure**

Heuristic based on ranking algorithm (HRA): In the previous sections, a number of methods for the KSLP problem have been investigated, such as the HMA, Yen's algorithm, the HEA, and the HELF. Those methods can orderly determine the shortest loop-less paths regarding their lengths. They can be used to determine

dissimilar shortest loop-less paths by checking the dissimilarity condition for the determined paths. The idea of HRAs is to apply try-and-error method for all the paths generated by the ranking algorithms. A shortest loop-less path is accepted if it is dissimilar from other previously selected paths. The general structure of HRAs is described in Alg. 4.11. The meanings of parameters are the same with those in Alg. 4.10. Since in HRAs paths are visited orderly by their lengths, the average length of the selected paths is surely minimal. This fact is the advantage of HRAs, however, HRAs may have to visit a significantly large number of loop-less paths for finding a given number of dissimilar paths.

Algorithm 4.11 General structure of the heuristic based on ranking methods for the DSLP problem.

```
1: procedure DSP-HeuristicRanking(G(V, E), s, t, k, ε)
2:     D ← ∅                                          ▷ Set of dissimilar paths
3:     Imax ← maximum number of iterations
4:     i ← 0                                          ▷ Number of iteration
5:     while |D| < k and i < Imax do
6:         i ← i + 1
7:         p ← next loop-less path
8:         if Min_{q∈D} {Diss(p, q)} ≥ ε then
9:             D ← D ∪ {p}
10:        end if
11:    end while
12:    Return D
13: end procedure
```

4.6.3 A New Heuristic and Results

As shown in Section 4.5, the HELF for the KSLP problem dominates other methods in terms of running time, thus in order to evaluate the HRA and the penalty method for the DSLP problem, we compare the penalty method using the bidirectional Dijkstra for finding shortest path, and the heuristic using the HELF for generating loop-less paths. The heuristic using the HELF is named **heuristic based on Eppstein's algorithm using loop-and-similarity filters** (HELSF). It follows all the steps as those in the HELF to generate orderly loop-less paths, but every generated path is checked both the loop-less condition and the dissimilarity condition.

Table 4.5 gives the computational results of the HELSF and the penalty method using the bidirectional Dijkstra on the map of Hanoi, Vietnam. The penalty method are implemented with three different penalty factors, i.e. $\beta = 0.2$, $\beta = 0.4$, and $\beta = 0.6$. The first left column labeled "Min Diss" shows the values of lower bound on the dissimilarity between two paths. These values are in the range $[0.10, 0.40]$. The

rest of the columns are grouped into 4 blocks. The first block is the computational results of the HELSF, the last three blocks are the computational results of the penalty method with different values of penalty factors. In turn, each block has two columns: one labeled "Time(s)" shows the average running times in second, the other labeled "Length" shows the average ratios of the average length of selected paths to the length of the shortest path, e.g. the value 1.15 in the "Length" column means the average length of all the determined paths is 1.15 times the shortest length. The results show that the running times of the penalty method are significantly less than those of the HELSF. Furthermore, when the lower bound on the dissimilarity increases, the running time of the HELSF increases significantly while the running time of penalty method is not considerably affected.

Table 4.5: Computational results of the HELSF and the penalty method using the bidirectional Dijkstra.

Min Diss.	**HELSF**		**Penalty-Bidirectional Dijkstra**					
			$\beta = 0.2$		$\beta = 0.4$		$\beta = 0.6$	
	Time(s)	Length	Time(s)	Length	Time(s)	Length	Time(s)	Length
0.10	0.1346	1.0162	0.0215	1.0909	0.3220	1.1336	0.0366	1.2163
0.15	0.2373	1.0223	0.0220	1.0935	0.0227	1.1336	0.0395	1.2163
0.20	0.5359	1.0285	0.0221	1.0989	0.0394	1.1336	0.0271	1.2163
0.25	1.3659	1.0383	0.0344	1.1165	0.0371	1.1381	0.0338	1.2163
0.30	1.9469	1.0587	0.0223	1.1165	0.0225	1.1514	0.0329	1.2163
0.35	3.2555	1.0847	0.0243	1.1323	0.0275	1.1721	0.0434	1.2163
0.40	8.4200	1.1223	0.0231	1.1411	0.0377	1.1784	0.0225	1.2326

However, the HELSF provides better set of dissimilar paths in terms of the average length of the paths, i.e. the average length of paths determined by the HELSF is less than those by the penalty method.
In conclusion, both the HELSF and the penalty method have advantages and disadvantages in comparison to the other. According to the objectives, e.g. running time or average of lengths, the most suitable method for the DSLP problem is selected.

4.7 Multi-objective Shortest Paths Problem

As defined in Subsection 4.1, the multi-objective shortest paths problem (MOSP) is the problem of determining $k \geq 1$ best paths corresponding to a given number of objectives. Normally, these objectives conflict with each other, thus it is normally impossible to get a solution that is the optimal solution regarding all the objectives. The motivation of this section comes from a fact in transport that drivers may choose their paths with respect to not only one but a number of objectives. In the first 5 sections of this chapter, routing problems regarding one objective, i.e. length, have

been studied. In Section 4.6 the dissimilarity between paths is also considered. In addition to routing problems related to path-selecting behaviors in transportation networks, we introduce the MOSP problem in this section.
In Subsection 4.7.1, basic facts, including some recent studies as well as existing approaches, about the MOSP problem are introduced. Subsection 4.7.2 presents a particular case of the MOSP problem, named bi-objective shortest paths (BOSP) problem, where only two objectives are considered simultaneously. In order to determine quickly a set of a given number of optimal paths for the BOSP problem, we present the multi-objective combinatorial optimization (MOCO) approach with a simple heuristic following the approach.

4.7.1 Introduction

Given a graph $G(V, E)$ where each link $e \in E$ has $m > 1$ attributes (objectives). The map from links on G to its objectives is given as

$$\begin{aligned} f : E &\longmapsto \mathbb{R}^m \\ e &\longrightarrow f(e) = (f_1(e), f_2(e), \ldots, f_m(e)), \end{aligned}$$

where $f_i(e)$, $i = 1, \ldots, m$, is the i^{th} attribute of the link e. If $p = (e_1, e_2, \ldots, e_l)$ is a path on G, then the vector of objectives of p is $f(p) = (f_1(p), f_2(p), \ldots, f_m(p))$, where $f_i(p) = \sum_{e \in p} f_i(e)$, for $i = 1, 2, \ldots, m$.

Definition 4.7.1. *Given two paths p_1 and p_2, we say that p_1 **dominates** p_2, denoted as $p_1 <_D p_2$, if $f_i(p_1) \leq f_i(p_2)$ for all $i = 1, 2, \ldots, m$ and there exists $1 \leq j \leq m$ such that $f_j(p_1) < f_j(p_2)$.*

A path $p \in P_{st}$ is a **non-dominated path** from s to t if p is not dominated by any other paths in P_{st}. The MOSP problem is the problem of finding all or $k \geq 1$ non-dominated paths from a given source node to a given destination node.

Definition 4.7.2. *Let C be a cycle (loop) in graph $G(V, E)$. The cycle C is an **absorbent cycle** if there exists $i \in \{1, 2, \ldots, h\}$ such that $f_i(C) < 0$.*

In the MOSP problem, if a connected graph contains an absorbent cycle, then there is an infinite non-dominated path, thus there is no finite algorithm that can find all the non-dominated paths, i.e. no algorithm can determine all the non-dominated paths in a finite time. This is confirmed in the following theorem.

Theorem 4.7.3. *Given a connected graph with an absorbent cycle. The is no finite algorithm that can find all the non-dominated paths.*

Proof. Let $p = (v_1, \ldots, v_m, C, v_n, \ldots, v_h)$ be a path from s to t containing an absorbent cycle C. We have the set $S = \{p^l \mid p^l = (v_1, \ldots, v_m, C^l, v_n, \ldots, v_h)\}$, where

C^l is a path including l times cycle C. Because C is an absorbent cycle, so there is $u \in \{1, 2, \ldots, k\}$ such that $f_u(C) < 0$. Assume that there exists a finite algorithm that can find the set of all non-dominated paths $X = (q^1, q^2, \ldots, q^N)$. Let $f_u(q^v) = \arg\min\{f_u(q^i) \mid 1 \leq i \leq N\}$. Because $f_u(C) < 0$ there is $l > 0$ such that $f_u(p^l) < f_u(q^v)$. This means that p^l is not dominated by any paths in X and the algorithm cannot determine p^l. The conflict indicates that the assumption is not true. □

According to Theorem 4.7.3, algorithms, that aim at determining all the non-dominated paths, cannot apply to graphs with an absorbent cycle. Since this thesis focuses on road networks where objectives on links are positive, we consider only graphs without absorbent cycles.
There are a number of approaches for the MOSP problem have been proposed by various researchers in recent decades. The labeling approach is a popular approach for the MOSP problem with many algorithms and speed-up strategies for the approach. According to the review of Medaglia et al. [50], Martins [49] is one of the pioneers in the labeling approach for the MOSP problem. Another approach is the **near shortest path** (NSP) that generates a number of candidates from the shortest-length path and selects non-dominated paths from those candidates. The **multi-objective combinatorial optimization** (MOCO) approach aims at determining the non-dominated supported paths that are the shortest path with respect to a linear combination of the objectives. This approach recently receives attention of various researchers, such as Ehrgott and Gandibleux [19], Geisberger et al. [30]. A survey on this approach is investigated by Bast et al. [4].
In the next subsection, we introduce the bi-objective shortest paths problem and the MOCO approach, that can work efficiently on large transportation networks. We also present a simple heuristic for finding a set of k non-dominated paths following the MOCO approach.

4.7.2 Bi-Objective Shortest Paths Problem

The **bi-objective shortest paths** (BOSP) problem is a case of the MOSP problem when only two objectives are considered simultaneously. For instance, in traffic systems where traffic jams occur frequently, drivers may choose their paths based on both traveling time at un-congested condition (free flow) and traveling time at congested condition (over-capacity flow). The BOSP problem has all the features of the MOSP problem, therefore, algorithms for the MOSP problem can be applied for solving the BOSP problem. However, there are also a large number of particular algorithms proposed for the BOSP problem or proposed for the MOSP problem but work efficiently on the BOSP problem. The multi-objective combinatorial optimization (MOCO) is such an algorithm that is efficient for determining a set of

non-dominated paths for the BOSP problem. In this subsection, we introduce the MOCO algorithm with a heuristic based on it for the BOSP problem.

Given a graph $G = (V, E)$ where each link $(i, j) \in E$ has two objectives: the first one is denoted as c_{ij} and the second one is denoted as t_{ij}. Let p be a path on G, the objectives of p are

- $C(p) = \sum_{(i,j)\in p} c_{ij}$—the first objective of p,
- $T(p) = \sum_{(i,j)\in p} t_{ij}$—the second objective of p.

The BOSP problem is to find a set $D \subset P_{st}$ of k non-dominated paths from s to t in terms of the first objective and second objective of paths.

Theorem 4.7.4. *If* $p^* = \arg\min_{p\in P_{st}} [\beta C(p) + (1-\beta)T(p)]$ *where* $0 < \beta < 1$, *then* p^* *is a non-dominated path.*

Proof. For a proof by contradiction, we assume that p^* is dominated by a path $p \in P_{st}$, i.e.,

$$C(p^*) \geq C(q) \text{ and } T(p^*) \geq T(q), \tag{4.2}$$

where equalities (4.2) do not happen concurrently. Because $0 < \beta < 1$, the following inequalities are held

$$\beta C(p^*) \geq \beta C(q) \text{ and } (1-\beta)T(p^*) \geq (1-\beta)T(q). \tag{4.3}$$

Since equalities (4.3) do not happen simultaneously, it holds that

$$\beta C(p^*) + (1-\beta)T(p^*) > \beta C(q) + (1-\beta)T(q). \tag{4.4}$$

Because (4.4) conflicts with the definition of p^*, the theorem is proved. □

Definition 4.7.5. *A path* $p^* \in P_{st}$ *from* s *to* t *is a* ***supported path*** *if there exists* $0 < \beta < 1$ *such that* $p^* = \arg\min_{p\in P_{st}} [\beta C(p) + (1-\beta)T(p)]$. *Otherwise,* p^* *is a* ***non-supported path****.*

Definition 4.7.6. *Given a graph* $G(V, E)$ *where each link* e *has two objectives* c_e *and* t_e, *and* $0 < \beta < 1$. *The* ***virtual length*** *of link* e, *denoted as* $w(e)$, *is defined as* $w(e) = \beta c_e + (1-\beta)t_e$.

Theorem 4.7.7. *Given a graph* $G(V, E)$ *and* $0 < \beta < 1$. *Each link* $e \in E$ *has two objectives* c_e *and* t_e. *If* p^* *is a shortest path in terms of virtual lengths, then* p^* *is a supported path.*

Proof. The virtual length of path p^* is calculated as the sum of virtual lengths of the links on p^*, i.e.

$$\begin{aligned} W(p^*) &= \sum_{e \in p^*} [\beta c_e + (1-\beta) t_e] \\ &= \beta C(p^*) + (1-\beta) T(p^*). \end{aligned}$$

Since p^* is a shortest path, for any $q \in P_{st}$ we have $W(p^*) \leq W(q)$, i.e.

$$\beta C(p^*) + (1-\beta) T(p^*) \leq \beta C(q) + (1-\beta) T(q), \quad \forall q \in P_{st}.$$

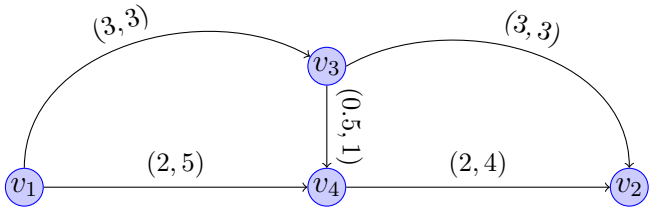
This implies that p^* is a non-dominated path, according to Theorem 4.7.4. □

Based on Theorem 4.7.7, we introduce a simple heuristic, named **heuristic on multi-objective combinatorial optimization** (HMOCO), to determine a set of k of non-dominated paths that are also supported paths. The HMOCO is described in Alg. 4.12 where s is the source node, t is the destination node, and $k \geq 1$ is the number of paths. There are at most $2k-1$ iterations. In each iteration, a new value of β in the range $(0, 1)$ is given. The function ShortestPath$(G(V, E, W), s, t)$ returns the shortest path from s to t in terms of virtual length W. The returned path is a non-dominated supported path from s to t. If the path is not yet selected, it is put on the set of selected paths. The value of β can be changed according to different strategies to get as much as possible supported paths.

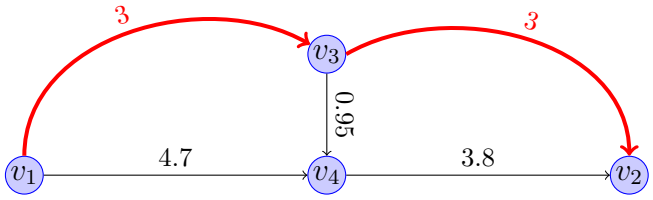
Algorithm 4.12 The HMOCO for the MOSP problem.

```
1:  procedure HMOCO(G(V, E), s, t, k)
2:      D ← ∅                                          ▷ Set of selected paths
3:      i ← 1
4:      while i < 2k and |D| < k do
5:          β ← i/(2k)
6:          for all e ∈ E do
7:              w_e ← βc_e + (1 − β)t_e                ▷ Update the virtual length
8:          end for
9:          p_i ← ShortestPath(G(V, E), s, t)          ▷ Regarding virtual lengths
10:         if p_i ∉ D then
11:             D ← D ∪ {p_i}
12:         end if
13:         i ← i + 1
14:     end while
15:     Return D
16: end procedure
```

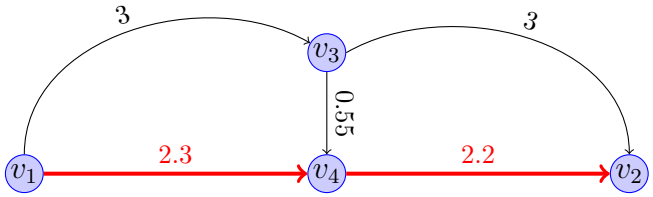
Note: All the paths determined by Alg. 4.12 are supported and non-dominated, however, there may be non-dominated, non-supported paths that could not be de-



(a) Links with two objectives (c, t)



(b) Virtual lengths with $\beta = 0.1$



(c) Virtual length with $\beta = 0.9$

Figure 4.7: Example of a non-dominated, non-supported path.

termined by the algorithm. For instance, Fig. 4.7 (a) shows a graph where each link e has two attributes denoted as (c_e, t_e). Three possible paths from v_1 to v_2 are: the path $p_1 = (v_1, v_4, v_2)$ with $C(p_1) = 4$, $T(p_1) = 9$, the path $p_2 = (v_1, v_3, v_2)$ with $C(p_2) = 6$, $T(p_2) = 6$, and the path $p_3 = (v_1, v_3, v_4, v_2)$ with $C(p_3) = 5.5$, $T(p_3) = 8$. All the paths, i.e. p_1, p_2, and p_3 are non-dominated paths.
With $\beta = 0.1$ the virtual lengths of the links in the graph are shown in Fig. 4.7 (b) where p_2 is the shortest path on the graph with respect to the new virtual lengths. Thus, p_2 is a supported path. Similarly with $\beta = 0.9$ the virtual lengths of the links are shown in Fig. 4.7 (c), and p_1 is the shortest path with respect to the new virtual lengths. The path p_1 is also a supported path.
We prove that p_3 is a non-supported path by contradiction. Assuming that p_3 is a supported path, i.e. there exists $0 \leq \beta \leq 1$ such that

$$p_3 = \arg \min_{p \in \{p_1, p_2, p_3\}} \{\beta C(p) + (1 - \beta) T(p)\},$$

we have

$$5.5\beta + (1 - \beta)8 \leq 4\beta + 9(1 - \beta)$$

and

$$5.5\beta + (1 - \beta)8 \leq 6\beta + 6(1 - \beta).$$

These inequalities are equal to $2.5\beta \leq 1.0$ and $2.0 \leq 2.5\beta$. This leads to a conflict that $1.0 \geq 2.5\beta \geq 2.0$. Thus, the path p_3 is not a supported path.

In conclusion, the multi-objective shortest paths (MOSP) problem is an important problem relating to path-selecting behaviors of drivers in transportation networks. There are a large number of approaches proposed for the MOSP problem, such as the labeling approach, the near shortest path approach, the multi-objective combinatorial optimization (MOCO) approach, etc. A particular case of the MOSP problem is the bi-objective shortest paths (BOSP) problem that is the most investigated by researchers recently. We have introduced a simple heuristic, named HMOCO, following the combinatorial optimization approach for the BOSP problem. Although the HMOCO cannot guarantee that all non-dominated paths can be determined, it can be useful in applications that need to determine quickly a given number of non-dominated paths.

4.8 Conclusions and Discussion

In this chapter we have introduced various problems of finding shortest paths in terms of one or many objectives and approaches for those problems. Among them Eppstein's algorithm is the best one in terms of time complexity, however, the determined paths by the algorithms may contain loops. In order to reduce number of visited path with loops, we introduced the loop filters for the Eppstein's algorithm. Based on these filters we propsed two new heuristics for the KSLP and the DSLP problems, in Section 4.5 and Section 4.6. The new heuristics, named HELF and HELSF, were tested on real city maps and showed theirs advantages in computational times to the other examined algorithms. In the last section of the chapter, i.e. Section 4.7, we introduced the multi-objective shortest paths (MOSP) problem with a particular case when only two objectives are considered simultaneously. The multi-objective combinatorial optimization (MOCO) is a prominent approach to find a set (may be not all) of non-dominated paths.

For the further works on the KSLP problem, we are investigating methods for reducing the computational time of the proposed loop filters for Eppstein's algorithm. Such methods could have a significant impacts on the total running time of the HELF and the HELSF. Other filters, that can detect bad parent paths, are also taken into account.

Chapter 5

Mixed Traffic Assignment Modeling

In this chapter, we investigate traffic assignment modeling on mixed traffic systems (MTS) that are popular in a number of developing countries, e.g., the Philippines, Taiwan, India, Thailand, and Vietnam, etc. The user equilibrium model, mentioned in Section 2.6, is utilized to develop a new traffic assignment model for mixed traffic systems. At the optimal solution of the new model, an equilibrium for each kind of vehicle is obtained. The promising computational results of the model using the real data in Hanoi, Vietnam, in Chapter 6 show the accuracy of the model and open a wide range of applications in further traffic planning problems.
The chapter is organized as follows. In the first section, i.e. Section 5.1, we introduce the basic knowledge about mixed traffic systems with an emphasis on mixed traffic systems dominated by motorcycles. Section 5.2 gives the details of the user equilibrium model, such as the mathematical formulations and methods for solving the model. We also introduce an important improvement, proposed by LeBlanc et al., for the Frank-Wolfe algorithm to solve the UE model. Section 5.3 introduces a new UE model for mixed traffic systems. The model separates traffic demands into two groups: the demands of 2-wheel vehicles and the demands of 4-wheel vehicles. Furthermore, public transportation vehicles are also taken into account in the new model. The mathematical formulation of the new model as well as an efficient method to solve it are investigated in this section. At last, Section 5.4 are conclusion and discussions.

5.1 Mixed Traffic Systems

A traffic system is a system consisting of traffic-related objects, such as roads, vehicles, signal lights, and driving policies, etc. In order to point out the characteristics of a traffic system we present the definition of traffic lane—one of the most important objects in traffic systems.

Definition 5.1.1. *A **traffic lane** is a part of a road that is designed for a single line of vehicles traveling one after one. The crucial role of lanes is to guide drivers and reduce the traffic conflicts. A lane is normally marked by two color lines, mostly white or yellow.*

The widths of traffic lanes for cars vary from 2.7 to 4.6 meters with respect to the country and the types of roads. For instance, in Germany the minimum lane width of the two-lane roads is 3.5 meters, however, the popular lane width of roads in the Autobahn system—the federal highway system in Germany without speed limit for some classes of vehicles, is 3.75 meters. The lane width of highways in the United States of America (USA) is typically 3.7 meters. Whereas, the width of lanes for bicycles is normally in the range of 1.2 to 1.5 meters.

Figure 5.1: An example of a mixed traffic system in Vietnam. Source: Vnexpress online newspaper, 2015.

In **standard traffic systems** lanes are assigned to different kinds of vehicles with respect to the size and the speed of vehicles. For instance, the lanes for bicycles are normally small and stay on the left-most side or the right-most side of the roads, while the lanes for **4-wheel vehicles**, e.g. cars, buses or trucks, are broader and are located at the center of the roads. In a number of cases, the lanes for 4-wheel vehicles are divided according to the kinds or speeds of 4-wheel vehicles, e.g. personal cars, trucks, or public buses.
A **mixed traffic system** (MTS) is characterized by a mixture of various kinds of vehicles traveling together without dedicated lanes for each kind of vehicle.

Figure 5.1 is an example of an MTS with a number of different kinds of vehicles, e.g. motorcycles, personal cars, and buses. Due to the differences over occupied road surface, speed, and driver behavior, it is normally more difficult to control an MTS than to control a standard traffic system. This fact comes partly from vehicles that are as small as motorcycles, but as fast as cars. These vehicles are called **2-wheel vehicles**, for convenience we often use motorcycles to mention this kind of vehicle. Note that other kinds of vehicles with two wheels but low speeds, e.g. bicycles, are not taken into account. It is dangerous to make narrow lanes for only 2-wheel vehicles because their speeds (in the cities) are as high as those of a cars and they are very flexible. Moreover, it is not effective in terms of economy, especially in traffic systems where 2-wheel vehicles dominate other kinds of vehicles. In most mixed traffic systems 2-wheel motor vehicles are allowed to travel on the standard lanes for 4-wheel vehicles. Because of their small sizes, they can pass over other vehicles (even cars) on the same lane. This means the traveling on lanes is not in a single line any more.
In a number of countries motorcycle is the most popular kind of personal vehicle. When motorcycles dominate other kinds of vehicles, the traffic system is called **mixed traffic system dominated by motorcycles** (MTSDM). In an MTSDM the traffic flow unit is normally the motorcycle unit (MCU). In order to estimate the flow on a link in an MTS, the equivalent values of different kinds of vehicles in MCU are investigated by various researchers, e.g. Chandra and Kumar [13], Chu et al. [14], Cao and Sano [11], etc.

5.2 User Equilibrium Model

As mentioned in Section 2.6 the UE model is based on the first principle of Wandrop [67] which states that the traveling times on all the actually used paths between an **original-destination** (O-D) pair of zones are the same and less than those of any unused paths. In this section, we make a review on proposed mathematical formulations for the UE model and then introduce the Frank-Wolfe algorithm for these formulations with an important improvement of LeBlanc et al. Some further materials about topics covered in this section are as follows.

1. M.J. Beckmann, C.B. McGuire, and C.B. Winsten. *Studies in the Economics of Transportation*. Rand Corporation, 1955, [5].

2. M. Frank and P. Wolfe, *An algorithm for quadratic program*, 1956, [27].

3. L.J. LeBlanc, E.K. Morlok, and W.P. Pierskalla. *An efficient approach to solving the road network equilibrium traffic assignment problem*, 1975, [46].

4. Y. Sheffi, *Urban Transportation Networks: Equilibrium Analysis with Mathematical Programming Methods*, 1985, [60].

Given a graph $G(V, E)$ representing a transportation network, in order to formulate the UE model, we use some notations as follows.

- $P \subseteq V$ is the set of **population zones** that have traffic demands to/from at least one other zone.
- d_{rs} is the **traffic demand** from the node $r \in V$ to the node $s \in V$. The demand d_{rs} is greater than zero if both of the nodes r and s are population zones, i.e. $d_{rs} \geq 0$ if $r, s \in P$, otherwise, $d_{rs} = 0$.
- x_{ij} is the total traffic flow on the link (i, j).
- $t_{ij}(x_{ij})$ is the traveling function on the link (i, j) where $x_{ij} \in E$ is the traffic flow on the link.
- $z_{ij}(x_{ij}) = \int_0^{x_{ij}} t_{ij}(\omega)d\omega$ is the contribution of the link (i, j) to the objective function.

