HISTORICAL TRAFFIC-TOLERANT PATHS IN ROAD NETWORKS

LI PUI HANG

M.Phil
The Hong Kong Polytechnic University

2015

The Hong Kong Polytechnic University

Department of Computing

Historical Traffic-Tolerant Paths in Road Networks

Pui Hang LI

A thesis submitted in partial fulfillment of the requirements

for the degree of

Master of Philosophy

November 2014

CERTIFICATE OF ORIGINATLITY

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written, nor material that has been accepted for the award of any other degree or diploma, except where due acknowledgement has been made in the text.

.....

Pui Hang LI

November 2014

Abstract

Nowadays, with the extensive installation of roadside sensors and the advancement of mobile technology, historic traffic information is widely collected. It is valuable in transportation analysis and planning, e.g., evaluating the reliability of routes for representative source-destination pairs. Also, it can be utilized to provide efficient and effective route-search services. In view of these applications, this thesis proposes the traffic-tolerant path (TTP) problem in road networks. The problem takes an integer k, a source-destination (SD) pair, and historic traffic information as input, and returns k paths that minimize the aggregate historic travel time. Unlike the shortest path problem, the TTP problem has a combinatorial search space that renders the optimal solution expensive to compute. First, we prove the NP-hardness of the TTP problem. Second, we propose an exact algorithm with effective pruning rules to reduce the search time. Third, we develop an anytime heuristic algorithm that makes 'best-effort' to find a low-cost solution within a given time limit. Extensive experiments on real and synthetic traffic data demonstrate the effectiveness of TTPs and the efficiency of our proposed algorithms.

List of Publications

- Pui Hang Li, Man Lung Yiu, Kyriakos Mouratidis. Historical Traffic-Tolerant Paths in Road Networks. *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (GIS)*, 2014.
- Pui Hang Li, Man Lung Yiu, Kyriakos Mouratidis. Discovering Historic Traffic-Tolerant Paths in Road Networks. *Geoinformatica*, submitted.

Acknowledgements

To begin with, I would like to express my deepest sense of gratitude to my advisors, Dr. Ken Yiu, for his patient guidance. He brings me to explore the research world of database community and provides me with numerous advice so that I can be open-minded and think critically during my study. Without him, I would not have completed this research work.

Next, I give my truthful to Dr. Kyriakos Mouratidis for his useful comments. His suggestions greatly improve the quality of my work.

I would like to thank Dr. Eric Lo since his belief in research becomes an indispensable pointer of my work.

Two heads are always better than one. I have learnt a lot and have got inspired from my database group members during my study. Thanks to Ziqiang Feng, Andy He, Yu Li, Bo Tang, Jeppe Thomsen, Petrie Wong, Wenjian Xu and Qiang Zhang for their comments and encouragement. I am glad to have many joyful moments with all of them.

Last but not least, I wish to thank my parents and my beloved, Tiffany Hui, for their unconditional love and support at all time. Their wholehearted

companion gives me vigor and courage to overcome everyday challenges.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and Motivations

Nowadays, historic traffic information is extensively collected by various means. For example, PeMS [2] developed by California Department of Transportation acquires traffic information (e.g., traffic flow and vehicle speed) through roadside sensors and Google Maps [4] collect floating car data by crowdsourcing techniques. It is valuable in transportation reliability analysis [28, 7], e.g., evaluating the reliability of routes for representative source-destination pairs, and online route services [4]. In order to support these applications, we propose a novel problem called the traffic-tolerant path (TTP) problem. This problem requires a road network $G(V, E, W_m)$ where W_m maps an edge e to a cost vector $w(e)$ in which each component represents the travel time of e at a time instant. Given an integer k and a source-destination (SD) pair (v_s, v_t), the TTP problem aims to find a set of k paths from v_s to v_t, denoted by $P^k_{s,t}$, that minimizes the

following error:

$$\xi(P_{s,t}^k) = \frac{1}{m} \cdot \left(\sum_{j=1}^{m} \min_{p \in P_{s,t}^k} \{\tau_j(p) - \tau_j(sp_j)\} \right) \tag{1.1}$$