5.2.1 The Original Formulation

In 1955, Beckmann [5] proposed a mathematical formulation whose optimal solution satisfies the first principle of Wandrop. In order to formulate the UE model, Beckmann uses the variable x_{ij}^s as the traffic flow on the link (i, j) going to s. The original mathematical formulation of the UE model is described as follows

$$\text{Minimize } z(x) = \sum_{(i,j)\in E} \int_0^{x_{ij}} t_{ij}(\omega)d\omega, \tag{OUE}$$

$$\text{Subject to } x_{ij} = \sum_{s\in P} x_{ij}^s \qquad \forall (i, j) \in E \tag{5.1}$$

$$d_{rs} = \sum_{j\in O(r)} x_{rj}^s - \sum_{i\in I(r)} x_{ir}^s \qquad \forall s \in P, r \in V, r \neq s \tag{5.2}$$

$$x_{ij}^s \geq 0 \qquad \forall (i, j) \in E, s \in P, \tag{5.3}$$

where $x = (x_{ij}^s)$ is the variable vector, $O(r)$ is the set of the out-going nodes from r, i.e. $O(r) = \{s \in V \mid (r, s) \in E\}$, and $I(r)$ is the set of the in-going nodes to r, i.e. $I(r) = \{v \in V \mid (v, r) \in E\}$. Equalities (5.1) state that the total flow on a link is the sum of all the flows going to the destination zones. Equalities (5.2) are flow constraints, i.e. for each zone $s \in P$, the total flow to s on the links going out from the node r is equal to the sum of the flows on the links going to r and the demand from r to s. Inequalities (5.3) are non-negativity constraints. The objective function of OUE can be rewritten as $z(x) = \sum_{(i,j)\in E} z_{ij}(x_{ij})$.

Lemma 5.2.1. *If $t_{ij}(x)$ is an increasing continuously differentiable function on $[0,\infty)$ for all $(i,j) \in E$, then the objective function of OUE is convex with respect to x_{ij} and x^s_{ij}.*

Proof. We have

$$\frac{\partial z(x)}{\partial x_{ij}} = \sum_{(u,v)\in E} \frac{\partial z_{uv}(x_(uv))}{\partial x_{ij}}.$$

Because with two different links (i,j) and (u,v) the total flows x_{ij} and x_{uv} on them are independent, it holds that $\partial z_{uv}(x_(uv))/\partial x_{ij} = 0$ if $(u,v) \neq (i,j)$. So we have $\partial z(x)/\partial x_{ij} = t_{ij}(x_{ij}) - t_{ij}(0)$. Since $t_{ij}(x_{ij})$ is increasing continuously differentiable, it holds that $\partial t_{ij}(x_{ij})/\partial x_{ij} \geq 0$. Hence

$$\frac{\partial^2 z(x)}{\partial^2 x_{ij}} = \frac{\partial t_{ij}(x_{ij})}{\partial x_{ij}} \geq 0.$$

This means that the objective function $z(x)$ is convex with respect to x_{ij}. We also have

$$\frac{\partial z(x)}{\partial x^s_{ij}} = \frac{\partial z(x)}{\partial x_{ij}} \times \frac{\partial x_{ij}}{\partial x^s_{ij}} = (t_{ij}(x_{ij}) - t_{ij}(0).$$

From this it holds that

$$\frac{\partial^2 z(x)}{\partial^2 x^s_{ij}} = \frac{\partial t_{ij}(x_{ij})}{\partial x^s_{ij}} = \frac{\partial t_{ij}(x_{ij})}{\partial x_{ij}} \frac{\partial x_{ij}}{\partial x^s_{ij}} = \frac{\partial t_{ij}(x_{ij})}{\partial x_{ij}} \geq 0.$$

Therefore, $z(x)$ is also convex regarding x^s_{ij}. □

Corollary 5.2.2. *If $t_{ij}(x)$ is an increasing continuously differentiable function on $[0,\infty)$ for all $(i,j) \in E$, then program OUE is a linear constraints convex minimization (LCCM) program.*

Proof. According to Lemma 5.2.1, the objective function of OUE is strictly convex. Moreover, it has linear constraints, so the program is an LCCM. □

In the UE model, the BPR function is widely used as the traveling time function on a link. In the rest of this thesis, we use the BPR function for all the formulations of the UE model, however, other traveling time functions could also be utilized. For each link $(i,j) \in E$ we have

$$t_{ij}(x) = T^0_{ij}\left[1 + \rho(\frac{x}{C_{ij}})^4\right] = a_{ij} + 5b^4_{ij}$$

where $a_{ij} = T^0_{ij}$ and $b_{ij} = \frac{\rho T^0_{ij}}{5C^4_{ij}}$, C_{ij} is the capacity, and T^0_{ij} is the traveling time at free flow of the link (i,j). Since the BPR function is an increasing continuously differentiable function, program OUE is an LCCM program, according to

Lemma 5.2.2. Thus, it can be solved by the Frank-Wolfe algorithm—mentioned in Subsection 2.5.3—however, the mathematical formulation of a real-scale traffic assignment (TA) problem may have millions of variables. For such large problems, the Frank-Wolfe algorithm takes considerable time to solve the model. Fortunately, this difficulty can be overcome with a crucial improvement by LeBlanc et al. that is investigated in the next subsection.

5.2.2 Solving by the Frank-Wolfe Algorithm

In 1975, LeBlanc et al. [46] introduced an important improvement to the Frank-Wolfe algorithm for solving OUE. They proposed a grouping technique that transforms a linear programming problem with $|V||E|$ variables into $|V|$ subproblems with $|E|$ variables. Therefore, instead of solving a big problem, we just need to solve a number of smaller subproblems. This improvement to the Frank-Wolfe algorithm gains advantages in both running time and memory consumption.

Recall that the Frank-Wolfe algorithm repeats two basic steps, namely finding a reducing direction and one-dimension search. In OUE $x = (x_{ij}^s)$, where $(i, j) \in E$ and $s \in P$, is the vector of the flow variables with the size $|x| = |E| \times |P|$. A bottleneck of the Frank-Wolfe algorithm is in the first step of determining a reducing direction. If $|V| = 5000$, $|E| = 10000$, and $|P| = 100$, then the size of the variable vector $|x| = 10^6$. In this case, it takes considerable time to solve the first step even with some very efficient algorithms, such as the Simplex algorithm that is an efficient algorithm proposed by Dantzig for linear programming. Moreover, in the Frank-Wolfe algorithm both of the steps are repeated a number of times. Thus, it is difficult to solve OUE in real time by using directly the original Frank-Wolfe algorithm. In order to divide the problem into smaller subproblems, LeBlanc et al. group variables into a number of smaller groups that are independent of each other. Because the variables in each subproblem are totally separated from any variables in other subproblems, the optimal solution to OUE is obtained by the optimal solutions of all the subproblems.
In the first step of the Frank-Wolfe algorithm we have

$$\text{Mininmize } \nabla z(x)y = \text{Minimize } \sum_{(i,j)\in E}^{s\in P} \frac{\partial z(x) y_{ij}^s}{\partial x_{ij}^s}. \tag{5.4}$$

Note that

$$\frac{\partial z(x)}{\partial x_{ij}^s} = \frac{\partial z(x)}{\partial x_{ij}} \frac{\partial x_{ij}}{\partial x_{ij}^s} = \frac{\partial z(x)}{\partial x_{ij}} = \sum_{(u,v)\in E} \frac{\partial z_{uv}(x_{uv})}{\partial x_{ij}}.$$

For every two different links $(i, j) \neq (u, v)$ the total flows on each of them is inde-

pendent of the total flow on the other, i.e.

$$\frac{\partial z_{uv}(x_{uv})}{\partial x_{ij}} = 0, \text{ thus we have } \frac{\partial z(x)}{\partial x_{ij}^s} = \frac{\partial f_{ij}(x_{ij})}{\partial x_{ij}} = t_{ij}(x_{ij}) = c_{ij}.$$

Therefore, (5.4) becomes

$$\text{Minimize} \sum_{s\in P}\sum_{(i,j)\in E} c_{ij}y_{ij}^s \geq \sum_{s\in P} \text{Minimize} \sum_{(i,j)\in E} c_{ij}y_{ij}^s. \tag{5.5}$$

The right hand side of (5.5) contains $|P|$ separated problems. For each node $s \in P$, there is a subproblem whose variables are the traffic flows on all the links to s, i.e. y_{ij}^s. The subproblem corresponding to the node $s \in P$ is

$$\text{Minimize} \sum_{(i,j)\in E} c_{ij}y_{ij}^s \tag{5.6}$$

$$\text{Subject to} \sum_{(i,j)\in E} y_{ij}^s - \sum_{(k,i)\in E} y_{ki}^s = d_{is} \qquad \forall i \neq s, i \in V \tag{5.7}$$

$$y_{ij}^s \geq 0 \qquad \forall (i,j) \in E.$$

Because there is no relationship between variables of a subproblem to those of any other subproblem, the equality in (5.5) is really achieved.
Program (5.6) is an LP, thus, it can be solved by the Simplex algorithm, however it can be transformed into the shortest path problem from all the nodes in P (excluding s) to s. Indeed, if we denote y_{ij}^{rs} as the traffic flow from r to s on the link (i,j), then we have $y_{ij}^s = \sum_{r\in P} y_{ij}^{rs}$. The optimal solution to program (5.6) is achieved at the optimal solutions of $|P|$ subproblems. Each subproblem is actually the shortest path problem from node $r \in V$ to s, that is formulated as

$$\text{Minimize} \sum_{(i,j)\in E} c_{ij}y_{ij}^{rs}$$

$$\text{Subject to} \sum_{(i,j)\in E} y_{ij}^{rs} - \sum_{(k,i)\in E} y_{ki}^{rs} = \begin{cases} d_{rs} & \text{if } i = r, \\ 0 & \text{otherwise,} \end{cases} \quad \forall i \neq s, i \in V$$

$$y_{ij}^{rs} \geq 0 \ \forall (i,j) \in E.$$

The solution to this program is

$$y_{ij}^{rs} = \begin{cases} d_{rs} & \text{if } (i,j) \text{ belongs to the shortest path,} \\ 0 & \text{otherwise.} \end{cases}$$

Thus, the solution to program (5.6) can be determined by finding the shortest paths

from every node $r \in V$ to the node s, and then assigning all the traffic demand d_{rs} to the shortest path from r to s. Note, because $d_{rs} = 0$ for all $r \notin P$, we actually need to find the shortest path from every node $r \in P$ to s.
In conclusion, with the improvement of LeBlanc et al., the Frank-Wolfe algorithm can solve effectively formulation OUE of the UE model on large graphs with thousands of nodes by solving a number of subproblems that can be easily solved by an existing shortest path algorithm, e.g. Dijkstra's algorithm or A-Star algorithm.

5.2.3 A Generalized Formulation

The disadvantage of the original formulation of the UE model, i.e. OUE, is that it restricts further constraints of routing behaviors, i.e., if we add more constraints about routing behaviors into OUE, it might lose the linearity, as a consequence, it is harder to solve the model. In order to overcome this disadvantage of formulation OUE, a general formulation of the UE model has been proposed. The general formulation separates the routing behaviors and the assignment modeling by predetermining the sets of potential paths for each pair of zones and doing traffic assignment modeling on those sets. Potential paths are defined as paths that drivers may select to travel from a zone to another zone according to their routing behaviors. Predetermining this set of paths enables us to add any kinds of routing behaviors into the UE model without changing the structure of its mathematical formulations.
Let P_{rs} be the **set of the potential paths** from $r \in P$ to $s \in P$, and the variable x_{rs}^p indicates **the traffic flow on the potential path** $p \in P^{rs}$. Let us denote

$$\delta_{ij}^p = \begin{cases} 1 & \text{if the link } (i,j) \text{ is on } p, \\ 0 & \text{otherwise.} \end{cases}$$

The **general formulation** of the UE model is expressed as

$$\text{Minimize } z(x) = \sum_{(i,j)\in E} z_{ij}(x_{ij}), \tag{GUE}$$

$$\text{Subject to } x_{ij} = \sum_{r,s\in P} \sum_{p\in P^{rs}} \delta_{ij}^p x_{rs}^p \qquad \forall (i,j) \in E \tag{5.8}$$

$$\sum_{p\in P^{rs}} x_{rs}^p = d_{rs} \qquad \forall r, s \in P \tag{5.9}$$

$$x_{rs}^p \geq 0 \qquad \forall r, s \in P, p \in P_{rs}, \tag{5.10}$$

where $x = (\ldots, x_{ij}^p, \ldots)$ is the vector of the flow variables on all the potential paths of all the O-D pairs of zones. Equalities (5.8) mean that the total flow on a link equals the sum of all the flows on the potential path containing the link. Constraints (5.9) ensure that the traffic demand between an O-D pair of zones is assigned completely

to the potential paths connecting them. We will prove that mathematical formulation GUE has a user equilibrium at the optimal solution, and it can be solved efficiently by the Frank-Wolfe algorithm with a similar grouping technique as the improvement of LeBlanc et al.

Indeed, in formulation GUE, x_{ij} is not used as a variable, we use it to make the formulation short and clear. The objective function $z()$ is increasing continuously differentiable with respect to each variable x_{ij}^p, thus program GUE is an LCCM program, and it satisfies the KKT conditions at the optimal solution. It means that at the optimal solution, with any O-D pair of zones $r, s \in P$ and $r \neq s$ and p is a potential path from r to s, there exist a series of numbers λ_{uv}, such that

$$x_{rs}^p \left(\frac{\partial z(x)}{\partial x_{rs}^p} - \sum_{u,v \in P} \lambda_{uv} l_{uv} \right) = 0 \text{ and } \frac{\partial z(x)}{\partial x_{rs}^p} - \sum_{u,v \in P} \lambda_{uv} l_{uv} \geq 0,$$

where

$$l_{uv} = \begin{cases} 1 & \text{if } u = r \text{ and } v = s, \\ 0 & \text{otherwise.} \end{cases}$$

The KKT conditions equal

$$x_{rs}^p \left(\frac{\partial z(x)}{\partial x_{rs}^p} - \lambda_{rs} \right) = 0, \tag{5.11}$$

and

$$\frac{\partial z(x)}{\partial x_{rs}^p} - \lambda_{rs} \geq 0. \tag{5.12}$$

We have

$$\frac{\partial z(x)}{\partial x_{rs}^p} = \sum_{(i,j) \in E} \left(\frac{\partial z_{ij}(x_{ij})}{\partial x_{ij}} \frac{\partial x_{ij}}{\partial x_{rs}^p} \right) = \sum_{(i,j) \in E} \left(t_{ij}(x_{ij} \frac{\partial x_{ij}}{\partial x_{rs}^p}) \right), \tag{5.13}$$

where

$$\frac{\partial x_{ij}}{\partial x_{rs}^p} = \delta_{ij}^p = \begin{cases} 1 & \text{if } (i,j) \text{ lies on path } p, \\ 0 & \text{otherwise.} \end{cases}$$

It can be seen that the right side of (5.13) is the traveling time on the path p from r to s, denoted as c_{rs}^p. Hence, equality (5.11) indicates that at the optimal solution, for every pair of zones $r, s \in P$ if $x_{rs}^p > 0$, i.e., the potential path p from r to s is actually used, then the total traveling time on p equals the constant value λ_{rs}. Inequality (5.12) means that the traveling time on any potential path from r to s is not less than λ_{rs}. These conditions prove that at the optimal solution the first principle of Wardrop is satisfied, i.e. GUE is a mathematical formulation of the UE model. This formulation will be used in the next section for the new UE model for

mixed traffic systems.

We now prove that the generalized formulation GUE can be solved efficiently by the Frank-Wolfe algorithm. Indeed, we also use the grouping technique similar to the grouping technique of LeBlanc et al. We have

$$\begin{aligned}\nabla z(x)y &= \sum_{r,s\in P}\sum_{p\in P_{rs}} \frac{\partial z(x)}{\partial x_{rs}^p} y_{rs}^p \\ &= \sum_{r,s\in P}\sum_{p\in P_{rs}} c_{rs}^p y_{rs}^p.\end{aligned}$$

The first step in the Frank-Wolfe algorithm becomes

$$\text{Minimize} \sum_{r,s\in P}\sum_{p\in P_{rs}} c_{rs}^p y_{rs}^p \geq \sum_{r,s\in P} \text{Minimize} \sum_{p\in P_{rs}} c_{rs}^p y_{rs}^p. \tag{5.14}$$

Because there are no constraints on the limitation of the flows on links, and the variable y_{rs}^p is independent of $y_{u,v}^q$ when $(r,s) \neq (u,v)$, the equality is really reached in (5.14). Thus, the first step of the Frank-Wolfe algorithm for GUE is separated into $|P|^2$ subproblems. Each subproblem corresponds to an O-D pair of zones, say r, s, that is formulated as

$$\begin{aligned}\text{Minimize} \sum_{p\in P_{rs}} c_{rs}^p y_{rs}^p \\ \text{Subject to} \sum_{p\in P_{rs}} y_{rs}^p = d_{rs} \\ y_{rs}^p \geq 0 \ \forall p \in P_{rs}.\end{aligned}$$

This linear program can be easily solved by simple operations. The optimal solution is obtained when all the traffic demands from r to s are assigned to the path on the set of potential paths P_{rs} with the smallest traveling time, i.e.

$$y_{rs}^p = \begin{cases} d_{rs} & \text{if } c_{rs}^p = \min\{c_{rs}^q \mid q \in P_{rs}\}, \\ 0 & \text{otherwise.}\end{cases}$$

In conclusion, formulation GUE for the UE model can be efficiently solved by the Frank-Wolfe algorithm using the grouping technique proposed by LeBlanc et al. Furthermore, it separates the routing behaviors and the main formulation, such that the further routing behaviors will not effect the main structure of the UE model. Due to the advantages of the GUE formulation, in the next section, we investigate a new UE model for mixed traffic systems with a mathematical formulation extended from GUE.

5.3 A New Model for Mixed Traffic Systems

Unlike in standard traffic systems where each kind of vehicle has its own lanes to travel, in a mixed traffic system (MTS) vehicles of various kinds travel together in the same lanes. In addition, different kinds of vehicles have different routing behaviors, thus, the potential paths for them are different. Our idea is to separate the traffic demands and potential paths with respect to kinds of vehicles. In this section, we investigate a new traffic assignment model for an MTS called **generalized user equilibrium for mixed traffic systems** (GUEM). First, we present the mathematical formulation of the GUEM model, then we discuss about solving the new model by using the Frank-Wolfe algorithm with the improvement of LeBlanc et al.

5.3.1 The Formulation of the GUEM Model

In this thesis, we only consider mixed traffic systems with two major kinds of vehicles that can be described as 2-wheel vehicles and 4-wheel vehicles. For convenience, we use motorcycles and cars to mention 2-wheel vehicles and personal 4-wheel vehicles. Buses are not includes in cars. Further kinds of vehicles can be added easily into the new model. The motorcycle unit (MCU) is used as the flow unit. Each of 2-wheel vehicles is equivalent to one MCU. The equivalent value of a car in MCU is investigated by various researchers, e.g. Chandra and Kumar [13], Chu et al. [14], and Cao et al. [11]. Given two population zones (nodes) $r, s \in P$, and a link $(i, j) \in E$, a number of new notations are used for the coming formulations as follows.

- d_{rs}^c is the traffic demands of cars (4-wheel vehicles) from r to s.
- d_{rs}^m is the traffic demands of motorbikes (2-wheel vehicles) from r to s.
- P_{rs}^c is the set of potential paths for cars from r to s.
- P_{rs}^m is the set of potential paths for motorbikes from r to s.
- α is the equivalent value of a 4-wheel vehicle in MCU.
- B_{ij} is the fixed bus flow on the link (i, j). We have $B_{ij} = b_{ij} \times \alpha_{bus}$, where b_{ij} is the bus flow on the link (ij) and α_{bus} is the equivalent value of a bus in MCU.
- $x_{rs}^{(c)p}$ is the variable of demands of cars traveling from r to s on the path $p \in P_{rs}^c$.
- $x_{rs}^{(m)q}$ is the variable of demands of motorbikes traveling from r to s on the path $q \in P_{rs}^m$.

The mathematical formulation of the new UE model for mixed traffic systems is as follows.

$$\text{Minimine } z(x) = \sum_{(i,j)\in E} z_{ij}(x_{ij}) \tag{GUEM}$$

$$\text{Subject to } x_{ij} = \sum_{r,s\in P} [\alpha \sum_{p\in P_{rs}^c} \delta_{ij}^p x_{rs}^{(c)p} + \sum_{q\in P_{rs}^m} \delta_{ij}^q x_{rs}^{(m)q}] + B_{ij} \quad \forall (i,j) \in E \tag{5.15}$$

$$d_{rs}^c = \sum_{p\in P_{rs}^c} x_{rs}^{(c)p} \quad \forall r,s \in P \tag{5.16}$$

$$d_{rs}^m = \sum_{q\in P_{rs}^m} x_{rs}^{(m)q} \quad \forall r,s \in P \tag{5.17}$$

$$x_{rs}^{(c)p} \geq 0 \qquad \forall r,s \in P, p \in P_{rs}^c \tag{5.18}$$

$$x_{rs}^{(m)q} \geq 0 \qquad \forall r,s \in P, q \in P_{rs}^m \tag{5.19}$$

where $x = (\ldots, x_{rs}^{(c)p}, \ldots, x_{rs}^{(m)p}, \ldots)$ is the vector of all variables of the flows on potential paths for both motorcycles and cars. Constraints (5.15) mean that the total flow on a link is equal to the sum of all the traffic flows all vehicles going through the link and the bus flow. Constraints (5.16) ensure that the traffic demands of cars from each O-D pair of zones are assigned completely to the potential paths for cars. Similarly, the meaning of constraints (5.17) is that all demands of motorcycles are assigned to potential paths for motorcycles. Constraints (5.18) and (5.19) are non-negativity constraints.

The set of potential paths for each kind of vehicle contains possible paths that drivers may select to travel regarding their objectives. This involves various factors, such as the characteristics of the traffic system, the weather or the culture of the country. For instance, in a number of countries drivers would not select paths whose length is two times greater than the shortest length. In this case, there should be an upper bound on the length of potential paths. The sets of potential paths may contain dissimilar shortest paths whose lengths are smaller than two times the shortest length. In another case, where traffic managers know about a number of popular paths between some O-D pairs, they can add it into the set of potential paths. Since there may be a great number of objectives and constraints on potential paths, we would not be able to go into details for all of them, however, we present some simple kinds of the sets of potential paths as follows.

- Set of k shortest loop-less paths. This set of potential paths can be used when drivers choose the path to travel regarding only the length of paths.
- Set of k shortest loop-less paths with a given upper bound on the lengths of the paths. Similar to the previous set of potential paths, however, there is an upper bound on the lengths of paths. For instance, the length of paths should not exceed two times the length of the shortest path.

- Set of k shortest dissimilar loop-less paths. All the accepted paths are dissimilar from each other, i.e. the dissimilarity between two paths is not less than a given value.

- Set of k shortest dissimilar loop-less paths with an upper bound on the lengths of the paths, i.e. paths are dissimilar to each other and their length is less than a given upper bound.

- Set of k non-dominated paths according to a given number of objectives, such as the length, the probability of traffic jams, and the safety of roads, etc.

These kinds of potential paths can be determined by using an efficient algorithm for the KSP problem mentioned in Chapter 4.

Theorem 5.3.1. *The new traffic assignment model, formulated as program GUEM, is a UE model in the sense that at the optimal solution of the program there is an equilibrium based on drivers for each kind of vehicle.*

Proof. We have

$$\begin{aligned}\frac{\partial z(x)}{\partial x_{rs}^{(c)p}} &= \sum_{(i,j)\in E}\left(\frac{\partial f_{ij}(x_{ij})}{\partial x_{ij}} \times \frac{\partial x_{ij}}{\partial x_{rs}^{(c)p}}\right)\\ &= \sum_{(i,j)\in E} t_{ij}(x_{ij}) \times \frac{\partial x_{ij}}{\partial x_{rs}^{(c)p}}\\ &= \alpha \sum_{(i,j)\in E} t_{ij}(x_{ij}) \times \delta_{ij}^{p}\\ &= \alpha w_p(x), \end{aligned} \tag{5.20}$$

where $w_p(x)$ is the traveling time on the path p for cars at the traffic flow x. Similarly, we have $\partial z(x)/\partial x_{rs}^{(m)q} = w_q(x)$, where $w_q(x)$ is the traveling time on the path q for motorcycles at the traffic flow x.
Because GUEM is an LCCM program, at the optimal point the KKT conditions are satisfied, i.e. given $r, s \in P$, $r \neq s$, there exist λ_{uv}^{c} and λ_{uv}^{m} such that

$$\begin{aligned} x_{rs}^{(c)p}\left(\frac{\partial z(x)}{\partial x_{rs}^{(c)p}} - \lambda_{rs}^{c}\right) &= 0\\ \frac{\partial z(x)}{\partial x_{rs}^{(c)p}} - \lambda_{rs}^{c} &\geq 0\\ x_{rs}^{(m)q}\left(\frac{\partial z(x)}{\partial x_{rs}^{(m)q}} - \lambda_{rs}^{m}\right) &= 0\\ \frac{\partial z(x)}{\partial x_{rs}^{(m)q}} - \lambda_{rs}^{m} &\geq 0. \end{aligned}$$

These conditions equal two groups of conditions as follows.

(KKTcar)	(KKTmotor)
$x_{rs}^{(c)p}\,(\alpha w_p(x) - \lambda_{rs}^c) = 0$	$x_{rs}^{(m)q}\,(w_q(x) - \lambda_{rs}^m) = 0$
$\alpha w_p(x) - \lambda_{rs}^c \geq 0.$	$w_q(x) - \lambda_{rs}^m \geq 0.$

The (KKTcar) conditions indicate that any car path, say $p \in P_{rs}$, that is actually used by at least one car driver, i.e. $x^{(c)p} > 0$, has the same traveling time λ_{rs}^c/α. The traveling time on any path that is not used by any car driver, i.e. $x^{(c)p} = 0$, is greater or equal to λ_{rs}^c/α. These conditions ensures the equilibrium state for car drivers at the optimal solution. Similarly, the (KKTmotor) conditions ensure the equilibrium state for motorcycle drivers. □

5.3.2 Solving the GUEM Model

In order to solve program GUEM, we also use the Frank-Wolfe algorithm with the same grouping technique applied to program GUE. In the step of determining the reducing direction, we have

$$\begin{aligned}
&\text{Minimize } \nabla z(x)\cdot y \\
&= \text{Minimize } \sum_{r,s\in P}\left[\sum_{p\in P_{rs}^c}\frac{\partial z(x)\cdot y_{rs}^{(c)p}}{\partial x_{rs}^{(c)p}} + \sum_{q\in P_{rs}^m}\frac{\partial z(x)\cdot y_{rs}^{(m)q}}{\partial x_{rs}^{(m)q}}\right] \\
&= \text{Minimize } \sum_{r,s\in P}\left[\alpha\sum_{p\in P_{rs}^c} w_p(x)\cdot y_{rs}^{(c)p} + \sum_{\bar{q}\in P_{rs}^m} w_q(x)\cdot y_{rs}^{(m)q}\right] \\
&\geq \sum_{r,s\in P}\left[\alpha\times\text{Minimize }\sum_{p\in P_{rs}^c} w_p(x)y_{rs}^{(c)p} + \text{Minimize }\sum_{q\in P_{rs}^m} w_q(x)y^{(m)q}\right]. \qquad (5.21)
\end{aligned}$$

The equality can be reached in (5.21) according to the same reason as in the case of GUE. Therefore, (5.21) can be solved by dealing with a number of subproblems. Each O-D pair of zones $r, s \in P$ has two corresponding subproblems formulated as follows.

$\text{Minimize } \sum_{p\in P_{rs}^c} w_p(x)y_{rs}^{(c)p} \qquad (5.22)$	$\text{Minimize } \sum_{q\in P_{rs}^m} w_q(x)y_{rs}^{(m)q} \qquad (5.23)$
$\text{S. t. } d_{rs}^c = \sum_{p\in P_{rs}^c} y_{rs}^{(c)p}$	$\text{S. t. } d_{rs}^m = \sum_{q\in P_{rs}^m} y_{rs}^{(m)q}$
$y_{rs}^{(c)p} \geq 0\ \forall p \in P_{rs}^c.$	$y_{rs}^{(m)q} \geq 0\ \forall q \in P_{rs}^m.$

These subproblems are very easy to solve with a number of basic mathematical

operations. The optimal solution to linear program (5.22) is

$$y_{rs}^{(c)p} = \begin{cases} d_{rs}^{c} & \text{if } w_p(x) = \min\{w_{\bar{p}}(x) \mid \bar{p} \in P_{rs}^{c}\}, \\ 0 & \text{otherwise.} \end{cases}$$

And the optimal solution to program (5.23) is

$$y_{rs}^{(m)q} = \begin{cases} d_{rs}^{m} & \text{if } w_q(x) = \min\{w_{\bar{q}} \mid \bar{q} \in P_{rs}^{m}\}, \\ 0 & \text{otherwise.} \end{cases}$$

5.4 Conclusions and Discussion

In this chapter, we investigated mathematical formulations of the UE model, such as OUE and GUE. The advantage of formulation GUE is that, it separates the flow constraints and the routing behaviors of drivers, such that it is able to add further constraints of routing behavior without changing the structure of the formulation.
A well known algorithm for solving programs OUE and GUE is the Frank-Wolfe algorithm. In 1975, LeBlanc et al. introduced the crucial improvement for this method to deal with these programs on large transportation networks. The improvement is based on the grouping technique of variables, where a big problem is divided into a number of subproblems that can be efficiently solved by the Simplex algorithm or by existing algorithms for the shortest path problem, e.g. Dijkstra's algorithm and A-star algorithm.
Based on the UE model, we have introduced a new traffic assignment model, named GUEM, for mixed traffic systems. The model is formulated for two major kinds of vehicles, namely 2-wheel vehicles and 4-wheel vehicles, however, it can be extended to further kinds of vehicles. The new model also takes the fixed bus flow on each link into account. In order to solve GUEM, the Frank-Wolfe algorithm is used with a grouping technique similar to those of LeBlanc et al. Due to the grouping technique, the model can be solved by solving a number of subproblems that need only basic mathematical operations. At the optimal solution of formulation GUEM of the new model, an equilibrium based on each kind of vehicle is gained. This means the GUEM model is a UE model. The accuracy of the model is examined in the traffic system dominated by motorcycles in Hanoi, presented in Chapter 6.

Because of the generality of the GUEM model, we can add more constraints on routing behaviors of drivers and also apply various kinds of traveling time functions to the model. It is in our plan to develop further traffic assignment models from the GUEM model, such as a dynamic user equilibrium model for mixed traffic where the traffic demands are dynamic.

Chapter 6

A Case Study: Hanoi Vietnam

This chapter presents the case study in Hanoi—the capital city of Vietnam—where the traffic system is characterized as a mixed traffic system dominated by motorcycles. There are an increasing number of serious traffic-related problems in Hanoi, e.g. traffic jams, air pollution, fatal accidents, etc. Traffic planning is determined as one of the priority investments for the future development of the city, however, there is a serious lack of particular research on the traffic in Hanoi, and on the urban traffic in Vietnam, in general.
Our contributions to this chapter include the traffic data collection with an online survey on the traffic behaviors in Vietnam, the parameter calibration for the BPR function applied to traffic system in Hanoi, and the implementation of the GUEM model using the collected data. The predicted traffic flows resulting from the new model GUEM are analyzed and compared to the real data of the traffic situation provided by the Remon-Hanoi project. The big agreement between the predicted output and the real traffic situation proves the accuracy of the new model. It opens a wide range of applications in traffic planning not only in Hanoi, but also in other major cities in Vietnam.
The outline of this chapter is given as follows. Section 6.1 gives the overview of the traffic system in Hanoi while Section 6.2 introduces the traffic data acquisition. In Section 6.3, we investigate a method for calibrating the parameters of the BPR function applying to the traffic in Hanoi. The implementation and the computational results of the GUEM model using the BPR function using the new calibrated parameters and the collected data are presented in Section 6.4. The last section, i.e. Section 6.5, is for the conclusion and discussions.

6.1 The Hanoi Traffic System

Vietnam is a developing country in Southeast Asia with more than 94 million inhabitants by 2015, according to [12]. Vietnam's economy has been growing rapidly

in the last two decades after the reformation (Doi moi) of the national economic system. Accompanying with the fast growth of economy is the high urbanization at an 2.95% annual rate of change in the period 2010-2015. By 2015, approximately 33.6% of the total population live in urban areas. This fact creates a large pressure on the existing urban traffic infrastructure, especially in major cities, e.g. Hanoi, Ho Chi Minh City, DaNang, and HaiPhong, where the government has invested a large amount of money and efforts to improve the traffic systems. According to the report of the World Bank [64], the amount of money invested in the traffic infrastructure reached 4.5% of the gross domestic product (GDP) of Vietnam in 2002. However, the traffic system cannot meet the rapidly increasing traffic demands, especially in urban areas. A number of traffic-related problems still remain serious in major cities.

Hanoi is the capital city and also the second largest city in Vietnam. According to the survey made by the general statistic office (GSOV) of Vietnam [31]—available on the website: www.gso.gov.vn—Hanoi has a population of nearly 6.7 million with about 3 million inhabitants living in the urban districts of Hanoi by 2011. In the city of Hanoi there are about $300,000$ personal cars and over 3.8 million motorcycles in 2012, according to [66], excluding the number of vehicles coming from other neighbor areas. The number of personal vehicles is increasing by 15% every year but the traffic infrastructure is still poor. It cannot meet the increasing demand of traffic, see [40]. The traffic system in Hanoi is a mixed traffic system dominated by motorcycles, i.e., the traffic contains various kinds of vehicles traveling together without dedicated lanes for each kind of vehicles, and motorcycles dominate other kinds of vehicles. Figure 6.1 shows a traffic jam at the Khuat Duy Tien intersection—the first intersection with 4 traffic levels in Hanoi. It is also the major gate in the West of Hanoi connecting the neighbor metropolitan area Ha Dong to the center of Hanoi. Traffic jams occur in this intersection frequently because of not only the extremely large amount of traffic demands but also the inefficient traffic controlling. As it can be seen, motorcycles, personal cars, and buses use the same lanes. The number of motorcycles dominates the number of other kinds of vehicles.

Figure 6.2 shows the development of vehicles in Hanoi in 2002, 2010, and 2015. It can be seen in the diagrams that the share of personal cars is rising rapidly, however, the share of motorcycles still dominates others. The share of public buses in 2015 may be higher since the data is based on the online survey where most of participants are officers while students and old people are the two main groups using public buses. However, the data shows the trend that the total share of personal cars and public buses is increasing up to almost 24% while the total share of motorcycles reaches the saturation proportion at approximately 76%. It is a fact that the public transport system can meet approximately 10% of the traffic demands. In the next couple of years, the new city electric tram system will be launched, planned in 2016. The new system may enhance the proportion of public transportation, however, it

Figure 6.1: Traffic congestion at the Khuat Duy Tien intersection in Hanoi on October 8, 2015. Source: VnExpress online newspaper.

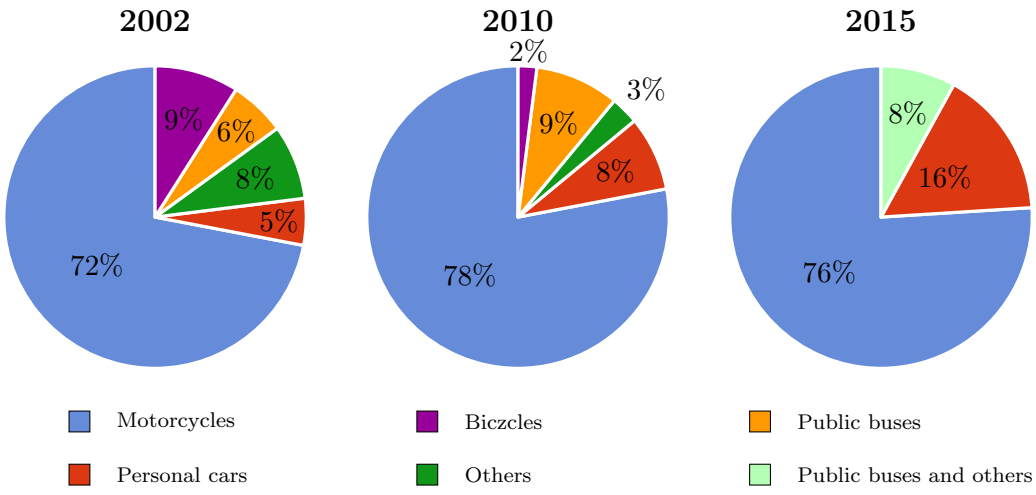


Figure 6.2: Vehicle share in Hanoi in 2002, 2010, and 2015. Source: Tranmoc 2002, Molt 2010, and Nam 2015.

is predicted that motorcycles will remain the major kind of vehicles in the next decade, i.e. by 2030. A number of the characteristics of the traffic system in Hanoi are pointed out as follows.

- It is a typical mixed traffic system dominated by motorcycles that account about 75% of the total number of vehicles. The rest of vehicles are mostly personal cars, small vans, and public buses.
- There are a large number of narrow roads that allow only 2-wheel vehicles to travel on.
- Drivers often break the traffic rules, e.g., they do not keep the minimum safety distance to other vehicles, drivers turn left or right without considering the traffic flow, etc. They may select paths even when their decisions can cause congestion.
- There are a considerable number of bidirectional roads without any separators, thus drivers can change to the opposite direction wherever they want.
- It is popular that residential houses locate beside the roads. This fact leads to behaviors of crossing the road to go home frequently even in rush hours.

With such characteristics, the traffic system in Hanoi is difficult to control efficiently. A number of traffic projects in Hanoi focusing on assigning traffic flows and building new traffic infrastructures have been launched. Recently, traffic planning in Hanoi receives an increasing attention of researchers. Some of them investigate macroscopic problems, such as traffic management [39, 64] and urban traffic development [42]. Some others study specific problems, e.g. characteristics of traffic systems dominated by motorcycles [11, 14, 38, 52], mixed traffic simulation [18], and public transportation system [55]. Our purpose, with a different objective, is to investigate a suitable TA model for the traffic system in Hanoi. The new model functions as the background for further applications in traffic planning, such as transportation network design, traffic management, and projects evaluation.

6.2 Data Acquisition

The lack of data is one of the biggest obstacle in traffic planning in Hanoi. A number of data are totally unavailable while some others are out of date. Even with some available data, the accuracy of them is still a question. Therefore, the first task of a traffic project in Hanoi is always data collection. In this section, we present methods for data collection in three subsections corresponding to geographical data, traffic demand data, and an online survey on traffic behaviors in Vietnam.

6.2.1 Geographical Data

The concerning geographical data is the transportation map of Hanoi that includes the data of geographical objects, e.g. roads, intersections, and connectors, etc. A number of map sources are available on the Internet, such as Google Maps, Openstreetmap, etc. In this doctoral project we imported mostly from **openstreetmap** (OSM) [36] that provides open, free, editable maps of the whole world. All the OSM data are contributed by volunteers and released with an open-content license. There are various methods for exporting OSM data. The simplest way is to use the "Export" function on the main page www.openstreetmap.org, however, this method cannot apply to a large area. The map data of large areas, e.g. a country, can be obtained from other resources, such as on the website geofabrik.de.
Because the OSM data were contributed by volunteers on the Internet without warranty of the accuracy of the data, all the exported OSM data were corrected via a number of steps. In the first step the locations of links and nodes were compared with those in the map provided by Here Map and Google Maps on the websites maps.google.com and www.here.com, respectively. In the second step, we used the software, named **Java OpenStreetMap** (JOSM), to filter out unnecessary map objects, such as the locations of restaurants, banks, and super markets, etc.
In OSM data, roads are classified into a number of popular types of highways, e.g. primary, secondary, and living street, etc. This classifying does not give exact information of the widths of the ways. Therefore, in the third step, we collected also the data of the widths of links by using the distance estimating tools on the Google Earth—a virtual globe, map, and geographical information program provided by Google—that is available at the website earth.google.com.

6.2.2 Traffic Demand and Traffic Flow Data

The traffic demands of an origin-destination (O-D) pair of zones are the number of trips from the origin to the destination within a given range of time, e.g. an hour or a day. The matrix of all the traffic demands from each zone to each other is called the **O-D matrix** or **demand matrix**. We investigated the O-D matrices of both motorcycles and personal cars. There were very few such kinds of data available in Hanoi. To our knowledge, by 2015, the only traffic demand data in Hanoi were collected in a project named the comprehensive urban development program in Hanoi capital city (HAIDEP) in 2007 [42]. The HAIDEP project was a project in cooperation between Japan International Cooperation Agency and Hanoi People's Committee for planning the space development of the urban area in Hanoi with vision to the year 2030. The project investigated seriously the traffic demands between 313 population zones corresponding to 313 wards (or quarters) of all the districts of Hanoi and Ha Tay—a former province next to Hanoi that was subsumed

into Hanoi in August, 2008.
In this doctoral thesis, we focus on the traffic demand data in the urban area of Hanoi, thus we consider each of the rural districts and urban wards as one population zone. As a result, the number of zones is reduced by 180 to 133. The O-D matrix for the 133 zones was recalculated from the former O-D matrix for 313 zones. Furthermore, the data of the HAIDEP project were collected in the range 2005 to 2007, i.e. the data were almost 10 years old by 2015. In order to update the data, we adapted the data according to the growth of the population. For each O-D pair of zones, the traffic demands were updated by multiplying the former data demand with the ratio of population 2015 to the population in 2005, i.e.

$$d_{2015} = d_{2005}\frac{P_{2015}}{P_{2005}},$$

where d_{2005} and d_{2015} are the traffic demands between an O-D pairs and P_{2005}, P_{2015} are the populations of Hanoi in 2005 and in 2015, respectively. Because the rates of population growth vary according to areas, it could be better if we update the traffic demand based on the information of population growth of the zones instead of the population growth of the whole city. However, the data of population of each zone were not easy to acquire.
Another investigated kind of traffic data is the statistic of traffic flows on the links in the traffic system in Hanoi. This kind of data was recorded by the CadPro company [10] that was employed by the Hanoi government for building and maintaining the traffic management system (TMS) in Hanoi. By early 2015, CadPro has established hundreds of traffic cameras at the major intersections. Each traffic camera sends the video stream to the server where the data stream is analyzed by various image processing techniques to estimate the average speed as well as the traffic flow on the road. The data is updated in every single time interval, e.g. 2 minutes, or 5 minutes.

6.2.3 Traffic Survey in Vietnam

One of the most challenging tasks in traffic assignment modeling is to figure out the routing behaviors of drivers. In Vietnam by 2015 there were only few surveys on the traffic behaviors of drivers in urban areas, they were even not publicly available. That was our motivation to make a survey on urban traffic in Vietnam. The survey was launched on August 15th, 2014 using Google Form—available at the website www.google.com/forms. It was distributed online via a number of social networks and popular forums in Vietnam, e.g. Facebook and Otofun groups. By September 15th, 2015 there were 316 participants, where 84% of them come from Hanoi and more than 7% of the participants come from Ho chi Minh City. The rest stay in other major cites in Vietnam. On the survey, we did not ask questions about personal

information, thus the participants could feel free to give their answers.
The first part of the survey are questions about the job groups, the kind of vehicles, the living locations, as well as the working locations. Whereas the second part and the third part are questions about the daily demands of traveling and the driving behaviors. The result of the kinds of vehicles in 2015 is illustrated in Fig. 6.2 in comparison to the existing data in 2002 and in 2010, see [55]. It can be seen that, in the period from 2002 to 2010, the shares of motorcycles and personal cars increase rapidly due to the economic growth. In the period from 2010 to 2015, the share of personal cars keeps increasing rapidly, while the share of motorcycles might be saturated and goes down slightly. The increase in the number of buses is not as fast as those of the personal cars and motorcycles. As a result, the proportion of public bus decreases slightly. Figure 6.3 visualizes the shares of the working groups. The result shows that nearly 90% of the participants have fixed working places. The number of officers is more than a half of the total participants.

In the second part, a number of questions about the daily traffic demand are given, e.g. the departure time and the favorite paths, etc. According to the results of the survey, the trip distributions of the participants are illustrated in Fig. 6.4. The trips are categorized into 4 groups: the trips from home to the working place in the morning, the trips from the working place to home in the afternoon, the trips going home at the lunch time, and the trips back to working place after the lunch time. In Vietnam, it is common that employers go home for the lunch instead of having lunch in restaurants or in canteens. The time interval is one hour, e.g. the value of the trips at $x = 6$ is the percentage of trips from 6 AM to 7 AM. Note that on the the survey of the HAIDEP project, only the number of trips within a day between each O-D pair of zones was investigated, the trip distribution within a day was not in the consideration. With the trip distribution and the traffic demand within a day, the traffic demand in each time interval can be calculated easily. As it can be seen, the most crowded time intervals are from 7 AM to 9 AM when people go to work and from 5 PM to 7 PM when people go home. This is in high agreement with the traffic situation in Hanoi, since most of traffic jams occur during those time intervals. Especially, in only one hour from 7 AM to 8 AM, the number of trips takes 23.48% of the total number of trips within a day—nearly 6 times higher than the average number in one hour.

In the third part, a number of questions about driving behaviors are given. On the multi-choice question about the behaviors when drivers meet traffic congestion, there are 4 options: driving on the sidewalks to get over the congestion, staying orderly in the queue, turning back, and other actions. The results in Fig. 6.5 show that there are nearly 40% of the participants consider the action “turning back”, about 17% of the participants consider the action “driving on the side tracks”, and only 60% of them consider the action “staying on the queue and waiting”—the appropriate

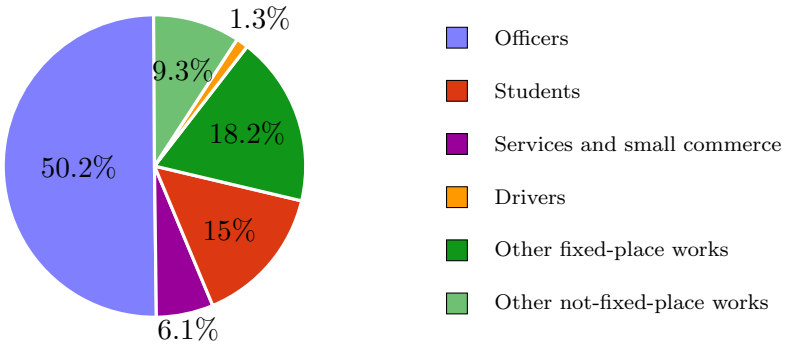


Figure 6.3: The share of working groups in Hanoi 2015. Source: Nam, 2015.

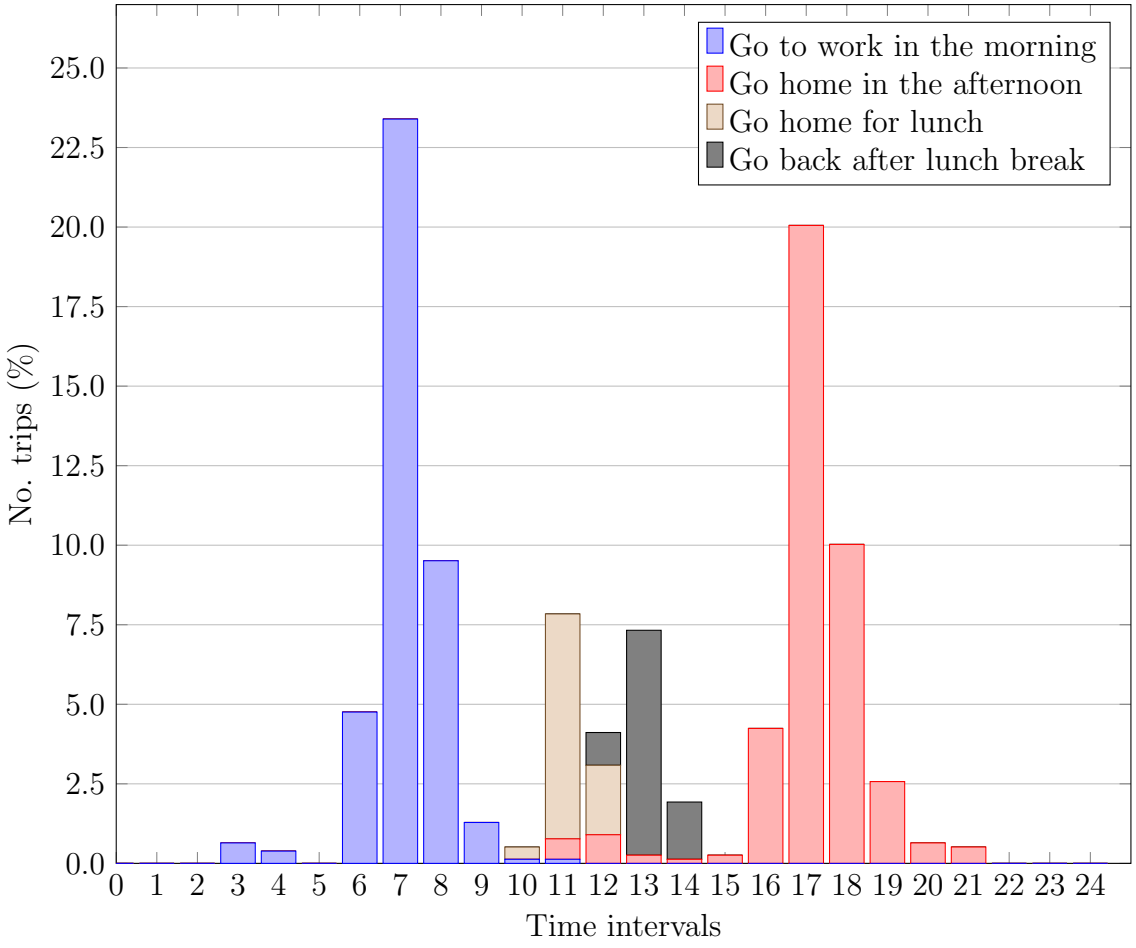


Figure 6.4: The trip distributions corresponding to time intervals within a day. Source: Nam, 2015.

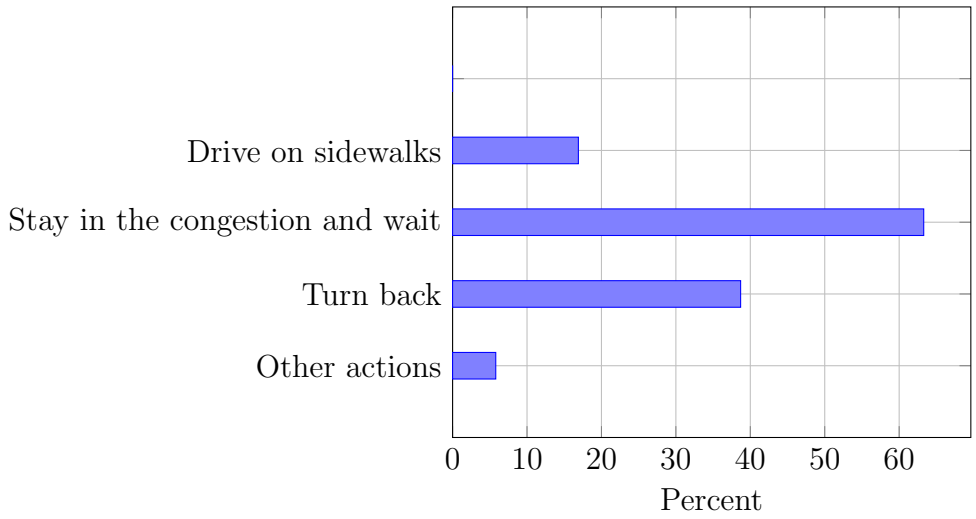


Figure 6.5: The actions of drivers in traffic congestion. Source: Nam, 2015.