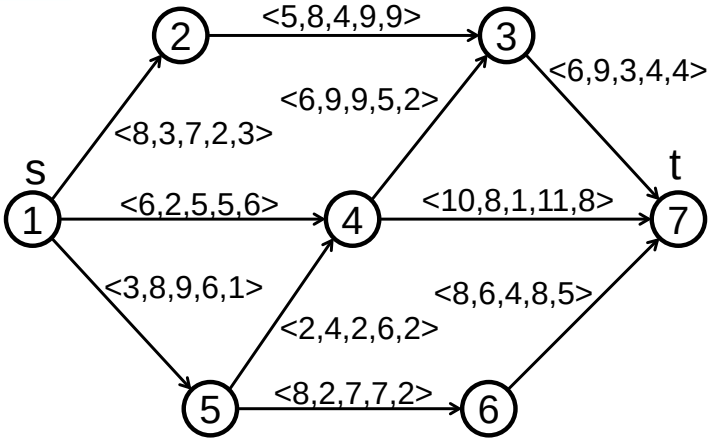
where $\tau_j(p)$ is the travel time of path p at time instant j and sp_j denotes the path with the shortest travel time from v_s to v_t at time instant j.

Observe that the minimization of $\xi(P_{s,t}^k)$ is equivalent to the minimization of the following measure:

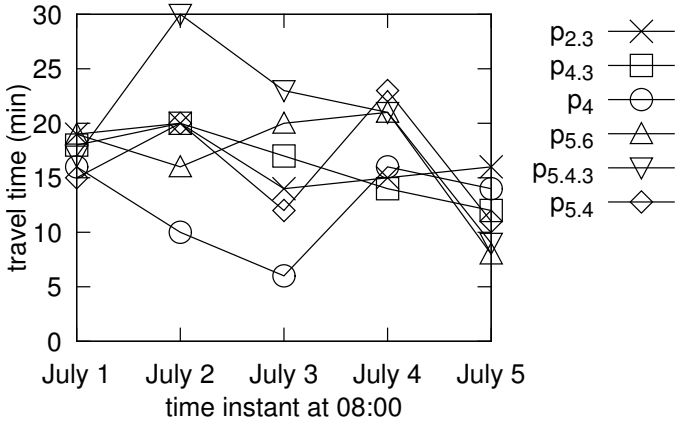
$$\Psi(P_{s,t}^k) = \sum_{j=1}^{m} \min_{p \in P_{s,t}^k} \tau_j(p) \tag{1.2}$$

because sp_j is independent of $P_{s,t}^k$ and the summation function is distributive.

Conceptually, by minimizing $\xi(P_{s,t}^k)$, the TTP problem is to extract a set of k paths such that at least one of them can best approximate the shortest travel times for a given SD pair at any time. We illustrate this problem by the sample road network shown in Figure 1.1(a). Each edge is labeled with 5 weights that represent the travel times of edges at 5 time instants (e.g., 8:00am on July 1 – 5 and i.e., $m = 5$). Suppose that the SD pair (v_s, v_t) is (v_1, v_7). There are 6 possible paths from v_s and v_t, and their travel times at different time instants are shown in Figure 1.1(b). For clarity, we indicate the intermediate nodes in a path in the subscript, e.g., the path $p_{5.4.3}$ passes through v_5, v_4, v_3. Assume that $k = 2$. The optimal path set is $P^k = \{p_4, p_{5.6}\}$ because it has the minimum $\Psi(P^k)$ value of 56.



(a) a directed road network



(b) travel times of paths

Figure 1.1. A running example of the TTP problem

1.1.1 Applications

Application 1: transportation planning analysis

Transportation planners evaluate the reliability of transportation systems by analyzing the reliability of routes for representative SD pairs, which are chosen by their expertise. For example, representative SD pairs could be: city center to airport, ports to the industrial area, etc. Their current practice [28, 7] is to

select only one route per SD pair and calculate the travel time reliability of each route. Our proposed TTPs can provide k paths instead of a single path per SD pair to transport planners for reliability analysis. Since the k paths minimize historic aggregate travel time $\Psi(P^k_{s,t})$, they can be regarded as the alternatives from which planners can obtain a more comprehensive insight of the reliability of transportation systems. For example, Figure 1.2 shows four traffic-tolerant paths of a SD pair in UK highway network and these paths can be used by transport planners for reliability analysis.

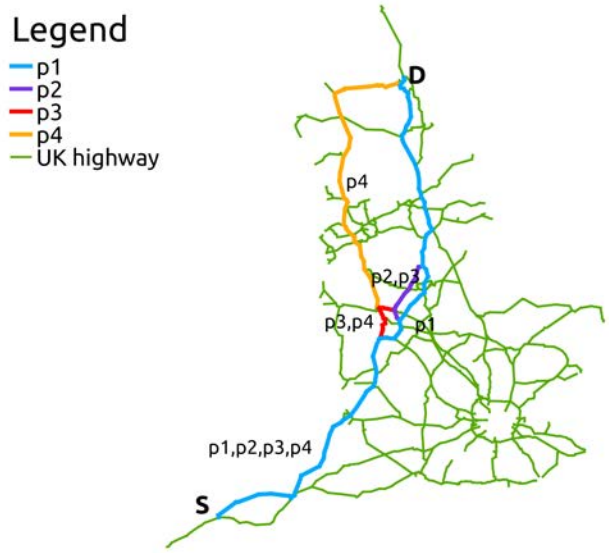


Figure 1.2. Traffic-tolerant paths in UK highway network, $k = 4$

Application 2: online route services

With the wide deployment of mobile devices and the advancement of car navigation systems, online route services have access to real-time traffic information[1] and provide query users with shortest path(s) according to up-to-date traffic. In

[1]Collected by roadside sensors [2], crowdsourcing [4], or purchased from traffic information providers [6].

fact, the majority of queries are *recurrent queries* issued by users, e.g., finding the fastest route from home to office at 8:00am every day. Such regular patterns appear in human trajectories, as revealed in current scientific studies [22, 33].

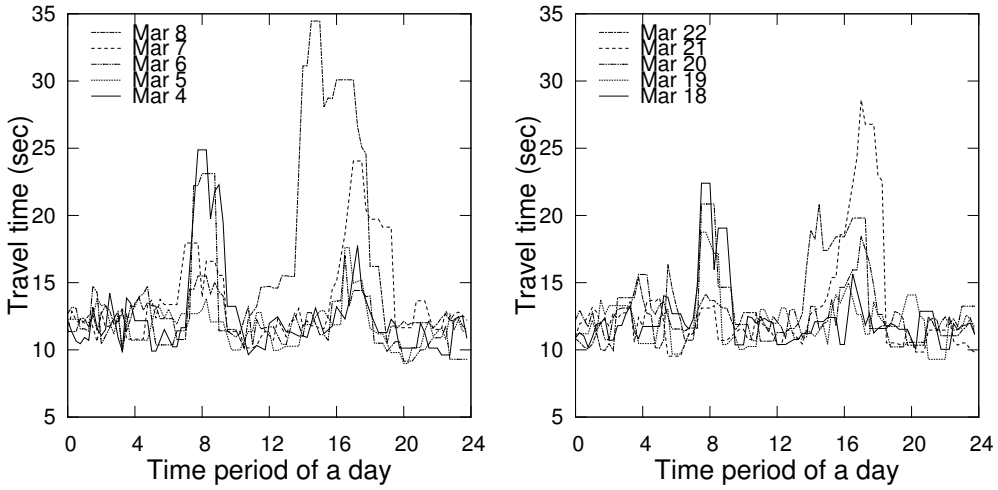
Although an online route service may apply shortest path indices [23, 31, 9, 18, 8, 38] to answer queries efficiently, such indices incur substantial maintenance time[2] with respect to frequent traffic updates (e.g., every 30 seconds [2]), rendering itself hard to answer shortest path queries with the latest traffic.