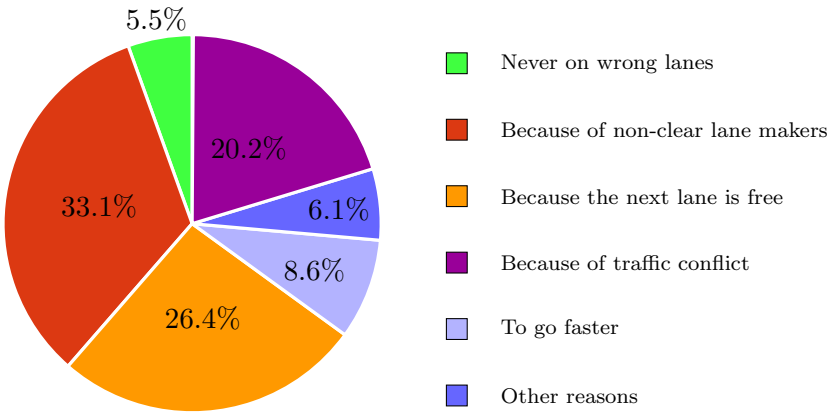


Figure 6.6: The causes of lane crossing actions. Source: Nam, 2015.

action because other actions, i.e. driving on the sidewalks and turning back, could make the traffic situation worse.

In Hanoi, the policy about separated lanes for cars and motorcycles was tested on a number of roads, thus on the survey, we also give a question about lane crossing behaviors. The results of this question is visualized in Fig. 6.6. It is a big surprise that only 5% of the participants confirm that they have never gone on the wrong lanes. The rest, i.e. 95% of the participants admit that they have at least one time gone on the wrong lanes because of various reasons, e.g. unclear lane separators, going faster, or traffic conflict, etc.

On the question about the number of alternative paths from home to the working place and back, the most common answers are two or three paths, while the number of participants having more than 5 alternative paths is not significant, see Fig. 6.7. There are about 10% of the participants having only one path from home to the working place and back. This is reasonable, since the traffic system in Hanoi depends mostly on a number of major roads known as ring roads and centering roads. Thus, in the rush hours, a very large number of people go on the same major roads. For instance, to travel from the East to the West of the city, people have to use one of the three ring roads. Similarly, to travel between the North and the South of the city, people have to choose one of the centering roads.

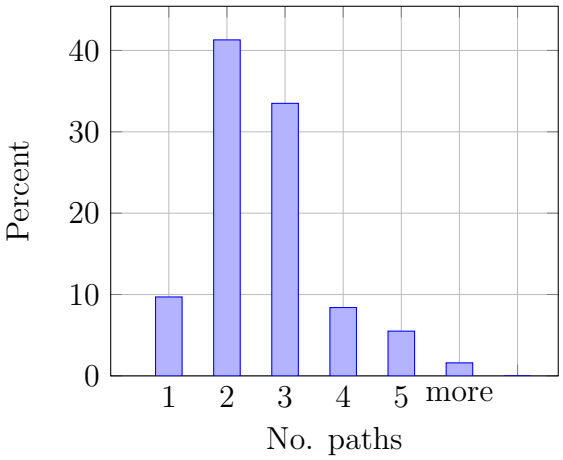


Figure 6.7: Number of alternative paths. Source: Nam, 2015.

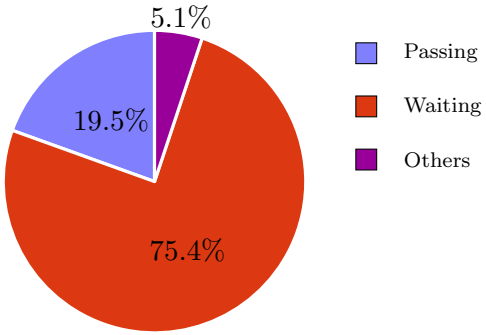


Figure 6.8: Actions of drivers at a traffic light turning red. Source: Nam, 2015.

In this part, we investigate the daily used paths to go to work and to come back home of participants. This kind of data is used to evaluate the set of predicted potential paths for each O-D pair of zones used in the new traffic assignment modeling. In Hanoi, crossing red light is a common traffic behavior, especially when there are no traffic polices at the intersections. Figure 6.8 shows the results of the survey on drivers' actions when they are in a traffic flow going through an intersection and the traffic light is changing to red but the flow of people keeps going on. It can be seen that, about 19.5% of the participants would pass the red light if they are in such a situation. This is also

a reason for the frequent traffic conflicts at the intersections.
Finally, we investigate which factors affect the routing behaviors of drivers. The result in Table 6.1 points out that most of participants consider the probability of congestion, the length, and the traffic density on the path as main factors affecting their routing behaviors. This indicates that the two most major factors influencing the routing behaviors of drivers are geographic distance and the probability of traffic jams.

Table 6.1: The effected factors on routing behaviors. Source: Nam, 2015.

Factors	**Selection** (%)
Probability of congestion on the path	**83.1**
The length of the path	**72.5**
The traffic density of the path	**51.1**
The cleanliness of the path	39.1
Number of traffic lights on the path	31.3
Appearance of traffic polices on the path	17.3
Other factors	8.1

On the survey, we also ask the participants for the personal opinions of the participants about the traffic system, their proposed solutions, as well as their attitude toward the public transportation system. Most of the participants claim that the traffic system in Hanoi is dysfunctional and should be improved as soon as possible. A number of reasons for the bad traffic situations as well as proposed solutions to deal with these problem are figured out. This shows that the traffic problems in Hanoi not only are the problems of the government but also involve people living in the city.

6.3 Parameter Estimation of the BPR Function

The traveling time function BPR is used widely in traffic assignment models, in particularly for the UE model, however, the parameters for the function are different according to many factors, such as the kinds of traffic systems, the included components of traveling time, etc. Therefore, the original suggested values of parameters of the BPR function may be not proper for applying to in a number of traffic systems. In this section, we investigate the parameters of the BPR function applying to the MTSDM in Hanoi. The new BPR function consists of not only the driving time, but also the waiting time at the signal line on the link. The new BPR function, denoted as NBPR, is evaluated by comparing with the BPR function using original suggested parameters, and with the simulated time in the traffic simulation software,

named VISSIM—a microscopic multi-modal traffic flow simulation software package developed by Planung Transport Verkehr AG (PTV group) in Karlsruhe, Germany.

6.3.1 Link Capacity in Motorcycle Unit

Since we consider the MTSDM in Hanoi where motorcycles occupy about 75% total number of vehicles and all kinds of vehicles travel in the same lanes, it is reasonable to define the traffic flow unit as Motorcycle Unit (MCU) rather than Passenger Car Unit (PCU). In order to estimate the practical capacity of a given link in an MTSDM, we use the certain **saturation flow** of the link that is investigated by Hien and Montgomery in [38]. In the research, they also use MCU as the flow unit and a regression model to develop a formula of saturation flow within 4 seconds of the green phase of the signal light on the link. The formula of the saturation flow of a link in Hanoi is given as follows.

$$4S = 12.08 + 2.13\,(W - 3.5) - 47.12\,\frac{Prt}{Rrt} - 36.15\,\frac{Plt}{Rlt}, \tag{6.1}$$

where S is the average number of out-going vehicles in 1 second of the green phase, W is the width of the link in meters, $\frac{Prt}{Rrt}$ and $\frac{Plt}{Rlt}$ are the proportions between the number of turners and the turning radius on the right side and on the left side, respectively. In one cycle of the signal light, the maximum number of vehicles can go though the link is the number of vehicles go though the signal line at saturation flow in the green phase, thus, the practical capacity of a link can be calculated as

$$C_p = \frac{t_g \cdot S}{t_g + t_r}, \tag{6.2}$$

where t_g, t_r are the green and red times of the traffic light, respectively, and C_p is the practical capacity flow, i.e., the maximum vehicles can go through the link in one second without congestion.

In order to evaluate the proposed practical capacity formula in (6.2), simulation software VISSIM [23, 51] is utilized. Some common kinds of links in Hanoi are simulated. The traffic behaviors of drivers in Hanoi are also imitated in the software. The **simulated capacity** of a link is measured as the maximum entering flow such that no congestion is observed. This is done by adjusting the entering flow to the link. For example, at first, we run the simulation with the entering flow of 50 cars and 100 motorcycles. The entering flow is increased by 5 cars and 15 motorcycle each time. The adjustment terminates when congestion occurs, and the entering flow of the previous simulation is the simulated capacity of the link.
The results indicate that the differences between the predicted capacity and the simulated capacity are in the range $[-2.67\%, 8.70\%]$. These small differences prove that the formula in (6.2) is reliable, and it is used in the forthcoming sections.

6.3.2 Parameters Setting for the BPR Function

As mentioned in Chapter 2, the BPR function is given as

$$t(x) = T_0\Big(1 + \rho\,(\frac{x}{C_p})^{\beta}\Big),$$

where ρ and β were suggested by BPR engineers to be 0.15 and 4, respectively, without any explanation. The suggested parameters were questioned by some researchers. In [63] Steenbrink applied the BPR function with new parameter values $a = 2.62$ and $b = 5$, and he indicated that the new set of parameters is most suitable for the BPR function. However, in the paper Steenbrink used the steady capacity C_s instead of practical capacity, and the data was collected in some regions with low ratio $\frac{x}{C_s}$ of flow and capacity. Nevertheless, Florian and Nguyen [24], showed that the original BPR function with $a = 0.15$ and $b = 4$ generally gives a better traveling time estimation than the one proposed by Steenbrink.
In [62] Spiess formulated the conical volume-delay function as

$$t(x) = T_0\left(2 + \sqrt{a^2(1-y)^2 + b^2} - a(1-y) - b\right)\ , \tag{6.3}$$

where a is the parameter of the function, $y = \frac{x}{C_p}$ and $b = \frac{2\cdot a-1}{2\cdot a-2}$. The formula of the conical function is different from the formula of the BPR function, but it still has all the advantages of the BPR function. It even has one more advantage that the computation of the function without logarithms is easier than those of the BPR function. With all of its advantages, the conical function is also used widely in traffic planning to estimate the traveling time on links.
Figure 6.9 shows the traveling times estimated by the BPR function and the conical function using various sets of parameters. It is clear that, when the ratio between the flow x and the capacity C_p of the link is smaller than 1, both functions have close values, but when this ratio is greater than 1 the conical function seems to be closer to the BPR function with $\beta = 4$ than to the BPR function with $\beta = 5$. This refers that the BPR function, in general, predicts better the traveling times with $\beta = 4$ than with $\beta = 5$. This is in agreement with the paper of Florian and Nguyen mentioned above. Respecting to these results, we fix the value $\beta = 4$ for the BPR function, and the function becomes

$$t(x) = T_0\left(1 + \rho(\frac{x}{C_p})^4\right), \tag{6.4}$$

where T_0 and ρ are two parameters being calibrated.
We consider a link in an MTSDM with the length L and the maximum allowed speed on the link V_0. The driving time is denoted as t^d, and the average waiting time is denoted as t^w. Assume that one cycle of the traffic light consists of two phases: the

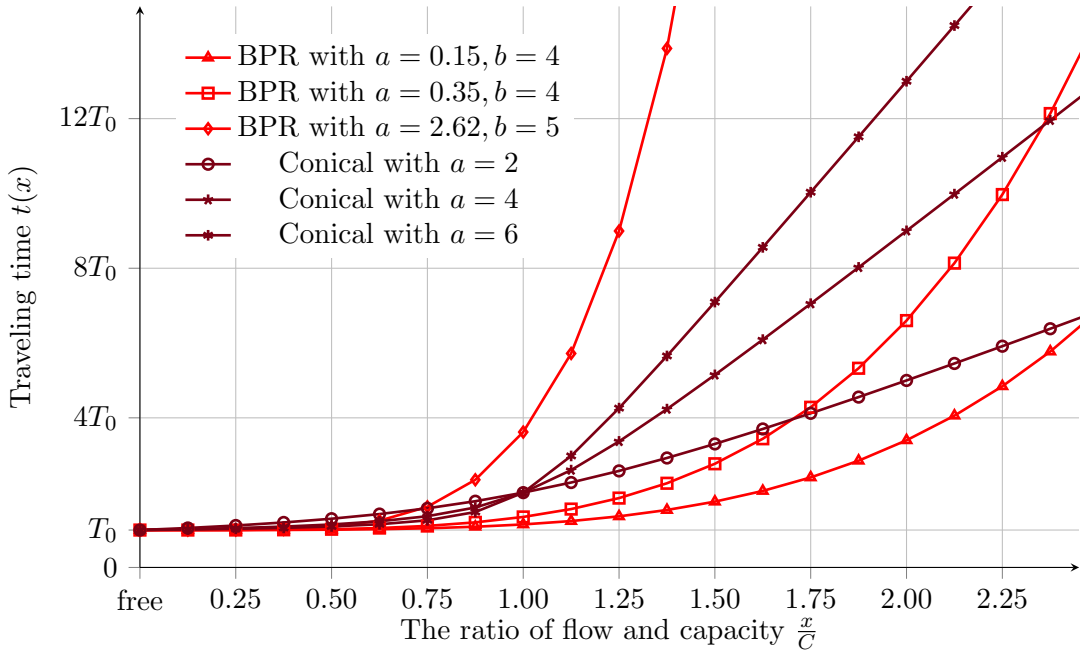


Figure 6.9: A comparison between the BPR and the conical functions using different parameters.

green phase with duration time t_r seconds and the red phase with duration time t_g seconds. The yellow phase is normally quite short and it is added to the previous phase (red or green). The signal cycle starts with the red phase at time 0. The variable t is the time when the vehicle arrived at the signal line. We consider two cases when the flow is free and when it is equal to the capacity of the link.

At free flow: vehicles are not influenced by each other, thus they can run at the maximum allowed speed V_0. The the running time at free flow is

$$t_0^d = \frac{L}{V_0}. \tag{6.5}$$

The dependence of the waiting time at free flow on the arrival time t is described in Fig. 6.10 (a). If $0 \leq t < t_r$, the signal light is on the red phase. Because of the free flow, there is no queue of waiting vehicles, thus the considering vehicle can go out as soon as the light turns to green. Hence, the waiting time is equal to the time waiting for the light changing to green $t_0^w = t_r - t$. If the vehicle arrives at the green phase, i.e. $t_r \leq t < t_r + t_g$, the vehicle can go through without stopping, i.e. $t_0^w = 0$. The average waiting time at free flow is computed as follows

$$\begin{aligned} t_0^w(average) &= \frac{1}{t_g + t_r}\left(\int_0^{t_r} (t_r - t)dt + \int_{t_r}^{t^r + t_g} 0dt\right) \\ &= \frac{t_r^2}{2(t_r + t_g)} \ . \end{aligned} \tag{6.6}$$

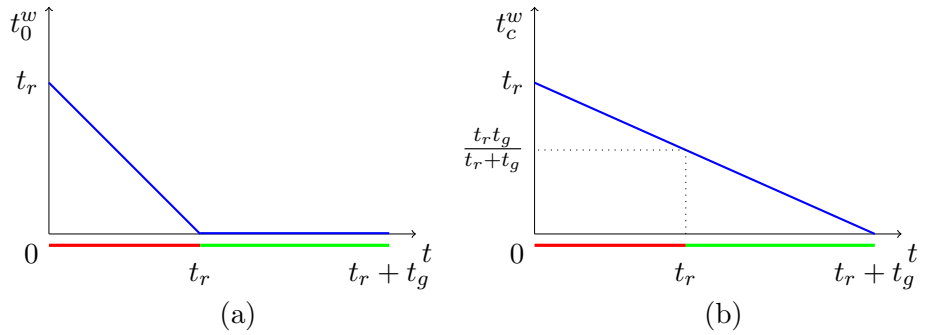


Figure 6.10: The waiting time of a vehicle w.r.t. arrival time (a) at free flow and (b) at capacity flow.

From (6.5) and (6.6), the total traveling time at free flow T_0 is described as

$$T_0 = \frac{L}{V_0} + \frac{t_r^2}{2(t_r + t_g)} \ . \tag{6.7}$$

At capacity flow: vehicles move smoothly on the link at a speed close to the maximum allowed speed V_0. However, the real driving distance is less than the length of the link L because of a waiting queue at the signal line. In our observation on the traffic in Hanoi, the driving time at capacity flow is similar to the driving time at free flow, i.e., $t_c^d = \frac{L}{V_0}$, however, the waiting time is much bigger than the waiting time at free flow. Figure 6.10 (b) describes the waiting time at capacity flow as a function of the arrival time t. Two cases have to be considered.

Case 1: $0 \leq t < t_r$. The signal light is now red, and the waiting time for the signal light turning from red to green is $t_r - t$. The number of vehicles in the queue is $t\,C_p$ (MCU) so the queuing time is $\frac{t\,C_p}{S}$. Replace C_p by the right side of (6.2) we have the queuing time is $\frac{t\,t_g}{t_r+t_g}$. So, the total waiting time in this case is

$$\begin{aligned} t_c^w(t) =& t_r - t + \frac{t\,t_g}{t_r + t_g} \\ =& t_r - t\frac{t_r}{t_g + t_r}. \end{aligned} \tag{6.8}$$

Case 2: $t_r \leq t < t_r + t_g$. The signal light is now green, but there is a queue of waiting vehicles which is entered the queue in the previous red phase. Thus, the waiting time in this case is the queuing time. Number of entered vehicle in the range $[0, t]$ is $t\,C_p$, however, the number of out-going vehicles in the range $[t_r, t]$ is $(t - t_r)S$.

Thus, the waiting time in this case is calculated as

$$\begin{aligned} t_c^w(t) &= \frac{t\,C_p - (t - t_r)S}{S} \\ &= t_r - t\frac{t_r}{t_g + t_r}. \end{aligned} \tag{6.9}$$

The right sides of (6.8) and (6.9) are the same, and they are the function of waiting time at capacity flow that is shown in Fig. 6.10 (b). The average of waiting time at capacity flow is calculated as follows

$$\begin{aligned} t_c^w &= \frac{1}{t_r + t_g}\int_0^{t_r+t_g}(t_r - t\cdot\frac{t_r}{t_r + t_g})dt \\ &= \frac{t_r}{2}. \end{aligned}$$

Therefore, the traveling time at capacity flow is

$$t_c = t_c^d + t_c^w = \frac{L}{V_0} + \frac{t_r}{2}. \tag{6.10}$$

According to the BPR function in (6.4), the traveling time at capacity flow can be computed as

$$t_c = T_0\left(1 + \rho(\frac{C_p}{C_p})^4\right). \tag{6.11}$$

This implies

$$\rho = \frac{t_c}{T_0} - 1. \tag{6.12}$$

From (6.7), (6.10), and (6.12), the parameter a can be calculated as

$$\rho = \frac{t_r t_g}{t_r^2 + 2\frac{L}{V_0}(t_r + t_g)}. \tag{6.13}$$

Replacing T_0 and ρ in (6.4) by the right side of (6.7) and the right side of (6.13), respectively, we have the new BPR traveling time function for the traffic system in Hanoi as follows

$$t(x) = \left(\frac{L}{V_0} + \frac{t_r^2}{2(t_r + t_g)}\right)\left(1 + \left(\frac{t_r t_g}{t_r^2 + 2\frac{L}{V_0}(t_r + t_g)}\right)\left(\frac{x}{C_p}\right)^4\right), \tag{6.14}$$

where C_p is the practical capacity of the link is calculated as (6.2).

We have presented the method of parameter calibration for the BPR function applying to the traffic in Hanoi, however, the method can be apply to other traffic systems.

6.3.3 Computational Results

Table 6.2 shows the values of the parameter ρ for a number of common kinds of links in Hanoi calculated as formula (6.13). These values range from 0.050 to 0.286. In most of the links, these value differs substantially from the default fixed value 0.150 suggested by the BPR engineers in [9]. The columns are divided into two groups with the same headers. The meanings of the columns in each group are as follows. The column labeled "$\frac{L}{V_0}$ (s)" indicates the running time in second of a single vehicle on a link with the length L at the maximum allowed speed V_0. The columns labeled "t_r (s)" and "t_g (s)" are the time in second of the red phase and the green phase, respectively. The column, labeled "T_0 (s)", indicates the results of the average traveling time (in second) on the link and the last column, labeled "ρ", shows the results of the estimation of parameter ρ.

Table 6.2: Values of parameter ρ for some common links in Hanoi.

$\frac{L}{V_0}$ (s)	t_r (s)	t_g (s)	T_0 (s)	ρ	$\frac{L}{V_0}$ (s)	t_r (s)	t_g (s)	T_0 (s)	ρ
30	60	30	50.000	0.200	90	60	30	110.000	0.091
30	45	45	41.250	0.273	90	45	45	101.250	0.111
30	30	60	35.000	0.286	90	30	60	95.000	0.105
30	40	20	43.333	0.154	90	40	20	103.333	0.065
30	30	30	37.500	0.200	90	30	30	97.500	0.077
30	20	40	33.333	0.200	90	20	40	93.333	0.071
60	60	30	80.000	0.125	120	60	30	140.000	0.071
60	45	45	71.250	0.158	120	45	45	131.250	0.086
60	30	60	65.000	0.154	120	30	60	125.000	0.080
60	40	20	73.333	0.091	120	40	20	133.333	0.050
60	30	30	67.500	0.111	120	30	30	127.500	0.059
60	20	40	63.333	0.105	120	20	40	123.333	0.054

In the first evaluation of the new function, VISSIM were again used to simulate traffic on a link in Hanoi. In the simulation, the entering flow were varied from free flow to 2 times of the link capacity, i.e., $0 \leq \frac{x}{C_p} \leq 2$, since these values are adequate for the real traffic situation in Hanoi. Figure 6.11 shows the simulated time and the predicted time by the new BPR function (using the new proposed parameters) and by the original BPR function (using original parameters) of a link in Hanoi.
In more details, Table 6.3 shows the gaps between the traveling times predicted by the two BPR functions, i.e. the original BPR function and the new BPR function and the average simulated traveling times on VISSIM. The results shows that the new BPR function can estimate approximately the total spent times on a link in an MTSDM and is significantly better than the original BPR function.

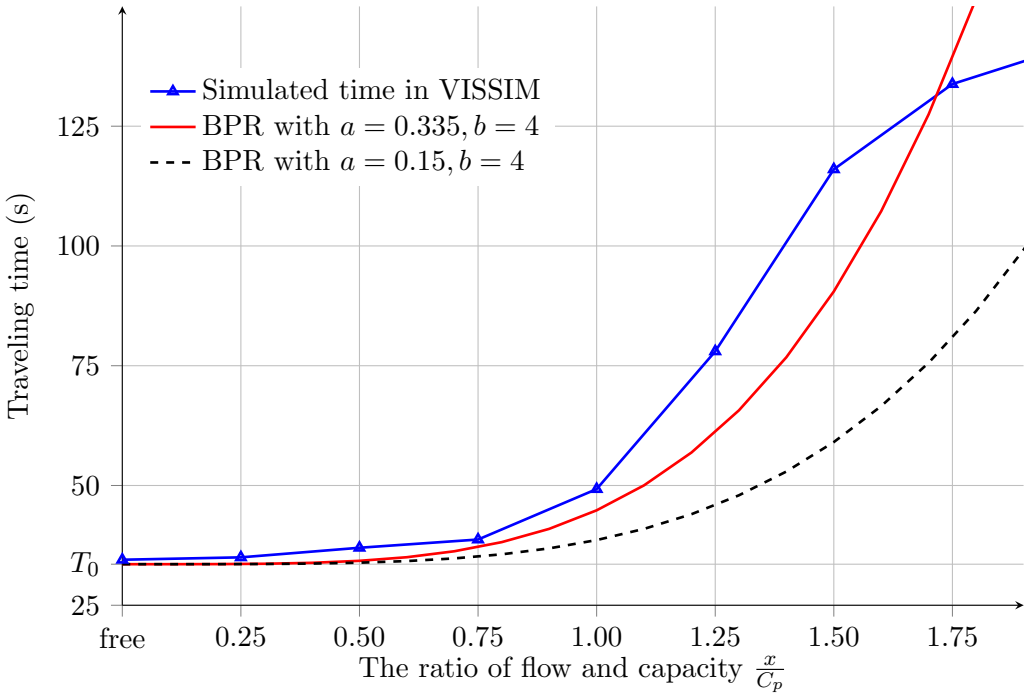


Figure 6.11: The predicted times by the original BPR function, the new BPR function, and the simulated times on VISSIM.

Table 6.3: Gaps between BPR functions and the simulated time.

Flow/Capacity	**New BPR** (%)	**Original BPR** (%)
free flow	2.70	2.70
0.25	3.97	4.03
0.50	7.38	8.43
0.75	4.11	9.19
1.00	8.99	21.60
1.25	21.79	41.20
1.50	21.98	49.08
1.75	3.89	39.61

6.4 Running the GUEM Model in Hanoi

The new traffic assignment model GUEM, investigated in Section 5.3, was examined in Hanoi with the collected data and the new BPR function. The examination is introduced in two subsections. While Subsection 6.4.1 presents notes on data preparation for the examination, Subsection 6.4.2 shows the computational results of the new model.