A promising method [29] is to pre-compute candidate paths (a solution pool) for users' recurrent queries (in an offline phase) and then update their travel times by live traffic information (during online operations). The candidate path with the shortest travel time is used to answer queries. This method is called candidate approach. It eliminates the index maintenance cost and bounds the online computation cost by the number of candidate paths. It scales better than existing methods based on shortest path indices.

In candidate approach, it is desirable to find a set of k candidate paths per query (v_s, v_t) such that *at least one* of such paths is fast for (v_s, v_t) at any time. This problem is challenging due to ever-changing traffic conditions. Although traffic conditions change continuously, they exhibit some patterns which can be exploited for candidate path computations. For example, Figure 1.3 shows the travel times of a road segment on weekdays of two weeks. There are two obvious spikes during 8:00am – 9am and 6:00pm – 7:00pm in these two weeks.

With traffic time patterns of road segments, by minimizing the travel time error, i.e., $\Psi(P^k_{s,t})$, we can obtain a set of traffic-tolerant paths (in Figure 1.2)

[2]The state-of-the-art shortest path index, AH [38], takes hundreds of seconds for index pre-computation on a road network with a million nodes.



(a) Mar 4-8, 2013 (b) Mar 18-22, 2013

Figure 1.3. Travel times of a road segment on A417 Road in UK [5]

and it is most likely that at least one of them equals to or is close to the fastest path under real-time traffic. Although Malviya et al. [29] have proposed some heuristics for finding these candidate paths, they do not necessarily minimize the historic travel time error between those heuristic paths and the fastest path in the road network.

1.1.2 Challenges and Contributions

Unlike the shortest path problem, the TTP problem has a combinatorial search space that renders the optimal solution expensive to compute because it aims to find k paths that minimize the aggregate travel time in history among all possible paths of a SD pair. First, we propose an exact algorithm with effective pruning rules to reduce the search time. Second, we develop an anytime heuristic algorithm that makes 'best-effort' to find a low-cost solution within a given time

limit. In summary, the contributions of this thesis are as follows.

- We propose a novel problem called the traffic-tolerant path (TTP) problem, which finds application in transportation planning and online route services.
- We prove that the TTP problem is NP-hard.
- We present an exact algorithm with effective pruning rules for computing the optimal solution.
- We devise two heuristic algorithms for the TTP problem. One of them offers an anytime feature which can terminate and return a solution within a given time budget.

1.2 Traffic-tolerant Paths

In this section, we present preliminaries and formulate the TTP problem. The following table (Table 1.1) summarizes the notations used in our formulation and in the rest of the thesis.

Table 1.1. Summary of notations

Symbol	Meaning
$G(V, E, W_m)$	A directed and weighted network
$v_s(v_t)$	Source (Destination)
(v_i, v_j)	An edge in E
$w_j(e)$	travel time of $e \in E$ at time instant j
$\mathcal{E}(p)$	The set of edges on path p
$\tau_j(p)$	travel time of p at time instant j
$P^k_{s,t}$	A k-combination
$\Psi(P^k_{s,t})$	Aggregate value of $P^k_{s,t}$

1.2.1 Problem Definition

Definition 1 (Road Network with Historic Traffic) *A road network is modeled as a directed and weighted graph $G(V, E, W_m)$, where V is the set of road junctions, E is the set of road segments, and $W_m : E \to \mathbb{R}^m_+$ is a mapping from edges to m-dimensional cost vectors. Given an edge $e \in E$, we denote its weight vector by $w(e)$ and its travel time at the j-th time instant by $w_j(e)$.*

In real world, $w_j(e)$ may correspond to the travel time of a road segment within the j-th time frame, where the length of each time frame could be, e.g., 30 seconds [2] or 15 minutes [5]. We regard this real-world time frame as *time instant* and each of them is indexed by an integer j. In practice, W_m is usually a subset of the entire historic traffic data and is determined by users according

to their requirements.

DEFINITION 2 (LOOP-FREE PATH) *A loop-free path* $p = \langle v_{a_1}, v_{a_2}, ..., v_{a_n} \rangle$ *is a sequence of distinct nodes such that for* $1 \leq i < n$, $(v_{a_i}, v_{a_{i+1}}) \in E$. *The start and end nodes on* p *are denoted by* $\mathcal{S}(p)$ *and* $\mathcal{T}(p)$ *respectively. They are also called the source* (v_s) *and destination* (v_t) *of* p.

For simplicity, we call a loop-free path as a path in this thesis. The travel time of a path p at time instant j is defined as:

$$\tau_j(p) = \sum_{i=1}^{n-1} w_j((v_{a_i}, v_{a_{i+1}})) = \sum_{e \in \mathcal{E}(p)} w_j(e) \tag{1.3}$$

where $\mathcal{E}(p)$ denotes the set of edges on a path p.

DEFINITION 3 (k-COMBINATION AND AGGREGATE SCORE) *Given a positive integer* k, *a* k-*combination* $P^k_{s,t} = \{p_1, p_2, ..., p_k\}$ *is a set of* k *paths such that all of them have the same start node as* v_s *and the same end node as* v_t. *The aggregate score of* $P^k_{s,t}$ *is defined as*

$$\Psi(P^k_{s,t}) = \sum_{j=1}^{m} \min_{p \in P^k_{s,t}} \tau_j(p) \tag{1.4}$$

We illustrate the above concepts by using the road network in Figure 1.1. Each edge is labeled with a cost vector that represents its travel times at 5 (historic) time instants. Table 1.2 displays all possible paths in the road network and their travel times. For the path $p_1 = \langle v_1, v_2, v_3, v_7 \rangle$, its travel time at time instant 3 is: $\tau_3(p_1) = (7 + 4 + 3) = 14$. Assume that $k = 3$, and consider a 3-combination $P^3_{1,7} = \{p_4, p_5, p_6\}$ for instance. Then, we have: $\min_{p \in P^3_{1,7}} \tau_1(p) =$

$\min\{19, 17, 15\} = 15$ and its aggregate $\Psi(P^3_{1,7})$ is $(15 + 16 + 12 + 21 + 8) = 72$.

Definition 4 (K Traffic-tolerant Paths Query) *Given a road network $G(V, E, W_m)$, $v_s, v_t \in V$, and a positive integer k, Traffic-tolerant Paths Query $\mathsf{TTP}(v_s, v_t, k)$ returns a k-combination $P^k_{s,t}$ such that for any possible $P'^k_{s,t}$, $\Psi(P^k_{s,t}) \leq \Psi(P'^k_{s,t})$.*

Let us use the road network in Figure 1.1 for illustration. Table 1.3 lists all possible 3-combinations and their corresponding aggregates with v_1 as source and v_7 as destination. There are $\binom{6}{3} = 20$ 3-combinations in total. The optimal solution of $\mathsf{TTP}(v_1, v_7, 3)$ is $P_{opt} = \{p_2, p_3, p_4\}$ since its aggregate score $\Psi(P_{opt}) = (16 + 10 + 6 + 14 + 8) = 54$ is the minimum among all combinations.