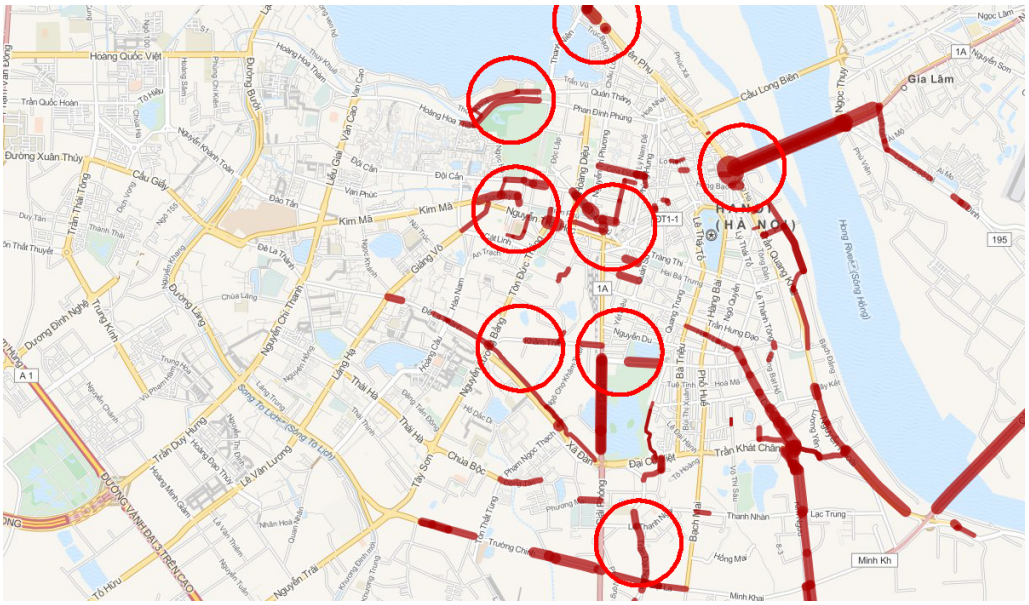
6.4.1 Data Preparation

We extracted the traffic demand in one hour between 7 AM to 8 AM—the most crowded time interval according to the trip distribution shown in Fig 6.4. The number of trips in this time interval is 23.48% the total number of trips within a day. We created two O-D matrices in this interval: one for motorcycles and the other for personal cars. The parameters of the BPR function on each link in the traffic system in Hanoi were calculated as mentioned in the previous section, i.e. Section 6.3. The lengths of links were calculated according to the longitude and the latitude of the nodes on the link. The width of a link was determined by the attribute "Width" of the link—contributed by users. For links without a given data of width, we estimated the link's width by its type, e.g. a primary highway has a default width of 12 meters. However, for most of main links in Hanoi, we estimated the width by using the "rule tool" of Google Earch application. The types of links were also used to determine the maximum allowed speed on the links. The times of the traffic signal phases and other information was also updated. They were set to the default values if no information is available.

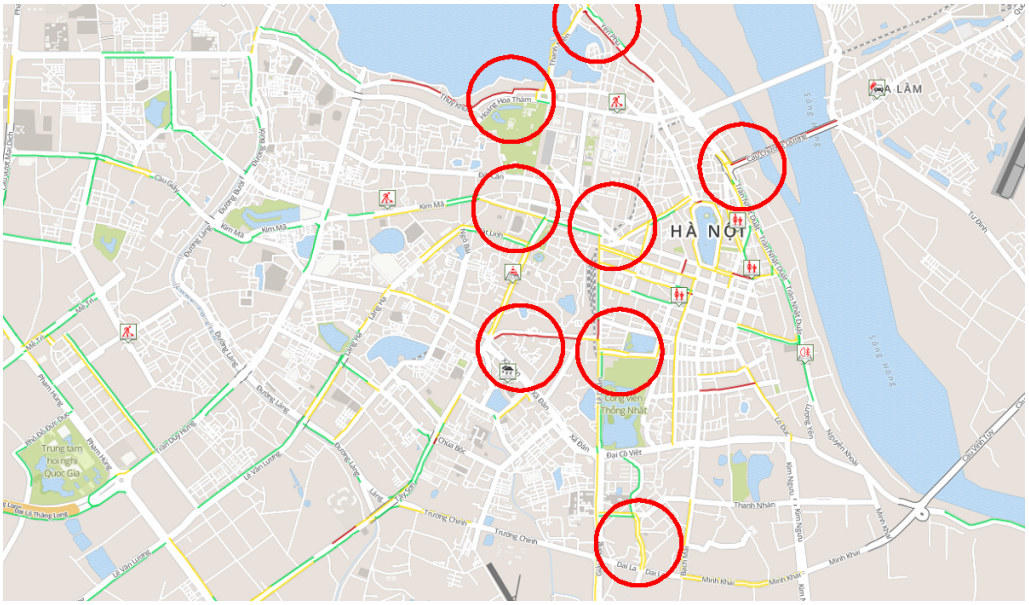
For each O-D pair of population zones, we defined a set of potential paths as the set of the 15 dissimilar shortest paths with the minimum dissimilarity 15% and an upper bound on the length as two times the the shortest length. It means that we computed the 15 shortest loop-less paths, such that the dissimilar between two of them is not less than 15% and the length of a determined path is not longer than two time the shortest length.

6.4.2 Computational Results

In order to evaluate the accuracy of the GUEM model on the traffic system in Hanoi, we analyzed the predicted flows by the model to figure out the most crowded links with respect to the ratio of the predicted flows and the capacity of the links. The information about these links is compared with the data of locations with a high frequency of traffic congestion.
Figure 6.12 shows the agreement between the predicted traffic flows and the real traffic situation provided by the Remon-hanoi project [59]. In Fig. 6.12 (a) the width



(a) Predicted traffic flows by the new UE model.



(b) Real traffic situation provided on the website of the Remon-hanoi project.

Figure 6.12: Predicted traffic flows in comparison to a real situation in Hanoi

of the dark red bar on a road is directly proportional to the ratio of the predicted flow to the capacity $\frac{x}{C_p}$. The visualization is on the TranOpt Plus software that is introduced in Chapter 7. Fig. 6.12 (b) is a real traffic situation of a common working day provided on the website remon-hanoi.net, see [59]. The green bars, yellow bars and dark-red bars on roads indicate that the speed levels of vehicles in those areas are a little bit slow, slow and very slow, respectively. The locations of roads with yellow and dark-red bars are marked with red circles for comparison with the locations of roads with a high probability of congestion predicted in Fig. 6.12 (a). The high agreement of the two sets of locations proves the accuracy of the new model GUEM on the traffic system in Hanoi.

6.5 Conclusion and Discussion

In this chapter, we have presented the whole project of traffic assignment modeling in Hanoi, Vietnam including data acquisition, model setting, and model implementation. In data acquisition, both the geographic data and traffic data are investigated from available resources. Furthermore, the results of the online survey on traffic behaviors in Vietnam with 316 participants are also introduced.

In order to run the new proposed model GUEM on the traffic system in Hanoi, we calibrated the parameters for the BPR function on the links in the Hanoi traffic system. The computational results showed that the new BPR function—the BPR using new set of parameters—can predict better the traveling time on a link in the traffic system of Hanoi than the original BPR function can.
The potential paths for cars and motorcycles from each O-D pair of zones were defined as the set of 15 dissimilar shortest loop-less paths, where the minimum dissimilarity between two paths is 15% and the upper bound on the length of a path is two times the shortest length. The GUEM model was run with all the collected data and gave the results, that were in big agreement with the real traffic situation.

The promising results of the GUEM model open a number of further developments. First, we are going to develop a dynamic user equilibrium model for mixed traffic systems based on the GUEM model where the traffic demand are dynamic, i.e. it can change dynamically. A stochastic user equilibrium model for mixed traffic systems is also in consideration for further developments of the GUEM model. Because all of the UE models depend significantly on the set of potential paths for each O-D pair of zones, we are going to improve the method of determining the potential paths from each pair of zones. This can be done by adding more constraints on routing behavior into the multi-objective shortest paths problem, e.g., when drivers are guided by a traffic information system. We also plan to test the GUEM model not only in Hanoi but also in another major cities in developing countries where the traffic is characterized by a mixture of 2-wheel and 4-wheels vehicles.

Last but not least, the running time of the GUEM model is also in our consideration to improve, since it takes about 4 minutes to run the model on a transportation network with approximate 4000 nodes and 10000 links on a personal computer with a 2 GB of RAM and a 2-GHz dual-core processor. In various traffic planning problems, e.g. network design, we may have to repeat the GUEM model a number of times, thus it is reasonable to keep improving the running time of the model.

Chapter 7

Software: TranOpt Plus

One of the main contributions of this thesis is a software, named TranOpt Plus, that provides a framework for map-based problems, such as traffic planning and routing problems. The current version, i.e. TranOpt Plus 1.0.2 released on March 12, 2015, includes three major features: map editing, dynamic routing, and traffic assignment modeling. All the implementations of the routing algorithms, mentioned in Chapter 4, and the traffic assignment models, mentioned in Chapter 5, are available in TranOpt Plus. In addition, further map-based applications can be easily embedded into TranOpt Plus.
The outline of this chapter is as follows. Section 7.1 overviews TranOpt Plus while Section 7.2 introduces the major features of the software. In Section 7.3, we present a number of the open-source libraries, used in TranOpt Plus. The traffic-network graph algorithm (TNGA) is introduced in Section 7.4, and the software packing process is described in Section 7.5. The last section, i.e. Section 7.6, is for conclusion and discussions.

7.1 Overview

The motivation for the development of TranOpt Plus comes from a fact that there are a large number of map-related problems needing visualization framework. For instance, the results of routing services or traffic assignment modeling on large maps could be challenging for users to understand and to evaluate if they (the results) are not visualized. Various open-source libraries for graph visualization are available, such as Open Graph Drawing Framework (OGDF), Gephi, and Graph Visualization (Graphviz). They can be obtained from the websites www.ogdf.net, gephi.github.io, and www.graphviz.org, respectively. However, they are more or less supporting general kinds of graphs while we focus on transportation networks, that can easily import and export available open-source data, e.g. from OpenStreetMap. Furthermore, available open-source packages are normally large with many libraries, that

are not in our interests. Thus, we investigate a possible small visualization software, that is able to cope with large graphs, such as transportation networks of cities. The software is named **TranOpt Plus**, and it consists of two major components: libraries for mathematical algorithms and libraries for visualization.

TranOpt Plus is programmed in C++11, that is the version released on August 12th, 2011 of the standard C++ programming language. In order to handle the code files, we use the free version of the Qt Creator—a cross-platform integrated development environment provided by the Qt Company. There is also a commercial version, however, it is out of the scope of our project. TranOpt Plus has been developed since December, 2013. The first version was released in March 2015, and the current version, i.e. TranOpt 1.0.2, was released on September 15th, 2015. The software currently supports only the Windows platforms, and it was tested on Windows 7 and Windows 8. Figure 7.1 is a screen shot of TranOpt Plus showing determined paths by the dynamic routing feature. Map objects, e.g. nodes and links, are displayed on the background made of map image layers, that are imported from the OpenStreetMap server.

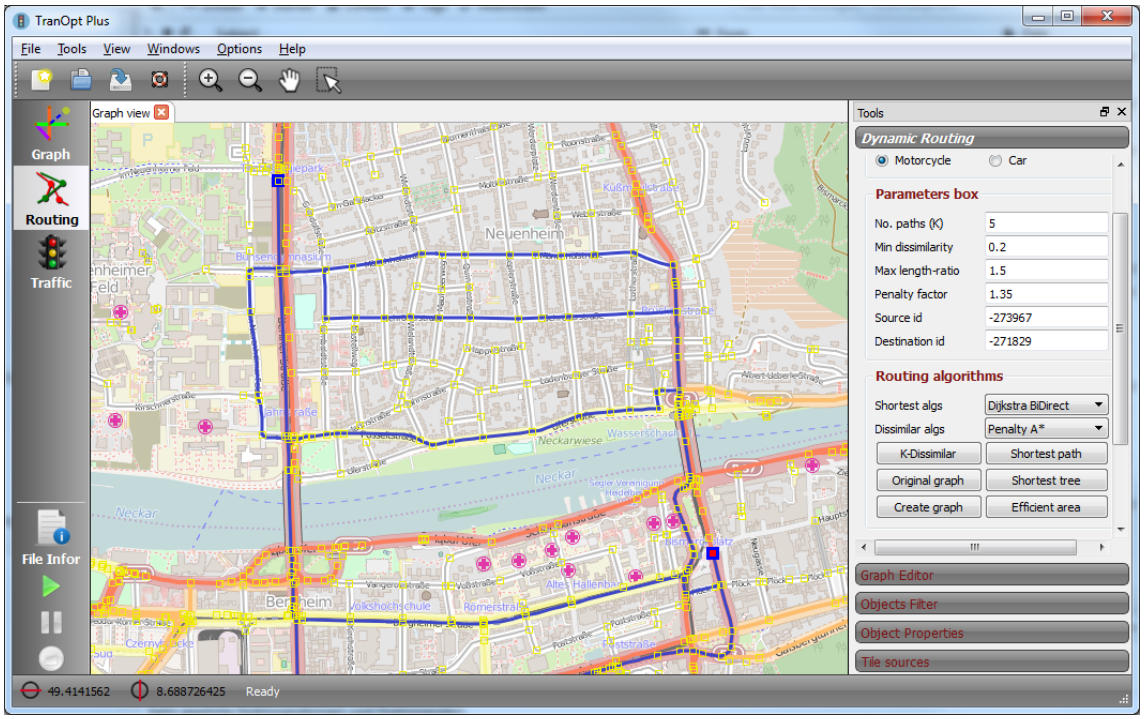


Figure 7.1: The interface of TranOpt Plus.

The source code of TranOpt Plus can be divided into two major parts: the problem-implementations and the problem-visualization. The former part is pure C++ and independent source code. They are packaged in the TNGA library. All the mentioned mathematical problems in this doctoral thesis, i.e. routing problems and traffic assignment models, are formulated and solved in the TNGA library. The

later part, i.e. visualization, consists of the source code for visualizing the problems, such as graphical user interface (GUI), map visualization, and traffic assignment illustration.
Table 7.1 shows the major libraries used in TranOpt Plus. Excluding the TNGA library, there are three major open-source libraries namely Qt library, Mapgraphics library and QCustomPlot library. The later two are also developed from the free version of the Qt library. The MapGraphics library supports for adding a number of map layers to applications with abilities of zooming, rotating, and customizing map objects. The libraries are developed by the user "Raptorswing" on GitHut - a web based version control system for software development. We use the version of the MapGraphics library released on March 1st, 2015. QCustomPlot is a Qt, C++ library that provides a number of tools for plotting and for data visualization. There are two versions of the QCustom Plot library: the free version (default version) is licensed under GNU General Public License, i.e., the library can be redistributed under GNU General Public License. It was developed by E. Eichhammer and published on the website www.qcustomplot.com. More details of these libraries are presented in Subsection 7.3 and Subsection 7.4.

Table 7.1: Major libraries in TranOpt Plus software.

Name	**Comment**	**Author**
	Open source libraries	
Qt Library	The free version	The Qt Company
MapGraphics	Fixed and further developed	Raptorswing
QCustomPlot	Plot functions	Emanuel Eichhammer
	Developed libraries	
TNGA/Routing	Routing algorithms	N. T. Nam
TNGA/TA	Traffic Assignment models	N. T. Nam

In addition, all the icons used in TranOpt Plus were originally imported from open resources, that have GNU General Public License. Some of them were customized and redistributed under the same license.
The general working architecture of TranOpt Plus is illustrated in Fig. 7.2 with three levels. The top level is the graphical user interface where users can edit maps, enter data, and see the results. At this level, users do not know about all the things behind, such as the problem formulations or the implementations of algorithms, etc. The second level is the converter, that reads the entered data, check the validity, and transforms them into pure mathematical problems. For instance, when users enter the source node and the destination node for the dynamic routing feature, the converter will convert the current map into a graph and call a suitable function in the TNGA library to solve it. The lowest level is the implementations of the TNGA

library where all mathematical problems are solved and sent the results back to the GUI level.

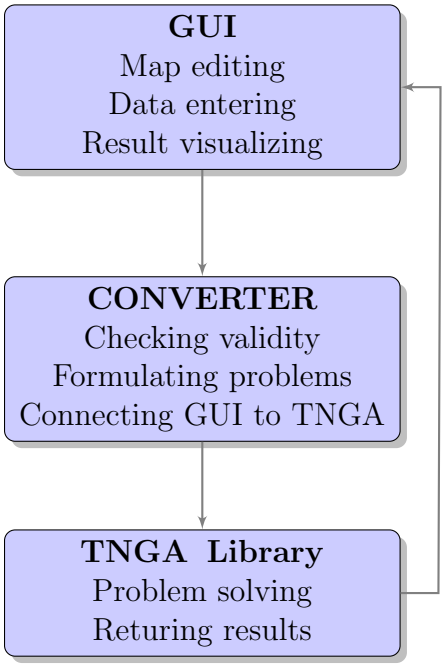


Figure 7.2: The working architecture of TranOpt Plus.

7.2 Main Features

The current version of the software, i.e. TranOpt Plus 1.0.2 released on September 15th, 2015, has three major groups of features: map editing, dynamic routing, and traffic assignment modeling. In this section we introduce these major features of the software and the other features are presented in the next section.

7.2.1 Map Editing

TranOpt Plus provides a number of tools for importing, creating, editing, and exporting transportation maps. Maps are stored in the file format, named **Extensible Markup Language** (XML), that is a markup language for encoding documents in a human-readable and machine-readable format. The data structure of graphs in TranOpt Plus is the similar as the data structure of OpenStreetMap (OSM) files. Therefore, TranOpt Plus can import a map in the OSM file as well as export the current map to a OSM file. The details of the map data structure used in TranOpt Plus are described in Appendix A with examples.
The simplest method to create a new map of an area on TranOpt Plus is to import an OSM file of the area from the OSM server on osm.org. The imported OSM file may

contain a large number of unnecessary objects, thus the function "Save as" is used to save the imported OSM file as a new OSM file, that contains only the map objects of the area. Users can also create new maps without importing existing maps. The first step is to locate the area by using a number of provided tools, such as zoom out, zoom in, and hand-hold. The software also provides various kinds of maps on the background, such that users can easily locate an area. Another facility supporting for locating an area is the "Location Search" tool, that has the interface shown in Fig. 7.3. Users can locate an area by giving the GPS coordinates of the area, or by selecting a country on the list of countries. GPS stands for Global Positioning System, that is operated and maintained by the U.S. Air Force. It provides the latitude and the longitude of positions on the Earth. The list of countries and their average GPS positions were edited from the collection of Frank Donnelly, that was mostly extracted from the NGA's GEOnet Names Server (GNS) and from USGS Geographic Names Information System (GNIS).

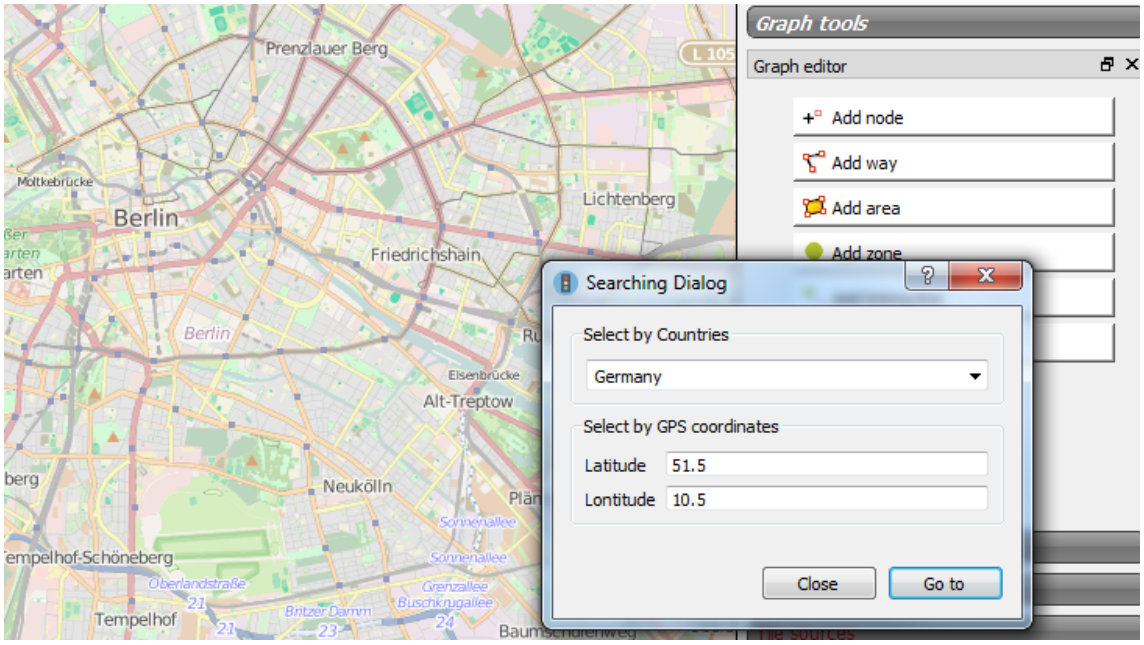


Figure 7.3: The "Location Search" tool of TranOpt Plus.

With the "Graph editor" tools, users can add new map (graph) objects, such as nodes, ways, areas, zones, and linking edges. Each map object has a number of internal and external attributes. The internal attributes are fundamental attributes, such as the longitude, the latitude, and the visibility status. All the internal attributes cannot be removed, however the values of those can be edited. All the map objects of the same kind have the same list of internal attributes. The external attributes of an object are defined by users for their purposes. TranOpt Plus provides tools for customizing the external attributes of the selected object, thus TranOpt Plus is able to work with various kinds of problems using different kinds

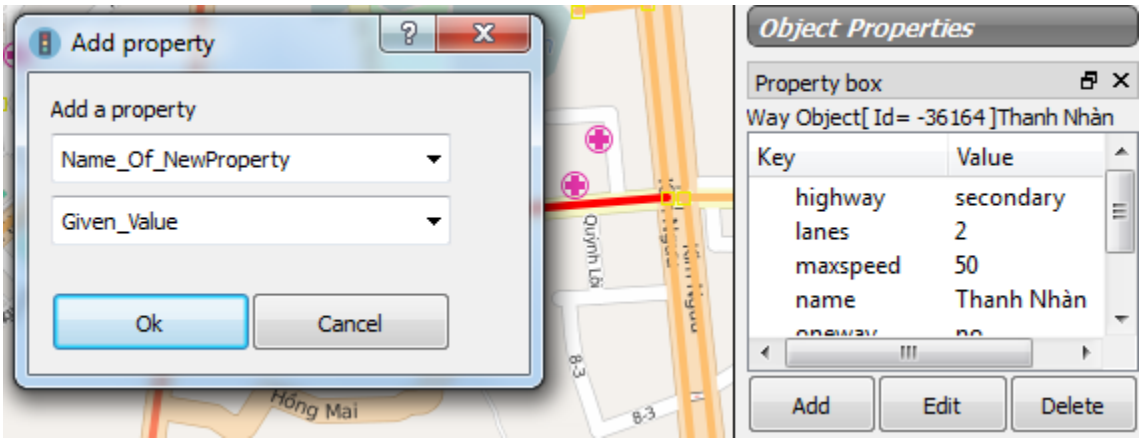


Figure 7.4: The interface of "Adding properties" tool for the selected object.

of data. For instance, in traffic assignment modeling, the capacity of a link is determined by the external attributes "highway" and "width". The assigned flow on a link is stored in the attribute "flow" of the link. Figure 7.4 shows the interface of the "Add property" feature of TranOpt Plus. Users can create new attributes and give a name and a value for each new defined attribute, or user can editdelete existing attributes.

Table 7.2: Notable attributes of the basic map items in TranOpt Plus.

Item	**Attributes**	**Possible values**
Node	"is_bendpoint"	yes (1) or no(0)
	"intersection"	yes or no
Zone	"tm_zone""	name of the zone
	"order"	order of the zone
Way	"highway"	trunk, primary, secondary, tertiary, residential, living street, unclassified
	"one_way"	yes or no
	"ban_car"	yes or no'
	"ban_motor"	yes or no
	"width"	width of the way in meter
Link	"flow"	the assigned flow
	"capacity"	the capacity of the link
Linking line	"tm_connector"	yes or no
	"highway"	NOT_HIGHWAY_DEFINED
Area	"tm_zone_boder"	yes or no

Table 7.2 shows a number of notable external attributes of the basic map objects

in TranOpt Plus. Some attributes are available in every object of a kind of map object, e.g. every way must have the "highway" attribute and its value should be one of the given values, i.e. trunk, primary, secondary, tertiary, residential, living street, and unclassified. In contrast, there are also external attributes, that may exist in a number of objects and may not exist in other objects of the same kind. For instance, "is_bendpoint" is an external attribute of node objects, however, some nodes do not contain the attribute explicitly, those nodes receive the default value of the attribute, i.e. false. Normally such attributes have boolean values, i.e. yes (1) or no (0).

7.2.2 Dynamic Routing

The **dynamic routing feature** provides a number of routing algorithms mentioned in Chapter 3 and in Chapter 4, e.g. Dijkstra's algorithm, A-Star algorithm, Eppstein's algorithm, the HELF, Yen's algorithm, and Martins' algorithm.

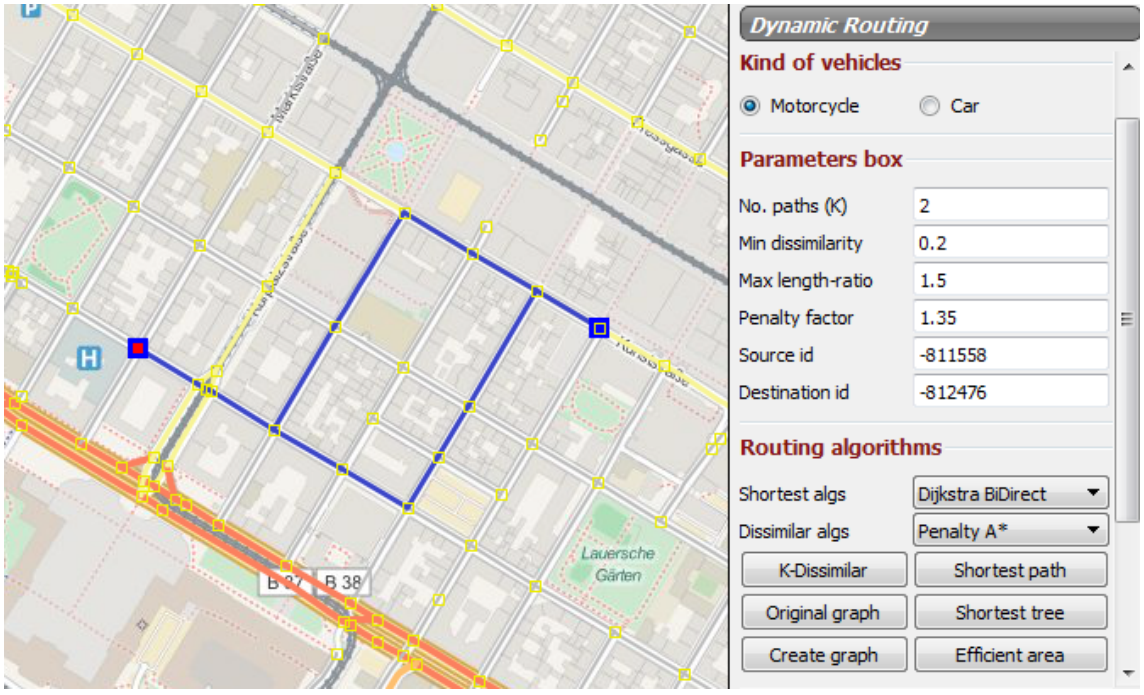


Figure 7.5: Finding dissimilar shortest loop-less paths on TranOpt Plus.