Table 1.2. All possible paths from v_1 to v_7, with historic travel times

Path	τ_1	τ_2	τ_3	τ_4	τ_5
$p_1\langle v_1, v_2, v_3, v_7\rangle$	19	20	14	15	16
$p_2\langle v_1, v_4, v_3, v_7\rangle$	18	20	17	**14**	12
$p_3\langle v_1, v_4, v_7\rangle$	16	**10**	**6**	16	14
$p_4\langle v_1, v_5, v_6, v_7\rangle$	19	16	20	21	**8**
$p_5\langle v_1, v_5, v_4, v_3, v_7\rangle$	17	30	23	21	9
$p_6\langle v_1, v_5, v_4, v_7\rangle$	**15**	20	12	23	11

Table 1.3. All possible 3-combinations $P^3_{1,7}$ and their aggregate scores

$\{p_1, p_2, p_3\}$: 58	$\{p_1, p_4, p_5\}$: 70	$\{p_2, p_4, p_6\}$: 65
$\{p_1, p_2, p_4\}$: 70	$\{p_1, p_4, p_6\}$: 66	$\{p_2, p_5, p_6\}$: 70
$\{p_1, p_2, p_5\}$: 74	$\{p_1, p_5, p_6\}$: 71	$\{p_3, p_4, p_5\}$: 56
$\{p_1, p_2, p_6\}$: 72	$\{p_2, p_3, p_4\}$: **54**	$\{p_3, p_4, p_6\}$: 55
$\{p_1, p_3, p_4\}$: 55	$\{p_2, p_3, p_5\}$: 55	$\{p_3, p_5, p_6\}$: 56
$\{p_1, p_3, p_5\}$: 56	$\{p_2, p_3, p_6\}$: 56	$\{p_4, p_5, p_6\}$: 72
$\{p_1, p_3, p_6\}$: 57	$\{p_2, p_4, p_5\}$: 72	

1.3 Thesis Organization

After illustrating the background and formulation of the TTP problem, we present the structure of this thesis. The rest of the work is organized as follows. We review the related work in Chapter 2. Then, we present the technical contributions in the following order:

- We prove that the TTP problem is NP-hard by reduction from *Set-Cover* problem. (Chapter 3)
- Next, we present a two-phase exact algorithm with effective pruning rules for computing the optimal solution. The two phases are candidate generation and optimal solution enumeration respectively. Due to the hardness of the TTP problem, we present two heuristic algorithms. One of them bounds the number of candidates in Phase I while another offers an anytime feature which can terminate and return a solution within a given time budget. (Chapter 4)
- Finally, we evaluate our proposed algorithms through extensive experiments. The evaluation is conducted on both real and synthetic data sets. The results of the experiments show that our approaches outperform their competitors. (Chapter 5)

After presenting our contributions, we conclude our work and envision our future research directions in (Chapter 6).

Chapter 2

Literature Review

In this chapter, we review the literature related to the TTP problem. Since it makes use of historic traffic data and has multidisciplinary applications, its related work includes i) road networks with historic traffic, ii) shortest path index, iii) multi-criteria queries in road networks, and iv) reliable route recommendation.

2.1 Road Networks with Historic Traffic

2.1.1 Time-dependent Networks

The first approach of incorporating historic traffic into road networks is time-dependent network. Kanoulas et al. [25] and Demiryurek et al. [16] associated road segments with time-varying functions which are established based on historic traffic data. Time-varying functions capture the traffic patterns of road

segments and return travel times of edges given a start time t. A time-varying function of a road segment is shown in Figure 2.1. The function indicates that peak hours occur on this road segment at 8:00am and 5:00pm. They extended A* algorithm to compute the time-dependent fastest path between source and destination in a time-dependent network. Gonzalez et al. [21] exploited the driving and road traffic patterns from historic data in order to consider some non-trivial factors in route planning such as the experience of local expert drivers. Nevertheless, when answering shortest paths queries, their works need to traverse the road network and hence the query cost is not *bounded*. Also, their routing algorithms only consider the time-varying functions built on historic traffic but do not react with real-time traffic updates. Our TTP problem enjoys bounded query cost by pre-computing the candidates of recurrent queries and responds to live traffic information by re-ranking the candidates.