The interface of the "Dynamic routing" feature is shown in Fig. 7.5. There are two routing options corresponding to the two popular kinds of vehicles: motorcycles and cars. The routing map is prepared, such that it contains only the links, which vehicles of the selected kind are allowed to travel on. The attributes "ban-car" and "ban-motor" on links indicate which kinds of vehicles are not allowed to travel on the links. For example, if the value of the attributes "ban-car" is "yes" or "1", then motorcycles are not allowed to travel on the link. Links without both of these

attributes allow vehicles of all kinds to travel on. In order to find the most suitable route, a number of parameters should be given by users, e.g. the identification numbers of the source node and the destination node must be given on the text boxes named "Source id" and "Destination id", respectively. The identification number of a graph object is displayed on the status bar at the bottom of TranOpt Plus when the object is selected. According to the selected algorithm, different parameters are required in advance. The text box named "No. paths (K)" is for giving the number of paths, that users want to determine. The text box named "Min dissimilarity" is for the minimum dissimilarity between two paths, that is required for determining dissimilar shortest loop-less paths. The text box named "Max length-ratio" is for the upper bound on the length of the determined path and the length of the shortest path. For instance, if the value 1.5 is given to this parameter, then the selected algorithm will find only paths whose length does not exceed 1.5 times the length of the shortest path. If the penalty method is selected to find dissimilar shortest loop-less paths, then the penalty factor should be entered in the text box named "Penalty factor".

In the group box, named "Routing algorithms", there are a number of provided algorithms and routing functions. The combo box named "Shortest algs" provides algorithms for the shortest path (SP) problem, such as Dijkstra's algorithm and A-Star algorithm and their variations following bidirectional approach. The combo box named "Dissimilar algs" provides algorithms for the DSLP problem, e.g. Yen's algorithm, Penalty A-Star, the HELSF. Each push button executes a certain function. The button named "K-Dissimilar" determines the dissimilar shortest loop-less paths from the source node referred by the given identification number to the determined destination node using the selected algorithm with the given parameters. The button named "Shortest path" determines the shortest path. The button named "Original graph" removes all the results of the previous execution and recovers the original map. The button named "Shortest tree" determines a shortest path tree from the given source or to a destination node. If both the source node and the destination node are given, then it determine a shortest path tree from the source node to all other nodes on the map. In order to determine the shortest path tree to a given destination node, the text box "Source id" must be empty.
The determined paths by the routing functions are marked in blue. The source node and the destination node are visualized as small squares with blue border and filled in red. The information area at the bottom of the dynamic panel displays the information on the execution, such as the running time, and the number of paths determined.

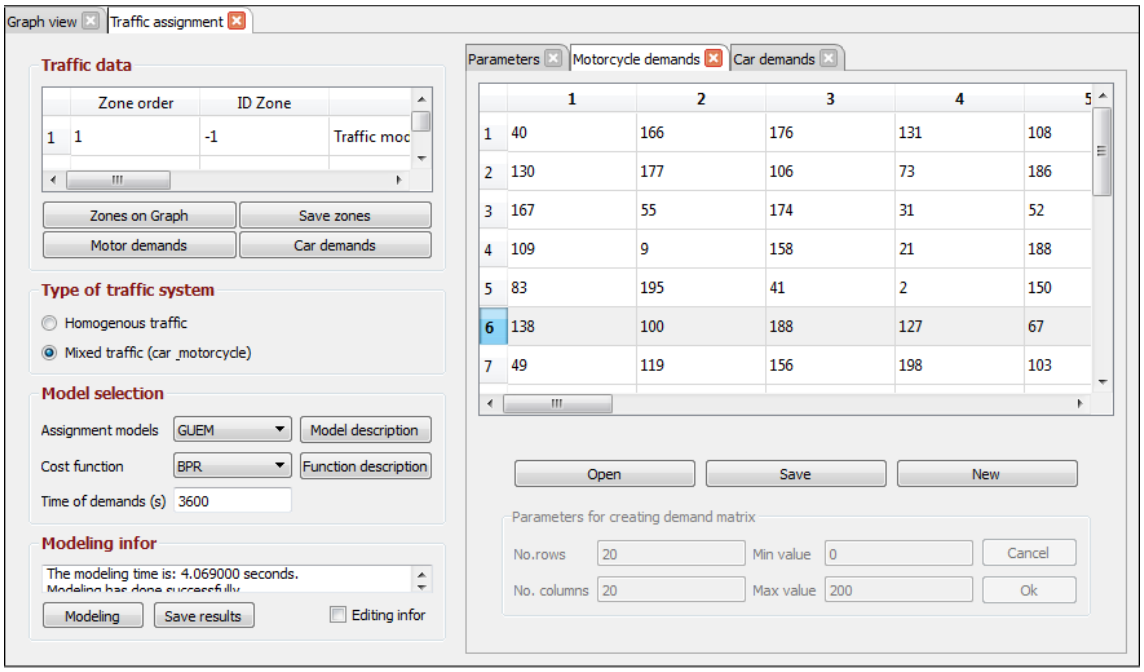


Figure 7.6: The interface for traffic assignment modeling feature.

7.2.3 Traffic Assignment Modeling

The **feature of traffic assignment modeling** provides tools to deal with TAM, such as entering input data, solving the selected model, visualizing the output, and analyzing the results. The current version TranOpt Plus 1.0.2 offers some TA models including the all or nothing (AON) model, the incremental (INC) model, and the general user equilibrium for mixed traffic systems (GUEM). Further models can be added easily.

Figure 7.6 is a screen shot of the modeling feature. The workflow of the feature is as follows. The first step is map preparation where a map containing population zones is require. Each zone must have at least two attributes namely "tm_zone" and "order". Note that the values of the attribute "order" of all zones must be in the sequence of integer numbers from 1 to the number of zones. This order is also the order of zones in the O-D matrices. All the attributes of map objects are provided and edited in this step, such that the map represents certainly the considering traffic system. The second step is to enter traffic data, i.e. enter the O-D matrices. Each O-D matrix corresponds to the traffic demand of a kind of vehicle. The size of an O-D matrix is equal to the number of zones available on the current map. In the case a provided O-D matrix has the size bigger than the number of zones, only the sub-matrix of it is used. Users can import the traffic demand from a traffic modeling demand (TMD) file, i.e. *.TMD, or enter it directly on the table corresponding to the matrix. The third step is to select the model and enter the required parameters. Depending on which model is selected, the corresponding panel of the parameters of the model is displayed. All the data and the entered parameters for the TA model are

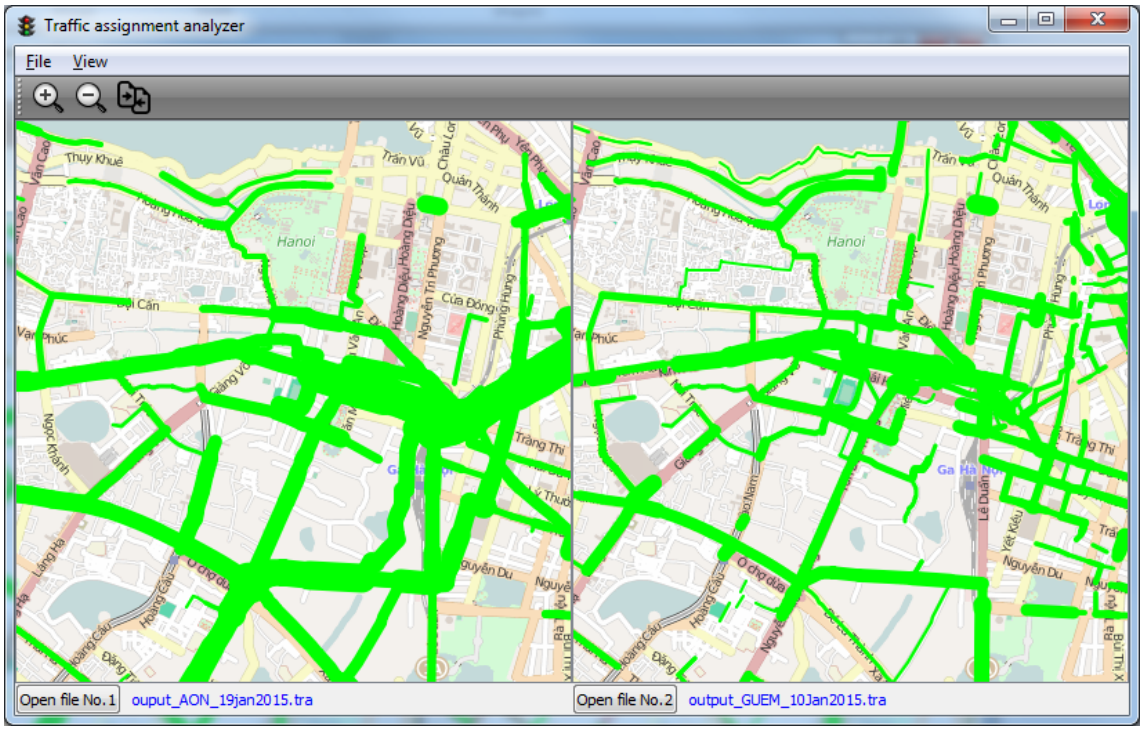


Figure 7.7: The results of the GUEM model and the AON model.

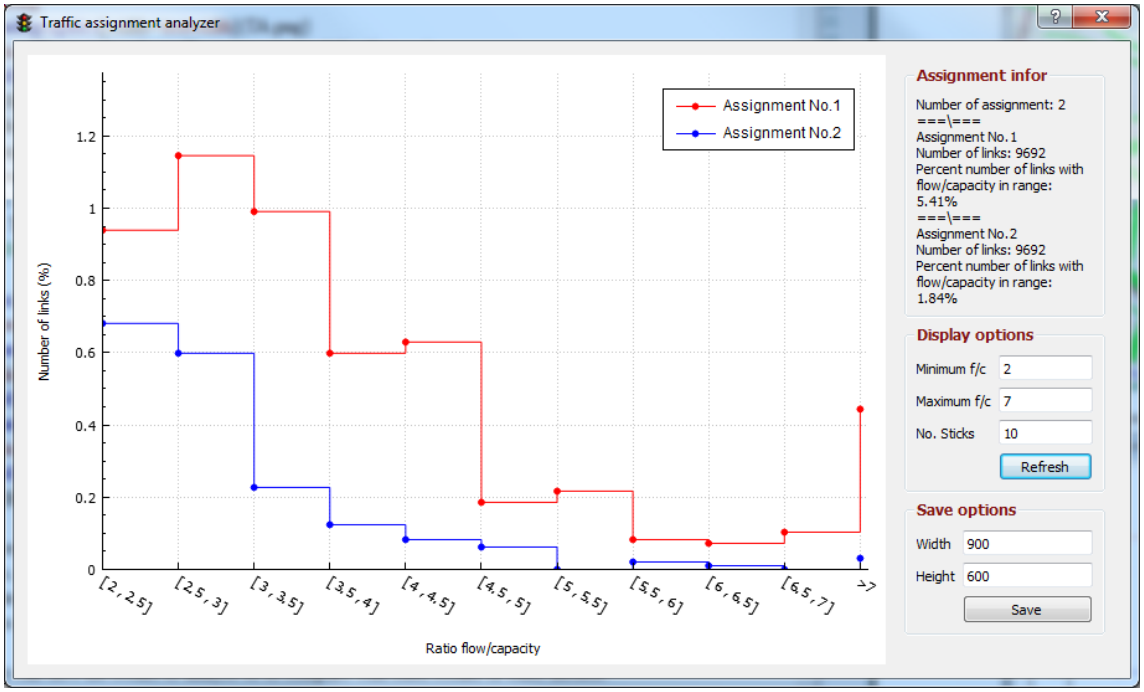


Figure 7.8: A comparison of the outputs of GUEM model and AON model.

collected and checked for validity by the converter. The model is then formulated and solved mathematically by a corresponding solver in the TNGA library. The predicted flows on links are assigned to the attribute "flow" of the links. They are displayed by color bars, e.g. green is the default color for the traffic flow. The width of a bar on a link indicates the volume of the predicted traffic flow on that link. For displaying the results of models, that contain a time sequence, i.e. models where the traffic demands are assigned in a number of time intervals, TranOpt Plus provides a playing control area at the left-down panel of the main form. In that case TranOpt Plus shows the assigned flows for every time interval one after another. Users can also save the predicted flows to analyze or to compare them with those by other models.
TranOpt Plus provides a number of tools for analyzing and comparing the predicted flows by different models. The tool named "**Comparing assigned flows**" with the interface in Fig. 7.7 displays the predicted traffic flows in Hanoi by the AON model on the left side and by the GUEM model on the right side. Due to this feature, users can observe the difference of the predicted flows on a particular area. In can be seen that the predicted flows by the AON model at the city center are significantly larger than those by the GUEM model. Users can also filter the links by the values of predicted flows or by the ratios between the predicted flow and the capacity of the links.
Another analyzing tool is the "**Assigned flows statistic**", that draws the statistics of the number of links whose the ratio of the predicted flow to the capacity is in a given range. For instance, Fig. 7.8 shows the statistic of the predicted flows of the AON model and the GUEM model on the traffic system of Hanoi, that are shown in Fig. 7.7. The statistic points out that the number of links with the ratio of the predicted flow to the capacity larger than 2 by the AON model is 5.4% the total number of links, while those by the the GUEM model is 1.84%. This reflects the characteristic of the AON model that drivers choose the shortest path (in length) to travel without considering the density of the traffic flow on the path. In contrast, in the GUEM model, drivers choose the paths by the traveling time, thus drivers avoid paths where the traffic flow is significantly larger than the capacity.

7.2.4 Supporting Features

Excluding the main features, TranOpt Plus provides also a number of supporting features, such as file information manager, map tile tool, save as tool, and customize tool.
The **"Filter tool"** is used for filtering the map objects according to either kinds of objects or traffic flows. In the filter by objects, users can show or hide objects in selected kinds. For instance, if the check box named "Show Node Objects" is disable, all the nodes on the current map will be invisible. Another filter is the

flow filter, that can hide or show link objects satisfying some constraint of flow and capacity.

The **Matrix tool** provides a number of basic matrix operators, e.g. plus matrices and minus matrices. The matrix tool is useful in preparing the O-D matrices. The **"Help tool"** consists of the manual of TranOpt Plus and the book of classes for software developers. The content for the help tool is stored as a **hyper text markup language** (HTML) file.

The **"Map tile tool"** is the tool for changing the back group of the current map. Each map tile is a fixed size square on the background. The current version TranOpt Plus 1.0.2 supports 4 kinds of map tile. The basic one is the "Grid Tile", that displays only the white color. Each kind of the rest display one kind of map image imported from the OSM server, i.e. MapQuest Aerial, MapQeustOSM, and OpenStreetMap. Users can also use a number of different kinds of map tile simultaneously, and can also change the opacity of each kind. Figure 7.9 shows the list of the kinds of map tile.

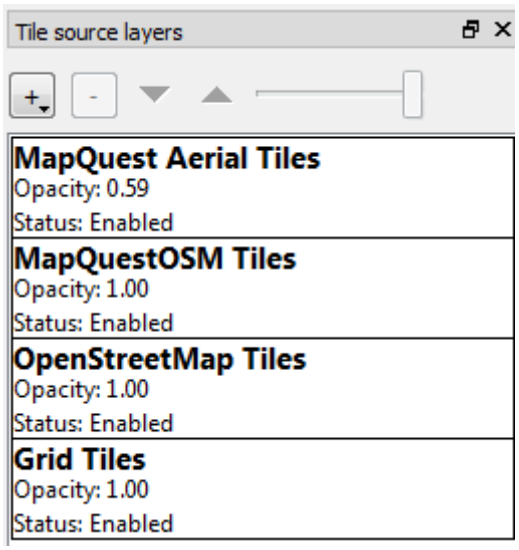


Figure 7.9: The widget for map sources controling.

The **"File information tool"** is the facility for file information management. The file information panel locates at the bottom of the left tool bar of TranOpt Plus. Figure 7.10 shows the information of the current map. It contains the map of Hanoi with the predicted flows by the GUEM model. Users can also edit the information of the file and save it for further usage.

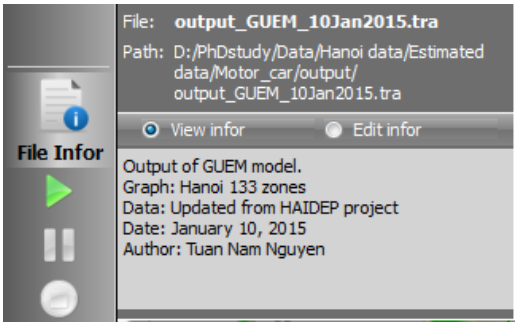


Figure 7.10: The file information management panel.

The **"Options tool"** can be categorized into two groups: the group of options for editing the displaying of map objects and the group of options for setting parameters for the traffic system. The display editing is the facility for changing the displays of the map objects. Users can change the color, the width of a kind of object, e.g. node, zone, link. The display width of each kind of highway (in pixel) can also changed. For instance, Fig. 7.11 shows the setting of the color of a selected link (edge). The traffic setting is the facility for setting a number of parameter for the traffic system, e.t. the maximum allowed speed (ms) on each kind of highway, the standard width (meter) of the highway.

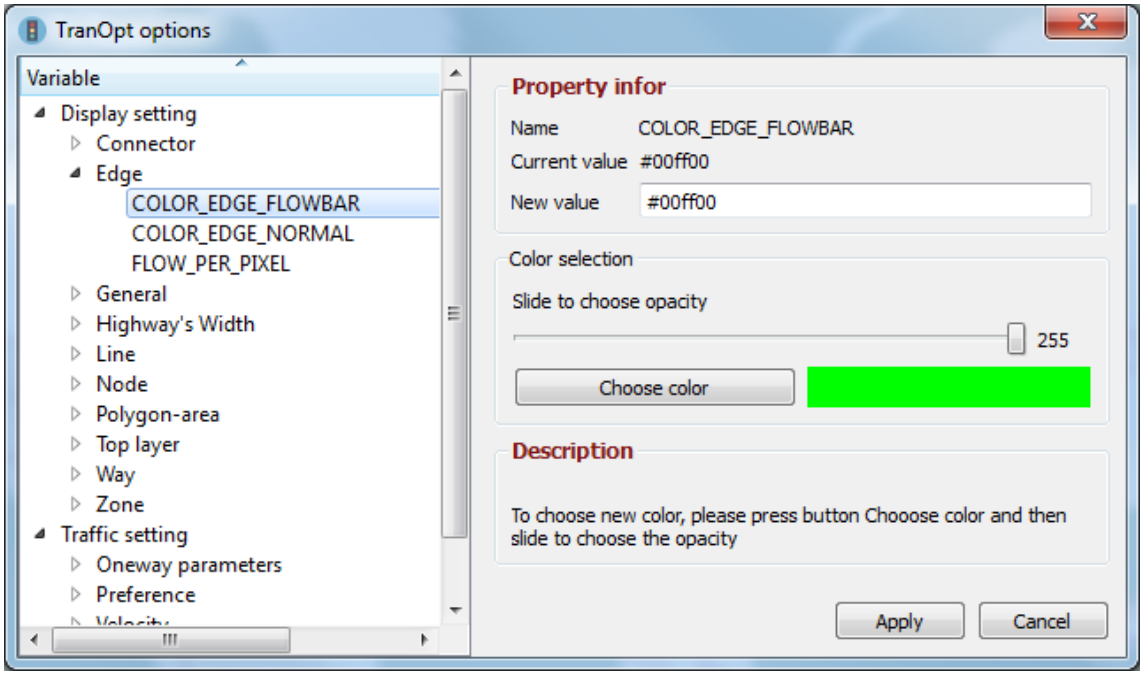


Figure 7.11: The "Options tool" of TranOpt Plus.

Figure 7.12: An image file exported by the "Save Scene As" feature.

The **"Save as tool"** is used for save the graph as a number given type formats, e.g. osm, xml. Further more, the whole map and the current view of the map can be saved as various formats, e.g. png format, or jpg format, or the pdf format. Figure 7.12 is the image file of the whole map of Hanoi, with 133 zones, exported by the "Save Scene As" function.
For convenience TranOpt Plus provides a number of **shortcut keys**, that are shown in Table 7.3.

Table 7.3: The list of shortcut keys of TranOpt Plus 1.0.2.

Shortcut keys	**Description**
	File menu
"Ctrl" + "N"	Create a new map (graph)
"Ctrl" + "O"	Open a map
"Ctrl" + "S"	Save the current map to the current file
"Ctrl" + "Shift" +"S"	Save the current map as different formats
"Ctrl" + "Alt" + "V"	Save current view to a file (*.png, *.jpg, *.pdf)
"Ctrl" + "Alt" + "S"	Save current scene to a file (*.png, *.jpg, *.pdf)
"Ctrl" + "Q"	Quit the application
	Tools and helps
"F1"	Open the TranOpt Plus's manual
"F2"	Matrix tools
"F3"	Traffic assignment results comparison tools
"F4"	Flow statistic tool
"Shift" +"F1' '	Open book of classes—the helping book for developers
"Ctrl" + "I"	Information of the current version of TranOpt Plus
	Open windows and others
"Alt" + "F"	Show the filter windows
"Alt" + "P"	Show the object property windows
"Alt" + "T"	Show the tile map windows
"Shift" + "G"	Show the graph editor windows
"Shift" + "R"	Show the routing windows
"Shift" + "T"	Show the traffic assignment modeling windows

7.3 Open-source Libraries

As we mentioned in Section 7.1, there are three major open-source libraries used in TranOpt Plus for visualization, namely Qt, QCustomPlot, and MapGraphics

libraries. The later two are developed from the first one, i.e. Qt. In this section we provides more details of these libraries, such as the utilities, the state of the art, and the limitation.

7.3.1 Qt Library

Qt is a cross-platform application framework being used widely for software developing widely, see [7]. Qt can run on various hardware platforms without or with a little change in the underlying code base. Some main supported platforms are Windows, Embedded Linux, Android, iOS, WayLand. Qt has many modules, for example, Qt Core, Qt GUI, Qt Multimedia, Qt Network, Qt WebKit, and Qt Widgets. It contains a number of libraries supporting strongly for graphical user interface and graph visualization in a wide range of applications.
In TranOpt Plus, we use the Qt Core, the Qt GUI, and the Qt Widgets to design the graphical interface as well as to visualize problems. The Qt libraries were developed in C++ with a number of new features, such as a powerful mechanism for communication between objects, named signals and slots, the queryable and designable object attributes, and the object ownership in a natural way with guarded pointers. Using Qt Core is similar to using the standard libraries in C++. The Qt GUI and Qt Widgets provide a large number of classes for graphical user interface (GUI) and visualization. Especially, the **Graphics View Framework in Qt Widgets** offers a surface (QGraphicsScene) for managing and interacting with a large number of graphical items, and a view widget (QGraphicsView) for displaying the items supporting zooming and rotation. The framework has ability to handle a millions of items efficiently while it consumes a proper amount of memory space.
Qt Creator is a cross-platform C++ and a part of software development kit (SDK) for Qt GUI application development framework. Two of the most valuable features of Qt creator are a visual debugger and an integrated GUI layout and form designer. The strong debugger enables developers detect not only the grammar errors but also the logical errors. Developers can stop the running program at any breaking point and observe the values of objects.

7.3.2 MapGraphics Library

MapGraphics library (MGL) is a library based on the Qt graphic framework. MapGraphics library is originally developed for integrating various kinds of maps into Qt applications. It provides a graphic framework similar to the one of Qt with a scene to manage the graphic object and a view to display the graphic items. Basically, the scene illustrates the flat surface of the earth and the view display all objects on a rectangle area on the scene. Each graphic item is a map object on the surface of the earth with a GPS position. MapGraphics library provides a

facility to convert the position of a point displayed on the computer's screen to the GPS position. Since the scene and the view defined in the MapGraphics library is developed from the scene and view of the Qt library, they are able to manage a large number of graphic objects.
In TranOpt Plus, we use the framework of MapGraphics library to visualize the map objects. Originally, the library provides the tile item as a square area with map image. We have further developed a number of graphic items, such as node, way, zone, and area. MapGraphics library is still in development, thus it has a number of bugs, that have been fixed partly during the project of TranOpt Plus.

7.3.3 QCustomPlot Library

The **QCustomPlot** is a library for plotting and data visualization. It is developed from the Qt Widgets module and it has no further dependencies. The library focuses on generating high quality two-dimensional plots, graphs, and charts. The QCustomPlot library supports most popular kinds of plots, graphs, or charts for data visualization. It is also able to export the results to various formats, e.g. png, jpg, and pdf. The library enables users to strongly customize plots and easily embed to Qt applications.
In TranOpt Plus, the QCustomPlot library is used to visualize the results of traffic assignment modeling. It is also utilized to visualize the comparison between the predicted flows by two different models, see Fig. 7.8.

7.4 The TNGA Library

The **traffic-network-graph algorithms** (TNGA) is a library providing a number of graph algorithms and traffic assignment modeling. It is a pure C++ library and plays a fundamental role in TranOpt Plus. The TNGA library has three major parts: mathematical facilities, routing algorithms, and traffic assignment models.

7.4.1 Mathematical Facilities

The mathematical facilities in the TNGA library include the data structure of the graph and a number of popular heaps. In the TNGA library, the adjacent list data structure is used to represent graphs. This means if $G(V, E)$ is a graph with $|V|$ nodes, its links are stored in an array of $|V|$ lists. Each list contains all the links going-out from the node, and also all links going-in the node. This data structure is suitable for large graphs with thousands of nodes.
There are four kinds of priority containers in the TNGA library: binary heap, Fibonacci heap, priority list and priority queue. All of them are programmed as templates, thus they are easy to use in many contexts with different kinds of data.

7.4.2 Routing Algorithms

The TNGA library provides a number of algorithms for routing problems, such as the shortest path (SP) problem, the k shortest loop-less paths (KSLP) problem, and the dissimilar shortest loop-less paths (DSLP) problem. Most algorithms implemented in the TNGA library are mentioned in Chapter 3 and Chapter 4. The algorithms, that are implemented in the TNGA library for the routing problems, are as follows.

- **The shortest path problem:** Both of Dijkstra's algorithm and A-Star algorithm are implemented in both versions: one directional search (original version) and bidirectional search. Further more, Dijkstra's algorithm is also implemented for one source to many destinations and for many sources to one destination. Both of the algorithms are implemented using a binary heap.

- **The k shortest loop-less paths problem:** There are 4 algorithms for the KSLP problem, that are implemented in the TNGA library: Yen's algorithm, Eppstein's algorithm, Martins' algorithm, and the HELF. Eppstein's algorithm and Martins' algorithm can also find the non-loop-less shortest paths. The Fibonacci heap is used to implement Eppstein's algorithm and the HELF.

- **The dissimilar shortest loop-less paths:** The HELDF is implemented as an extension of the HELD. The penalty method is also implemented with two versions: one using Dijkstra's algorithm and one using A-Star algorithm.

7.4.3 Traffic Assignment Models

The TNGA library provides a number of traffic assignment models and the methods for solving the models. The version 1.0.2 supports three traffic assignment models: the all-or-nothing (AON) model, the incremental (INC) model, and the GUEM model. Each model supports both standard traffic systems and mixed traffic systems. When a model is run on standard traffic systems, it assigns only the traffic demand of cars, and when it is run on mixed traffic systems, it assigns the traffic demand of cars (4-wheel vehicles) and motorcycles (2-wheel vehicles). Different models have different sets of parameters and different solvers. The predicted traffic flow on a link by a model is stored in the attribute named "flow" of the link.

7.5 Software Packing

TranOpt Plus was packed by the **Nullsoft Scriptable Install System** (NSIS) - an open-source system for creating installers on the Windows platform. It belongs to Nullsoft, and is distributed on nsis.sourceforge.net. The NSIS is a script-driven system and it runs on the given scripts.

Table 7.4: List of libraries and files for TranOpt Plus packing.

Name	**Folder**	**Comment**
tnga.dll	Main folder	Traffic-network-graph algirithms
MapGraphics.dll	Main folder	Open source library
qcustomplot.dll	Main folder	Open source library
Qt5Core.dll	Main folder	Qt library
Qt5Gui.dll	Main folder	Qt library
Qt5Network.dll	Main folder	Qt library
Qt5Concurrent.dll	Main folder	Qt library
Qt5PrintSupport.dll	Main folder	Qt library
Qt5Widgets.dll	Main folder	Qt library
Qt5Xml.dll	Main folder	Qt library
icudt51.dll	Main folder	Qt supporting library
icuin51.dll	Main folder	Qt supporting library
icuuc51.dll	Main folder	Qt supporting library
libgcc_s_dw2-1.dll	Main folder	Qt supporting library
libstdc++-6.dll	Main folder	Qt supporting library
libwinpthread-1.dll	Main folder	Qt supporting library
qminimal.dll	/platforms/	Platforms supporting library
qoffscreen.dll	/platforms/	Platforms supporting library
qwindows.dll	/platforms/	Platforms supporting library
TranOpt.exe	Main folder	Application file
TranOpt.ico	Main folder	The icon file
config.xml	Main folder	Configuration file
places.txt	/data/	Places with GPS positions
**.osm*	/examples/	Examples of OSM files
/TrafficModeling/**.html*	/manual/	Files for TranOpt manual
/tranopt/**.html*	/manual/	Files for TranOpt book of classes

Packing TranOpt Plus requires a number of libraries including the Qt libraries, the MapGraphics library, the QCustomPlots library, the TNGA library, and some other supporting libraries. Each library is given in the format of **Dynamic Link Library** (DLL), that is proposed by Microsoft.
Table 7.4 shows the list of included libraries and files for packing TranOpt Plus as an installer. The DLL files are categorized into some groups. The first three are the TNGA, the MapGraphics, and the QCustom library. A number of Qt and Qt supporting libraries are also included. Those mentioned libraries are located at the main installed folder. There is a subfolder, named "platforms", containing three supporting libraries for the Windows platform. A number of OSM files are included in the subfolder named "examples". The subfolder named "data" contains data files for the software, e.g. the text file of the places with GPS coordinates. All files for the feature "Help" are stored in the subfolder "manual" consisting of two smaller folders namely "TrafficModeling" and "tranopt". Each of these folders contains html files and pictures. The main folder also contains other files, such as the Windows executable application file "TranOpt.exe", the icon file "TranOpt.ico", and the configuration file "config.xml".

7.6 Conclusion and Discussion

In this chapter we have presented the TranOpt Plus software with the three major features, namely map editing, dynamic routing, and traffic assignment modeling, as well as a number of supporting features. The three open-source libraries used in TranOpt Plus are the Qt libraries, the MapGraphics library, and the QCustomPlot library. These libraries provide tools for programming the graphical user interface as well as the visualization of TranOpt Plus software. The TNGA library plays a fundamental role in the software, since it provides the classes for routing algorithms and traffic assignment modeling. The software were tested on Windows 7 and Windows 8.
TranOpt Plus provides a strong framework for further map-related problems, such as network design, facility location, traffic optimization, etc. All the computational results of running the GUEM model in Hanoi were done and visualized on TranOpt Plus, see Chapter 6.
Further works on TranOpt Plus are in consideration as follows.

- Improving the speed of displaying large graphs, e.g. graph with millions of objects, and managing caches storing map tiles more efficiently;
- Implementing the TNGA library using parallel computing for some algorithms;
- Adding further map-related problems into TranOpt Plus;
- Creating a new version of TranOpt Plus working on Linux systems.

Chapter 8

Conclusions and Discussion

This doctoral thesis has introduced three major fields related to urban traffic planning: routing problems, traffic assignment modeling, and software development. In Chapter 3, we implemented Dijkstra's algorithm and A-Star algorithm for the shortest path (SP) problem, following both the one-directional and the bidirectional search approaches. The results indicate a fact that the algorithms, following the bidirectional search approach, dominate those following the one-directional search approach. The running times of A-Star algorithm were generally better than those of Dijkstra's algorithm, however, the former algorithm, i.e. A-Star, needs a preprocessing step where a lower bound on the length of the shortest path between two nodes is calculated.

In Chapter 4, we introduced a new heuristic based on Eppstein's algorithm, using loop-filters named HELF for the k shortest loop-less path (KSLP) problem. The computational results showed that the loop-filters can predetermine a large number of paths, from which all the candidates are loop-less, thus, the running time decreases significantly. The HELF also dominates other algorithms, i.e. Yen's algorithm, the heuristic based on Marins' algorithm (HMA), and the heuristic based on Eppstein's algorithm (HEA), in terms of running time. By adding a similarity filter into the HELF, we developed a new heuristic, named HELSF, for the dissimilar shortest loop-less paths (DSLP) problem. The HELSF was compared to the penalty method, using the bidirectional Dijkstra, on a real map of Hanoi. The results showed that the penalty method consumed less running time than those of the HELSF, however, the HELSF determined better paths in terms of the average length.

In Chapter 5, we proposed a new general user equilibrium (UE) traffic assignment model named GUEM for mixed traffic systems. In the GUEM model, the traffic demands and the potential paths between each origin-destination (O-D) pair of zones are separated with respect to the kinds of vehicles, i.e. 2-wheel or 4-wheel vehicle. More than that, the constant public bus flows on links are taken into account and further kinds of vehicles can also be added to the new model. We have proved that,

at the optimal solution of the GUEM model, there is an equilibrium for each kind of vehicle.
In Chapter 6 we investigated the mixed traffic system dominated by motorcycles (MTSDM) in Hanoi. We did not only collect the available data from former projects but have also made an online survey on the traffic behaviors in major cities in Vietnam. Furthermore, we calibrated the parameters of the BPR function, applying to the MTSDM in Hanoi. The collected data and the new BPR function were used to run the GUEM model for the traffic system in Hanoi. The high agreement between the predicted flows and the real traffic situation in Hanoi proved the accuracy of the GUEM model.
Chapter 7 presented our software, named TranOpt Plus, that has three major features: map editing, dynamic routing, and traffic assignment modeling. The graphical user interface (GUI) of the software makes it easy to use even for people who have little knowledge of mathematics and computer science. Furthermore, TranOpt Plus plays as a frame-work for further map-based applications in traffic planning, e.g. network design, traffic controlling, and facility location.

With all the achievements in this project, we would like to contribute tools for traffic optimization in urban areas where the traffic system is a mixture of 2-wheel and 4-wheel vehicles. For further works, we are going to improve the algorithms for routing problems and investigate new traffic assignment models for mixed traffic systems. The TranOpt Plus software will be enlarged by new features.

Appendix A

Map Instances

This appendix presents the details of the data, used for all experiments in this doctoral thesis, such as the data format, the map instances, and the population zones in Hanoi. All the map instances, used for testing routing algorithms, are the real city maps exported from openstreetmap.org [36].

Listing A.1: The XML format of OSM files.

```
<?xml version='1.0' encoding='UTF-8'?>
<osm version='0.6' upload='true' generator='JOSM'>
  <bounds minlat='1.9' minlon='16.8' maxlat='1.99' maxlon='17.9' />
  <node id='91' lat='10.97' lon='106.89' >
        <tag k='amenity' v='parking'/>
  </node>
  <node id='92'  lat='10.95' lon='106.87' />
   <way id='1' changeset='10546455'>
        <nd ref='91' />
        <nd ref='92' />
        <tag k='highway' v='living_street' />
  </way>
   ...
 <relation id="318"  visible="true">
        <member type="way" ref="3" role="platform"/>
 </relation>
</osm>
```

The original exported files are in the standard OSM format, that is expressed in Listing A.1. There are a number of major map objects, such as, the boundary of the map, the nodes, the ways, and the relations between objects. The boundary is the area located by the given minimum and maximum latitudes, longitudes.
Each node and each way may have a number of different attributes including a unique identification number given in the attribute "id". The latitude and longitude of a node are given in the attributes "lat" and "lon", respectively. The identification

numbers of the nodes on a way are stored in the attribute "nd". Other attributes in nodes and ways are stored in tags. Each tag has two attributes: "k" for the name of the attribute and "v" for the value of the attribute. In order to filter out unnecessary objects in exported OSM files, e.g. the nodes representing restaurants, banks, supermarkets, we used the **the JOSM software**, which is a free extensible editor for OpenStreetMap (OSM) written in Java and supporting download and upload OSM data.

The map instances were collected in 13 cities in Germany, Vietnam, USA, Thailand, the Philippines, Taiwan, and Cambodia. Table A.1 provides the list of the map instances with basic information. The first column, labeled "Name" indicate the names of maps in short. The short name of a map instance is created as {the city code}-{country code}{number of nodes}. For example, the name MH-DE6k means that it is the map of the city Mannheim in Germany (Deutschland), and it has about 6 thousand nodes. These short names are used in all the tables of computational results for routing problems. The second column, labeled "City" shows the names of the cities of the maps. The third column, labeled "Country" indicates the name of the countries corresponding to the cities. The fourth and fifth columns, named "Nr. nodes" and "Nr. links", give the number of nodes and the number of links of the maps, respectively. Some instances are not the full size but apart of the maps of the cities.

Table A.1: The list of map instances.

Name	**City**	**Country**	**Nr. nodes**	**Nr. links**
HD-DE1k	Heidelberg	Germany	1752	3517
HP-VN2k	Hai Phong	Vietnam	2731	6733
BH-VN4k	Bien Hoa	Vietnam	4749	11788
NY-USA5k	New York	The USA	5653	11322
VT-VN5k	Vung Tau	Vietnam	5030	12978
MH-DE6k	Mannheim	Germany	6439	12504
DN-VN8k	Da Nang	Vietnam	8267	22927
HN-VN9k	Hanoi	Vietnam	9753	24205
PP-CB9k	Phnompenh	Cambodia	9896	26312
MNL-PP12k	Manila	The Philippines	12932	34638
TP-TW21k	Taipet	Taiwan	21137	49774
BK-TL22k	Bangkok	Thailand	22775	48139
HCM-VN24k	HoChiMinh City	Vietnam	24965	62566

Basically, each OSM file represents a map containing nodes and ways and a way in OSM is similar to a path in the graph, i.e. a way is a sequence of links. In order to transform a map in an OSM file to a graph, we used TranOpt Plus software. The

transformed graph is saved in a TNG format, which is described in Listing A.2. The first lines in a TNG file are the information of the graph, such as name of graph and information of the graph. The real data is stored from the lines between the line "START_DATA" and the line "EOF". The first data line is the number of nodes. The next lines are links on the graph. Each line representing a link has three numbers: the first number is the source node, the second one is the destination node, and the last one is the length of the link. The data of links and node data is separated by the line"NODE_POSITION". Each line of the node data representing a node has three numbers: the identification number of node, the longitude, and the latitude of the node. The TNG file may contain or not contain the node positions. The position of nodes are used in a number of algorithms, e.g. A-Star algorithm. If a TNG file does not contain the node positions, the default position at $(0, 0)$ is set to all nodes in the graph.

Listing A.2: The format of TNG file supported by graph in TNGA library.

```
NAME:              SampleGraph
INFOR:             Created by TranOpt Plus 1.0
START_DATA
2731
0          1       1309.19
0          2       662.159
1          0       297.658
2          1       36.8423
2          0       372.757
NODE_POSITION
0          106.64  20.86
1          106.72  20.84
2          106.65  20.85
EOF
```

Table A.2 shows the list of 133 population zones in Hanoi, used in Chapter 6 for running the GUEM model. The list is customized from the project HAIDEP [42]. The first column labeled "Ord" indicates the order of zones. The next two columns labeled "Lat" and "Lon" are the latitude and longitude of the centers of the zones, respectively. The column labeled "Name" indicates the name of the zones. The first 128 zones are the quarters of the 9 urban districts in Hanoi. The ending of a quarter is the brief code of the district's name, i.e. BD stands for Ba Dinh, HK stands for Hoan Kiem, TH stands for Tay Ho, LB stands for Long Bien, CG stands for Cau Giay, DD stands for Dong Da, HBT stands for Hai Ba Trung, HM stands for Hoang Mai, and TX stands for Thanh Xuan. The last 5 zones are the districts out of the

center of Hanoi, i.e. Dong Anh, Gia Lam, Tu Liem, Thanh Tri, and Ha Dong.

Table A.2: Population zones in Hanoi, customized from data of the HAIDEP project [42].

Ord	Lat	Lon	Name	Ord	Lat	Lon	Name
1	21.04719	105.84910	Phuc Xa, BD	68	21.02244	105.83726	Van Chuong, DD
2	21.04504	105.84139	Truc Bach, BD	69	21.02311	105.83114	Hang Bot, DD
3	21.04223	105.80932	Vinh Phuc, BD	70	21.01531	105.81257	Lang Ha, DD
4	21.03587	105.81013	Cong Vi, BD	71	21.01956	105.83862	Kham Thien, DD
5	21.03730	105.81840	Lieu Giai, BD	72	21.01723	105.83428	Tho Quan, DD
6	21.04157	105.84710	Nguyen Trung Truc, BD	73	21.01427	105.83110	Nam Dong, DD
7	21.03866	105.84129	Quan Thanh, BD	74	21.01514	105.83862	Trung Phung, DD
8	21.03829	105.82810	Ngoc Ha, BD	75	21.01194	105.82537	Quang Trung, DD
9	21.03347	105.83975	Dien Bien, BD	76	21.01137	105.82252	Trung Liet, DD
10	21.03427	105.82756	Doi Can, BD	77	21.01237	105.83713	Phuong Lien, DD
11	21.03016	105.80994	Ngoc Khanh, BD	78	21.00838	105.81798	Thinh Quang, DD
12	21.03086	105.82550	Kim Ma, BD	79	21.00435	105.83117	Trung Tu, DD
13	21.02688	105.81955	Giang Vo, BD	80	21.00669	105.83609	Kim Lien, DD
14	21.02068	105.81582	Thanh Cong, BD	81	21.00144	105.83926	Phuong Mai, DD
15	21.03704	105.85695	Phuc Tan, HK	82	21.00524	105.82214	Nga Tu So, DD
16	21.03882	105.85061	Dong Xuan, HK	83	21.00303	105.82846	Khuong Thuong, DD
17	21.03715	105.84671	Hang Ma, HK	84	21.01863	105.84583	Nguyen Du, HBT
18	21.03559	105.85168	Hang Buom, HK	85	21.01332	105.86585	Bach Dang, HBT
19	21.03464	105.84993	Hang Dao, HK	86	21.01588	105.85738	Pham Dinh Ho, HBT
20	21.03471	105.84776	Hang Bo, HK	87	21.01501	105.85073	Bui Thi Xuan, HBT
21	21.03301	105.84558	Cua Dong, HK	88	21.01670	105.85344	Ngo Thi Nham, HBT
22	21.03045	105.85464	L Thai To, HK	89	21.01247	105.84504	Le Dai Hanh, HBT
23	21.03275	105.85275	Hang Bac, HK	90	21.01277	105.85645	Dong Nhan, HBT
24	21.03199	105.84854	Hang Gai, HK	91	21.01076	105.85331	Pho Hue, HBT
25	21.02763	105.86192	Chuong Duong Do, HK	92	21.01117	105.86029	Dong Mac, HBT
26	21.02831	105.85046	Hang Trong, HK	93	21.00743	105.87108	Thanh Luong, HBT
27	21.02516	105.84257	Cua Nam, HK	94	21.00535	105.85697	Thanh Nhan, HBT
28	21.02857	105.84570	Hang Bong, HK	95	21.00609	105.85002	Cau Den, HBT
29	21.02517	105.85482	Trang Tien, HK	96	21.00429	105.84599	Bach Khoa, HBT
30	21.02298	105.84737	Tran Hung Dao, HK	97	20.99887	105.84438	Dong Tam, HBT
31	21.02036	105.85747	Phan Chu Trinh, HK	98	20.99793	105.86869	Vinh Tuy, HBT
32	21.02081	105.85179	Hang Bai, HK	99	21.00096	105.85182	Bach Mai, HBT
33	21.08508	105.80778	Phu Thuong, TH	100	21.00010	105.86067	Qunh Mai, HBT
34	21.07787	105.82572	Nhat Tan, TH	101	21.00036	105.85628	Qunh Loi, HBT
35	21.06825	105.83700	Tu Lien, TH	102	20.99594	105.85776	Minh Khai, HBT
36	21.05719	105.82616	Quang An, TH	103	20.99477	105.84818	Truong Dinh, HBT
37	21.06158	105.80383	Xuan La, TH	104	20.99499	105.89051	Thanh Tri, HM
38	21.05400	105.83865	Yen Phu, TH	105	20.98920	105.87462	Vinh Hung, HM
39	21.05264	105.81368	Buoi, TH	106	20.98315	105.83214	Dinh Cong, HM
40	21.04428	105.82383	Thuy Khue, TH	107	20.99090	105.86492	Mai Dong, HM
41	21.07525	105.89641	Thuong Thanh, LB	108	20.98834	105.85110	Tuong Mai, HM
42	21.06117	105.86853	Ngoc Thuy, LB	109	20.97384	105.82155	Dai Kim, HM
43	21.06790	105.91574	Giang Bien, LB	110	20.98359	105.84818	Tan Mai, HM
44	21.06332	105.89452	Duc Giang, LB	111	20.98520	105.86030	Hoang Vn Thu, HM
45	21.05752	105.90254	Viet Hung, LB	112	20.98429	105.84294	Giap Bat, HM
46	21.04928	105.88604	Gia Thuy, LB	113	20.97674	105.89654	Linh Nam, HM
47	21.04506	105.86919	Ngoc Lam, LB	114	20.97556	105.85496	Thinh Liet, HM
48	21.04399	105.92491	Phuc Loi, LB	115	20.97228	105.88445	Tran Phu, HM
49	21.03722	105.87306	Bo De, LB	116	20.96485	105.83706	Hoang Liet, HM
50	21.03094	105.91683	Sai Dong, LB	117	20.96245	105.87255	Yen So, HM
51	21.01900	105.88512	Long Bien, LB	118	21.00306	105.80367	Nhan Chnh, TX
52	21.02171	105.91477	Thach Ban, LB	119	21.00133	105.81494	Thuong Dinh, TX
53	21.03947	105.89693	Phuc Dong, LB	120	20.99718	105.82120	Khuong Trung, TX
54	21.00715	105.90002	Cu Khoi, LB	121	20.99634	105.83052	Khuong Mai, TX
55	21.04435	105.80324	Nghia Do, CG	122	20.99591	105.80400	Thanh Xuan Trung, TX
56	21.04548	105.79125	Nghia Tan, CG	123	20.99148	105.83890	Phuong Liet, TX
57	21.04023	105.77426	Mai Dich, CG	124	20.98656	105.81011	Ha Dinh, TX
58	21.03489	105.79253	Dich Vong, CG	125	20.98878	105.81858	Khuong Dinh, TX
59	21.03439	105.78506	Dich Vong Hau, CG	126	20.99336	105.79835	Thanh Xuan Bc, TX
60	21.03667	105.80142	Quan Hoa, CG	127	20.98553	105.79912	Thanh Xuan Nam, TX
61	21.02132	105.79046	Yen Hoa, CG	128	20.98239	105.81245	Kim Giang, TX
62	21.01033	105.79815	Trung Hoa, CG	129	21.14362	105.79761	Dong Anh
63	21.02861	105.82935	Cat Linh, DD	130	21.06122	105.96606	Gia Lam
64	21.02652	105.84011	Vn Mieu, DD	131	21.03709	105.70822	Tu Liem
65	21.02743	105.83278	Quoc Tu Giam, DD	132	20.90577	105.85712	Thanh Tri
66	21.02187	105.80411	Lang Thuong, DD	133	20.95418	105.74640	Ha Dong
67	21.01938	105.82552	O Cho Dua, DD				

Appendix B

Computational Results

This appendix shows the details of the computational results of the algorithms, mentioned in Chapter 4, i.e. the heuristic based on Martins' algorithm (HMA), Yen's algorithm, the heuristic based on Eppstein's algorithm (HEA), and the heuristic based on Eppstein's algorithm using loop-filters (HELF), for the KSLP problem. Table B.1, Table B.2, Table B.3, Table B.4, Table B.5, Table B.6, and Table B.7 show the computational results of the algorithms corresponding to 7 different values of k, i.e. $k \in \{5, 10, 20, 30, 40, 50, 60\}$.

The meanings of the columns of all the tables are the same as follows. The first column, labeled "Maps", indicates the map instances using for the experiments. The remaining columns are grouped in 4 blocks corresponding to the computational results of the 4 mentioned algorithms, i.e. the block "Martins (HMA)" for the HMA, the block "Yen" for Yen's algorithm, the bock "Eppstein (HEA)" for the HEA, and the block "HELF" for the HELF. In turn, each block has 3 columns. The first column in each block, labeled "Vs", indicates the average numbers of visited paths by the algorithms, the second column, named "Fs", show the average numbers of loop-less paths found by the algorithms within the given maximum number of iterations, and the last column, labeled "Time", gives the running times of the algorithms. The last row of each table shows the average values of the computational results in all the map instances.

For the heuristics, i.e. the HMA, the HEA, and the HELF, the maximum number of iteration is set to $1000 \times k$, e.g. $Imax = 5000$ for the case $k = 5$. All the experiments were implemented in a personal desktop computer with an Intel (R) dual Core at 2.20 GHz processor and 2 GB of RAM.

Table B.1: Computational results for the KSLP problem with $k = 5$.

Maps	**Martins (HMA)**			**Yen**			**Eppstein (HEA)**			**HELF**		
	Vs	Fs	Time	Vs	Fs	Time	Vs	Fs	Time	Vs	Fs	Time
HD-De1k	13.20	5	0.0195	5	5	0.0125	87.47	5	0.0073	28.60	5	0.0062
HD-VN2k	12.30	5	0.0345	5	5	0.0201	42.50	5	0.0121	13.10	5	0.0086
BH-VN4k	15.93	5	0.0643	5	5	0.0466	408.07	4.93	0.0203	53.07	5	0.0178
NY-USA5k	6.57	5	0.0350	5	5	0.0696	48.86	5	0.0197	7.43	5	0.0179
VT-VN5k	18.13	5	0.0993	5	5	0.0697	495.93	4.73	0.0253	70.73	5	0.0220
MH-DE6k	11.00	5	0.0785	5	5	0.0829	491.13	4.93	0.0308	24.00	5	0.0208
DN-VN8k	9.00	5	0.0713	5	5	0.0855	363.67	4.73	0.0417	13.00	5	0.0421
HN-VN9k	8.33	5	0.0690	5	5	0.0811	473.47	4.93	0.0441	17.07	5	0.0399
PP-CB9k	5.20	5	0.0709	5	5	0.2334	5.13	5	0.0512	5.13	5	0.0429
MNL-PP12k	8.60	5	0.1444	5	5	0.3190	180.33	5	0.0779	11.40	5	0.0568
TP-TW21k	5.83	5	0.1393	5	5	0.3591	6.33	5	0.1015	6.33	5	0.0941
BK-TL22k	10.47	5	0.2673	5	5	0.5815	10.87	5	0.1057	7.80	5	0.0916
HCM-VN24k	8.80	5	0.2779	5	5	1.0167	10.20	5	0.1237	7.87	5	0.1154
Average	**10.26**	**5**	**0.1055**	**5**	**5**	**0.2291**	**201.84**	**4.94**	**0.0509**	**20.43**	**5**	**0.0443**

Table B.2: Computational results for the KSLP problem with $k = 10$.