2.1.2 Probabilistic Networks

Probabilistic network proposed by Hua et al. [24] is another way to model historic traffic information. It takes travel time samples of each edge from historic traffic data and constructs *probability mass functions* (PMF) for each segment. Figure 2.2 is a probabilistic network example. The edges in the network are associated with a set of weights while every weight is accompanied with a probability, forming a PMF. Based on probabilistic networks, they further proposed probabilistic path queries and leveraged basic conditional probability principle to compute the paths that satisfy a weight (travel time) requirement l guaranteed by a certain probability τ. The queries require users to specify either l or τ. In transportation analysis, it is reasonable for planners to specify τ so as to find

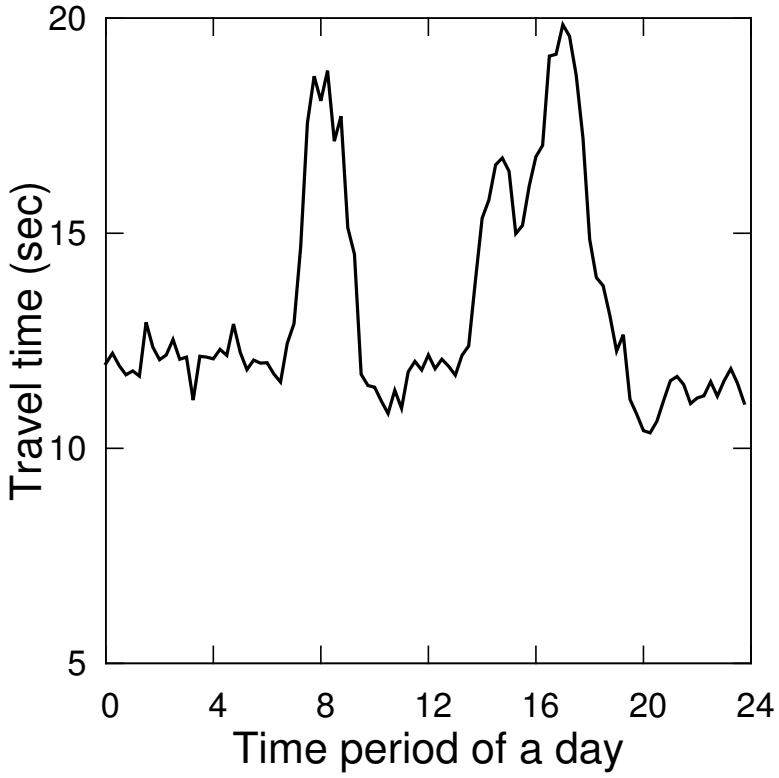
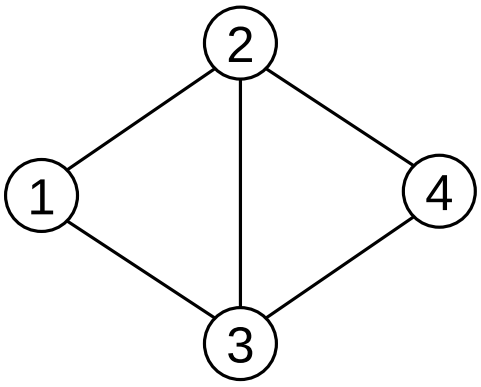


Figure 2.1. Time-varying function of a road segment on A417 Road in UK [5]

highly reliable paths. However, in online route services, both l and τ vary with different SD pairs in practice and the selection of τ is not discussed in their work. On the contrary, the TTP problem only requires a straightforward parameter of k and can be applied to both applications.