Maps	**Martins (HMA)**			**Yen**			**Eppstein (HEA)**			**HELF**		
	Vs	Fs	Time	Vs	Fs	Time	Vs	Fs	Time	Vs	Fs	Time
HD-De1k	29.27	10	0.0749	10	10	0.0311	437.33	10.00	0.0090	75.33	10	0.0106
HD-VN2k	29.20	10	0.0672	10	10	0.0286	1486.80	9.80	0.0206	91.50	10	0.0131
BH-VN4k	34.00	10	0.1775	10	10	0.1022	1648.73	9.40	0.0297	152.87	10	0.0197
NY-USA5k	15.43	10	0.0664	10	10	0.1021	1437.29	9.43	0.0423	20.71	10	0.0181
VT-VN5k	36.67	10	0.1889	10	10	0.1334	2895.27	9.27	0.0438	169.13	10	0.0237
MH-DE6k	29.93	10	0.1885	10	10	0.1237	1411.80	9.40	0.0445	63.87	10	0.0217
DN-VN8k	20.60	10	0.1415	10	10	0.1213	748.07	9.40	0.0610	37.80	10	0.0334
HN-VN9k	19.60	10	0.1462	10	10	0.1394	1348.93	9.40	0.0503	37.87	10	0.0389
PP-CB9k	13.13	10	0.1201	10	10	0.3411	12.33	10.00	0.0645	11.87	10	0.0393
MNL-PP12k	18.33	10	0.2223	10	10	0.4783	858.73	9.73	0.1000	38.00	10	0.0785
TP-TW21k	20.25	10	0.4489	10	10	0.7248	20.92	10.00	0.1023	18.75	10	0.1057
BK-TL22k	25.47	10	0.6782	10	10	0.9907	55.33	10.00	0.1071	25.87	10	0.0859
HCM-VN24k	17.60	10	0.5347	10	10	1.7383	44.53	10.00	0.2015	20.33	10	0.1351
Average	**23.81**	**10**	**0.2350**	**10**	**10**	**0.3888**	**954.31**	**9.68**	**0.0674**	**58.76**	**10**	**0.0480**

Table B.3: Computational results for the KSLP problem with $k = 20$.

Maps	**Martins (HMA)**			**Yen**			**Eppstein (HEA)**			**HELF**		
	Vs	Fs	Time	Vs	Fs	Time	Vs	Fs	Time	Vs	Fs	Time
HD-De1k	72.93	20	0.2500	20	20	0.0400	4491.13	18.93	0.0644	224.73	20	0.0105
HD-VN2k	76.60	20	0.2675	20	20	0.0669	5449.60	18.40	0.0518	162.20	20	0.0115
BH-VN4k	90.60	20	0.5651	20	20	0.1591	6756.73	17.60	0.0993	451.33	20	0.0245
NY-USA5k	29.00	20	0.1776	20	20	0.1891	2873.14	16.57	0.0449	46.29	20	0.0190
VT-VN5k	64.20	20	0.5099	20	20	0.2144	9608.93	16.93	0.1358	380.07	20	0.0335
MH-DE6k	70.40	20	0.5977	20	20	0.2697	3459.80	18.27	0.0566	125.20	20	0.0299
DN-VN8k	47.27	20	0.3687	20	20	0.2669	1774.73	18.73	0.0684	97.67	20	0.0329
HN-VN9k	46.73	20	0.3930	20	20	0.2537	2715.27	18.20	0.0579	73.07	20	0.0544
PP-CB9k	30.53	20	0.3193	20	20	0.4866	35.07	20.00	0.0453	25.87	20	0.0385
MNL-PP12k	35.67	20	0.4633	20	20	0.7549	2880.60	18.80	0.0879	107.27	20	0.0751
TP-TW21k	44.92	20	0.8954	20	20	1.0399	156.08	20.00	0.1044	71.83	20	0.0843
BK-TL22k	83.53	20	2.1887	20	20	1.5253	1520.73	19.87	0.1238	105.47	20	0.1123
HCM-VN24k	45.40	20	1.1649	20	20	2.4416	128.00	20.00	0.1597	41.40	20	0.1027
Average	**56.75**	**20**	**0.6278**	**20**	**20**	**0.5929**	**3219.22**	**18.64**	**0.0846**	**147.11**	**20**	**0.0484**

Table B.4: Computational results for the KSLP problem with $k = 30$.

Maps	**Martins (HMA)**			**Yen**			**Eppstein (HEA)**			**HELF**		
	Vs	Fs	Time	Vs	Fs	Time	Vs	Fs	Time	Vs	Fs	Time
HD-De1k	109.60	30	0.4417	30	30	0.0603	13571.07	27.27	0.1086	380.20	30	0.0243
HD-VN2k	128.20	30	0.6276	30	30	0.0620	10302.60	26.30	0.0657	227.80	30	0.0137
BH-VN4k	138.60	30	1.2033	30	30	0.2101	14158.53	24.87	0.1317	737.47	30	0.0323
NY-USA5k	49.29	30	0.3503	30	30	0.3421	4369.43	26.57	0.0674	90.43	30	0.0226
VT-VN5k	94.93	30	0.6573	30	30	0.2186	15960.73	24.53	0.1931	542.07	30	0.0384
MH-DE6k	99.53	30	0.8626	30	30	0.3551	8952.40	26.07	0.0910	219.53	30	0.0303
DN-VN8k	76.00	30	0.6385	30	30	0.3573	2979.07	28.07	0.1086	155.93	30	0.0369
HN-VN9k	91.47	30	0.9541	30	30	0.3685	6071.47	26.53	0.0901	127.07	30	0.0601
PP-CB9k	53.20	30	0.6087	30	30	0.6225	97.33	30.00	0.0634	41.87	30	0.0389
MNL-PP12k	57.47	30	0.8155	30	30	0.9727	4391.40	27.73	0.0929	156.73	30	0.0566
TP-TW21k	70.75	30	1.5146	30	30	1.3157	510.08	30.00	0.1062	117.75	30	0.1053
BK-TL22k	124.13	30	3.2221	30	30	2.1191	2723.87	29.20	0.1163	213.00	30	0.0965
HCM-VN24k	77.20	30	2.0504	30	30	3.3842	373.87	30.00	0.1168	72.13	30	0.1186
Average	**90.03**	**30**	**1.0728**	**30**	**30**	**0.7991**	**6497.07**	**27.47**	**0.1040**	**237.08**	**30**	**0.0519**

Table B.5: Computational results for the KSLP problem with $k = 40$.

Maps	**Martins (HMA)**			**Yen**			**Eppstein (HEA)**			**HELF**		
	Vs	Fs	Time	Vs	Fs	Time	Vs	Fs	Time	Vs	Fs	Time
HD-De1k	150.87	40	0.7285	40	40	0.1332	25676.13	32.87	0.2383	558.47	40	0.0223
HD-VN2k	171.30	40	0.9855	40	40	0.0758	16398.50	33.00	0.1297	299.30	40	0.0154
BH-VN4k	209.87	40	2.3832	40	40	0.3193	19139.07	31.33	0.162	932.13	40	0.0357
NY-USA5k	67.71	40	0.3803	40	40	0.4463	6789.14	35.29	0.0779	144.86	40	0.0246
VT-VN5k	127.00	40	1.0430	40	40	0.2797	20926.80	30.13	0.2176	691.60	40	0.0445
MH-DE6k	133.20	40	1.4725	40	40	0.4199	14033.73	33.60	0.1275	344.53	40	0.0336
DN-VN8k	108.73	40	0.9091	40	40	0.3349	5059.80	37.40	0.1023	231.47	40	0.0395
HN-VN9k	129.60	40	1.2869	40	40	0.4249	8133.87	34.67	0.0910	181.13	40	0.0445
PP-CB9k	71.93	40	0.8254	40	40	0.8101	209.73	40.00	0.0626	59.07	40	0.0443
MNL-PP12k	83.13	40	1.1271	40	40	1.1895	5960.13	36.47	0.1016	198.93	40	0.0629
TP-TW21k	99.42	40	2.2210	40	40	1.3517	859.50	40.00	0.1348	152.33	40	0.0951
BK-TL22k	172.40	40	4.9333	40	40	2.9394	4270.60	38.53	0.1243	306.93	40	0.1040
HCM-VN24k	112.60	40	3.1371	40	40	4.2633	635.53	40.00	0.1192	97.40	40	0.1164
Average	**125.98**	**40**	**1.6487**	**40**	**40**	**0.9991**	**9853.27**	**35.64**	**0.1299**	**322.94**	**40**	**0.0525**

Table B.6: Computational results for the KSLP problem with $k = 50$.

Maps	**Martins (HMA)**			**Yen**			**Eppstein (HEA)**			**HELF**		
	Vs	Fs	Time	Vs	Fs	Time	Vs	Fs	Time	Vs	Fs	Time
HD-De1k	203.13	50	1.3153	50	50	0.0913	32907.07	37.93	0.2601	744.87	50	0.0293
HD-VN2k	218.30	50	1.5765	50	50	0.1510	20693.70	39.80	0.1445	366.50	50	0.0170
BH-VN4k	270.33	50	3.5839	50	50	0.3136	25207.07	38.27	0.2471	1095.80	50	0.0403
NY-USA5k	89.00	50	0.6896	50	50	0.5944	8304.00	44.00	0.0700	166.86	50	0.0251
VT-VN5k	161.33	50	1.6091	50	50	0.4830	26292.80	36.00	0.2018	811.93	50	0.0516
MH-DE6k	168.27	50	1.8999	50	50	0.4537	24150.47	40.00	0.2257	833.33	50	0.0562
DN-VN8k	142.13	50	1.3208	50	50	0.4957	9022.60	46.60	0.1327	310.00	50	0.0415
HN-VN9k	168.13	50	2.1215	50	50	0.5731	10279.27	42.67	0.1237	254.33	50	0.0468
PP-CB9k	90.53	50	1.1689	50	50	0.9740	346.07	50.00	0.0467	76.53	50	0.0445
MNL-PP12k	112.53	50	1.6331	50	50	1.5154	7542.73	45.33	0.1183	238.00	50	0.0642
TP-TW21k	123.58	50	2.5532	50	50	1.6074	1218.92	50.00	0.1037	188.25	50	0.0984
BK-TL22k	224.00	50	7.4288	50	50	3.6551	8166.00	47.47	0.1731	392.67	50	0.1086
HCM-VN24k	150.73	50	4.7572	50	50	5.5929	904.67	50.00	0.1215	125.47	50	0.1179
Average	**163.23**	**50**	**2.4352**	**50**	**50**	**1.2693**	**13464.26**	**43.70**	**0.1515**	**431.12**	**50**	**0.0570**

Table B.7: Computational results for the KSLP problem with $k = 60$.

Maps	**Martins (HMA)**			**Yen**			**Eppstein (HEA)**			**HELF**		
	Vs	Fs	Time	Vs	Fs	Time	Vs	Fs	Time	Vs	Fs	Time
HD-De1k	256.40	60	2.0170	60	60	0.1930	42862.87	42.60	0.3160	1009.60	60	0.0507
HD-VN2k	265.90	60	2.4560	60	60	0.1765	25029.60	46.70	0.2051	448.70	60	0.0186
BH-VN4k	329.13	60	4.6749	60	60	0.4387	34325.47	44.73	0.2683	1297.13	60	0.0473
NY-USA5k	117.14	60	1.1367	60	60	0.9016	9752.86	52.57	0.1231	186.14	60	0.0243
VT-VN5k	194.07	60	2.7632	60	60	0.6829	34192.53	41.40	0.4509	928.20	60	0.0591
MH-DE6k	204.27	60	3.1373	60	60	0.5997	29528.47	45.73	0.5262	1019.27	60	0.0987
DN-VN8k	177.73	60	2.0107	60	60	0.7845	11642.93	55.40	0.2780	399.33	60	0.0413
HN-VN9k	206.13	60	2.2293	60	60	0.8335	12426.60	50.73	0.1413	322.73	60	0.0460
PP-CB9k	110.73	60	1.6583	60	60	1.2834	444.53	60.00	0.0769	91.80	60	0.0421
MNL-PP12k	139.73	60	2.6578	60	60	1.8380	9252.73	54.07	0.1984	280.07	60	0.0651
TP-TW21k	148.75	60	4.2668	60	60	4.3307	1682.25	60.00	0.1679	227.92	60	0.1003
BK-TL22k	273.53	60	14.3594	60	60	4.8158	10334.87	56.53	0.3545	484.67	60	0.1125
HCM-VN24k	190.20	60	9.5310	60	60	7.8309	1139.47	60.00	0.2065	147.40	60	0.1187
Average	**201.06**	**60**	**4.0691**	**60**	**60**	**1.9007**	**17124.24**	**51.57**	**0.2549**	**526.38**	**60**	**0.0634**

List of Algorithms

List of Figures

List of Tables

References

[1] A. Aggarwal, B. Schieber, and T. Tokuyama. Finding a minimum-weightk-link path in graphs with the concave monge property and applications. *Discrete & Computational Geometry*, 12(1):263–280, 1994. 46

[2] H. Aljazzar and S. Leue. K*: A heuristic search algorithm for finding the k shortest paths. *Artificial Intelligence*, 175(18):2129–2154, 2011. 46

[3] H. Bast. Efficient route planning. Lecture series, University of Freiburg, 2012. Accessed: 20.07.2015. 27

[4] H. Bast, D. Delling, A. Goldberg, M. Müller-Hannemann, T. Pajor, P. Sanders, D. Wagner, and R.F. Werneck. Route planning in transportation networks. *arXiv preprint arXiv:1504.05140*, 2015. 40, 76, 157

[5] M.J. Beckmann, C.B. McGuire, and C.B. Winsten. *Studies in the Economics of Transportation*. Rand Corporation, 1955. 83, 84

[6] R. Bellman. On a routing problem. Technical report, DTIC Document, 1956. 27

[7] J. Blanchette and M. Summerfield. *C++ GUI programming with Qt 4*. Prentice Hall Professional, 2006. 133

[8] D. Branston. Link capacity function: A review. *Transportation Research*, 10:223–236, 1976. 20

[9] Bureau of Public Roads, US. *Traffic assignment manual*. U.S. Dept. of Commerce, Bureau of Public Roads, Office of Planning, Urban Planning Division, 1964. 20, 21, 113

[10] CadPro JSC. Cadpro traffic management system (cadpro tms). www.cadpro.vn, 2015. Accessed: 29.06.2015. 102

[11] N.Y. Cao and K. Sano. Estimating capacity and motorcycle equivalent units on urban roads in hanoi, vietnam. *Journal of Transportation Engineering*, 138(6):776–785, 2012. 83, 91, 100

[12] Central Intelligence Agency, US. The world factbook: Vietnam. www.cia.gov, 2015. Accessed: 14.11.2015. 97

[13] S. Chandra and U. Kumar. Effect of lane width on capacity under mixed traffic conditions in india. *Journal of Transportation Engineering*, 129(2):155–160, 2003. 83, 91

[14] C.M. Chu, K. Sano, and S. Matsumoto. The speed, flow and headway analyses of motorcycle traffic. *Journal of the Eastern Asia Society for Transportation Studies*, 6:1496–1508, 2005. 83, 91, 100

[15] T.H. Cormen. *Introduction to algorithms*. MIT press, third edition, 2009. 7

[16] P. Dell'Olmo, M. Gentili, and A. Scozzari. On finding dissimilar pareto-optimal paths. *European Journal of Operational Research*, 162(1):70–82, 2005. 70

[17] E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959. 27, 28

[18] N.H. Do, Q.L.N. Le, K.C. Nam, and M.S. Lee. Determining the interrelations among traffic factors through traffic simulation analysis. *International Journal of Social Sciences*, 5(2), 2010. 100

[19] M. Ehrgott and X. Gandibleux. A survey and annotated bibliography of multiobjective combinatorial optimization. *OR Spektrum*, 22:425–460, 2000. 76

[20] D. Eppstein. Finding the k shortest paths. *SIAM Journal on Computing*, 28(2):652–673, 1998. 44, 46, 56

[21] E. Erkut. The discrete p-dispersion problem. *European Journal of Operational Research*, 46:48–60, 1990. 71

[22] E. Erkut, V. Akgn, and R. Batta. On finding dissimilar paths. *European Journal of Operational Research*, 121:232–246, 2000. 70, 71

[23] M. Fellendorf and P. Vortisch. Microscopic traffic flow simulator vissim. In *Fundamentals of Traffic Simulation*, pages 63–93. Springer, 2010. 108

[24] M. Florian and S. Nguyen. An application and validation of equilibrium trip assignment methods. *Transportation science*, 10:374–390, 1976. 109

[25] R.W. Floyd. Algorithm 97: shortest path. *Communications of the ACM*, 5(6):345, 1962. 27

[26] L.R. Ford and D.R. Fulkerson. *Flows in networks*, volume 1962. Princeton Princeton University Press, 1962. 27

[27] M. Frank and P. Wolfe. An algorithm for quadratic programming. *Naval research logistics quarterly*, 3(1-2):95–110, 1956. 4, 17, 83

[28] M.L. Fredman and R.E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*, 34(3):596–615, 1987. 27, 30

[29] H.N. Gabow. Scaling algorithms for network problems. In *Foundations of Computer Science, 1983., 24th Annual Symposium on*, pages 248–258. IEEE, 1983. 27

[30] R. Geisberger, M. Kobitzsch, and P. Sanders. Route planning with flexible objective functions. In *ALENEX*, volume 10, pages 124–137. SIAM, 2010. 76

[31] General Statistics Office of Viet Nam. Area, population and population density in 2011 by province. www.gso.gov.vn, 2011. Accessed: 25.06.2015. 98

[32] F. Glover, R. Glover, and D. Klingman. Computational study of an improved shortest path algorithm. *Networks*, 14(1):25–36, 1984. 27

[33] A.V. Goldberg. Point-to-point shortest path algorithms with preprocessing. In *SOFSEM 2007: Theory and Practice of Computer Science*, pages 88–102. Springer, 2007. 41, 157

[34] A.V. Goldberg and C. Harrelson. Computing the shortest path: A search meets graph theory. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 156–165. SIAM, 2005. 34

[35] A.V. Goldberg and T. Radzik. A heuristic improvement of the bellman-ford algorithm. *Applied Mathematics Letters*, 6(3):3–6, 1993. 27

[36] M. Haklay and P. Weber. Openstreetmap: User-generated street maps. *Pervasive Computing, IEEE*, 7(4):12–18, 2008. 101, 141

[37] P.E. Hart, N.J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions of systems science and cybernetics*, SSC-4(2), 1968. 27, 32

[38] N.Q. Hien and F. Montgomery. Saturation flow and vehicle equivalence factors in traffic dominated by motorcycles. In *Transportation Research Board 86th Annual Meeting*, number 07-2869, 2007. 100, 108

[39] K.V. Hung. *Traffic Management in Motorcycle Dependent Cities*. Ph.D.Thesis, Darmstadt University of Technology, Darmstadt, Germany, 2006. 100

[40] N.H. Hung and P.V. Ha. Dac thu cua giao thong do thi phu thuoc xe may va anh huong cua no toi nan un tac & tai nan giao thong, giai phap doi voi giao thong xe may trong do thi hanoi. Technical report, Hanoi university of transportation and communication, Hanoi, Vietnam, 2011. In Vienamese. 98

[41] N.A. Irwin, N. Dodd, and H.G. Von Cube. Capacity restraint in assignment programs. *Highway Research Board Bulletin*, (297):109–127, 1961. 20

[42] Japan International Cooperation Agency and Hanoi People's Committee. *The Comprehensive Urban Development Programme in Hanoi Capital City of the Socialist Republic of Vietnam (HAIDEP)*. The HAIDEP Project, Hanoi, 2007. 1, 100, 101, 143, 144, 157

[43] V.M. Jiménez and A. Marzal. Computing the k shortest paths: A new algorithm and an experimental comparison. In *Algorithm engineering*, pages 15–29. Springer, 1999. 46

[44] V.M. Jiménez and A. Marzal. A lazy version of eppsteins k shortest paths algorithm. In *Experimental and Efficient Algorithms*, pages 179–191. Springer, 2003. 46, 61

[45] B. Korte and J. Vygen. *Combinatorial Optimization: Theory and Algorithms*. Springer, fifth edition, 2011. 7

[46] L.J. LeBlanc, E.K. Morlok, and W.P. Pierskalla. An efficient approach to solving the road network equilibrium traffic assignment problem. *Transportation Research*, 9:309–318, 1975. 4, 83, 86

[47] R. Marti, J.L.G. Velarde, and A. Duarte. Heuristics for the bi-objective path dissimilarity problem. *Computer & Operations Research*, 36(11):2905–2912, 2009. 70

[48] E.Q.V. Martins. An algorithm for ranking paths that may contain cycles. *European Journal of Operational Research*, 18:123–130, 1984. 46, 49

[49] E.Q.V. Martins. On a multicriteria shortest path problem. *European Journal of Operational Research*, 16:236–245, 1984. 76

[50] A.L. Medaglia, D. Duque, and L. Lozano. An exact method for the biobjective shortest path problem for large-scale road networks. *European Journal of Operational Research*, 242(3):788–797, 2015. 76

[51] A. Mehar, S. Chandra, and S. Velmurugan. Highway capacity through vissim calibrated for mixed traffic conditions. *KSCE Journal of Civil Engineering*, 18(2):639–645, 2014. 108

[52] C.C. Minh. *Analysis of motorcycle behaviour at Midblocks and Signalized intersections*. Ph.D.Thesis, Nagaoka University of Technology, Japan, 2007. 100

[53] E.F. Moore. *The shortest path through a maze*. Bell Telephone System., 1959. 27

[54] W.W. Mosher. A capacity-restraint algorithm for assigning flow to a transport network. *Highway Research Record*, (6):41–70, 1963. 20

[55] C. Musil and C. Molt. Building a public transportation system in hanoi: Between emergency and constraints. *Hanoi past and future*, 2010. 100, 103

[56] J. Nocedal and S.J. Wright. *Numerical Optimization*. Springer, second edition, 2006. 7

[57] K.R. Overgaard. Urban transportation planning: traffic estimation. *Traffic Quarterly*, 21(2), 1967. 20

[58] M.M.B. Pascoal, E.Q.V. Martins, and J.L.E. Santos. Labeling algorithms for ranking shortest paths. citeseerx.ist.psu.edu, 1999. 46, 48

[59] Remon-hanoi. Remon - real time monitoring of urban transport - solutions for traffic management and urban development in hanoi. www.remon-hanoi.net, 2013. Accessed: 18.09.2014. 115, 117

[60] Y. Sheffi. *Urban Transportation Networks: Equilibrium Analysis with Mathematical Programming Methods*. Prentice-Hall, Inc., Massachusetts Institute of Technology, 1985. 7, 83

[61] R. Smock. An iterative assignment approach to capacity restraint on arterial networks. *Highway Research Board Bulletin*, (347):60–66, 1962. 20

[62] H. Spiess. Conical volume-delay functions. *Transportation Science*, 24(2):153–158, 1990. 20, 109

[63] P.A. Steenbrink. Transport network optimization in the dutch integral transportation study. *Transportation Research*, 8:11–27, 1974. 20, 109

[64] The World Bank. *Transport Strategy-Transition, Reform, and Sustainable Management: Vietnam's infrastructure challenge*. World Bank, 2006. 1, 98, 100

[65] The World Bank. *Vietnam Urbanization Review: Technical Assistance Report*. World Bank, 2011. 1

[66] Vietnamnet Online Newspaper. Hanoi faces parking crunch. vietnamnet.vn, 2012. Accessed: 25.06.2015. 98

[67] J.G. Wardrop. Some theoretical aspects of road traffic research. In *Inst Civil Engineers Proc London, UK*, pages 325–378, Great Britain, 1952. Institution of Civil Engineers. 3, 21, 83

[68] S. Warshall. A theorem on boolean matrices. *Journal of the ACM (JACM)*, 9(1):11–12, 1962. 27

[69] J.Y. Yen. Finding the k shortest loopless paths in a network. *Management Science*, 17:712–716, 1971. 44, 46, 53

[70] F.B. Zhan and C.E. Noon. Shortest path algorithms: An evaluation using real road networks. *Transportation Science*, 32(1):65, 1998. 27

Abbreviations

AON	All-or-nothing
BPR	Bureau of Public Roads
BOSP	Bi-Objective Shortest Paths
DSLP	Dissimilar Shortest Loop-less Paths
EA	Eppstein's Algorithm
GUE	General User Equilibrium
GUEM	General User Equilibrium for Mixed Traffic Systems
HEA	Heuristic Based on EA
HMA	Heuristic Based on MA
HELF	Heuristic Based on EA using Loop-Filters
HELSF	Heursitic Based on EA using Loop-Similarity-Filters
HRA	Heuristic Based on Ranking Algorithms
IP	Integer Program
KKT	Kuhn-Kuhn-Tucker
KSP	K Shortest Paths
KSLP	K Shortest Loop-less Paths
KSNLP	K Shortest Non-loop-less Paths
LCCM	Linear Constrained Convex Minimization
MCU	Motorcycle Unit
MOSP	Multi-Objective Shortest Paths
MTS	Mixed Traffic System
MTSDM	Mixed Traffic System Dominated by Motorcycles
ODM	One-dimentional Minimization
OSM	Open Street Map
PCU	Passenger Car Unit
TA	Traffic Assignment
TAM	Traffic Assignment Modeling
UE	User Equilibrium
UTP	Urban Traffic Planning

Mathematical Symbols

$\forall$	For all elements
$\mathbb{N}$	Set of integer numbers
$\mathbb{R}$	Space of real numbers
$\mathbb{R}^n$	n dimensional space
$\emptyset$	Empty set
$\|S\|$	Number of elements of the set S
$O(f)$	Asymptotic upper bound of the function f
∇z	Vector of the partial derivatives of the scalar-valued function z
$\nabla^2 z$	Square matrix of the second-order partial derivatives of z
A^T	Transpose of the matrix A
$\lceil x \rceil$	The smallest integer not less than x

Defined symbols

$G(V, E)$	Graph with the set of nodes V and the set of links E
$w(v_i, v_j)$	Length of the link (v_i, v_j)
$W(p)$	Length of the path p
$\oplus$	Links joint operation
P_{st}	Set of all possible paths from s to t
$S(p, q)$	Similarity between the paths p and q
$D(p, q)$	Dissimilarity between the paths p and q

Index

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W