(a) an undirected road network

Edge	Weight (Probability)		
(1,2)	10(0.4)	20(0.3)	25(0.3)
(1,3)	15(0.5)	20(0.3)	25(0.2)
(2,3)	5(0.3)	20(0.3)	25(0.4)
(2,4)	5(0.6)	10(0.2)	15(0.2)
(3,4)	10(0.6)	15(0.1)	20(0.3)

(b) probabilistic weights of edges

Figure 2.2. A probabilistic road network

2.2 Shortest Path Index

The classic solution to shortest path queries in road network is Dijkstra's algorithm [17]. It starts at a source and visits the nodes in the ascending order of their distance from the source by maintaining a priority queue. It can be further accelerated by bi-directional searches. However, it is not able to answer queries efficiently in sizable road networks. To accelerate the shortest path computation in real time, pre-computed indices have been widely studied. They can be classified into three categories, namely *goal directed approach*, *road hierarchical approach*, and *candidate approach*. We give a brief review on these three approaches in the following sections.

2.2.1 Goal Directed Approach

Conceptually, goal directed indices act as a tour guide and direct the shortest path search towards the destination. ALT [20], Arc-Flags [26] and hub-based labeling [8] are the representative algorithms in this category.

In the pre-computation phase, ALT picks a set of landmark nodes and computes the distances from them to every nodes in the network. With triangle inequality, lower bound distances from the visited nodes to the destination can be derived from the pre-computed distance information and used as admissible estimates in A* search. Arc-Flags first partition the network into m subgraphs and compute a bitmap index B for each edge. The bitmap of each edge indicates if the edge lies on a shortest path to a node in subgraphs. Both of them have been improved to support weight changes of edges [14, 13] for online route services. However, their index maintenance costs are huge and their query performances degrade with weight changes. Although hub-based labeling [8] offers promising response time among all start-of-the-art techniques, it does not support weight updates of edges and its re-computation cost is expensive also.

2.2.2 Road Hierarchical Approach

Road hierarchical approach exploits the implicit hierarchical structure of road network and store this information in a pre-computed index. The state-of-the-art techniques include Reach [23], highway hierarchy (HH) [31] [32], transit node routing (TNR) [9], contraction hierarchy (CH) [18], and arterial hierarchy (AH) [38].

Reach, HH, CH, and AH are based on similar concepts - some nodes are more superior than the others and hence the node ordering can be defined. Shortcut edges, which bypass inferior nodes, are built among those important (high-order) nodes in order to reduce the search space during shortest path search.

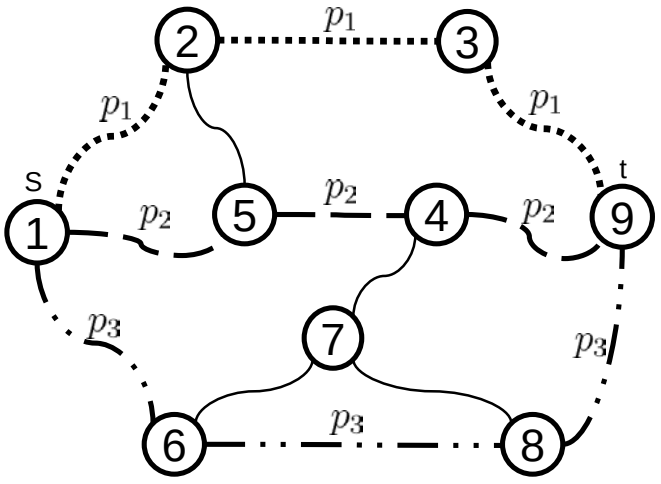
Although CH and AH can pre-compute their indices in hundreds of seconds

in [34, 38], they are not still applicable when live traffic updates may arrive frequently such as 30 seconds in [2]. Geisberger et al. [19] devised an update mechanism for CH for real traffic condition. However, the mechanism only supports weight increases and the update time is highly dependent on the order of the two end nodes of the updated edge. If the end nodes have higher orders, more shortcut edges are affected and their weight need to be updated accordingly. The other two techniques, Reach and TNR, have been evaluated to have much higher pre-computation cost than CH through extensive experiments in [10, 34].

2.2.3 Candidate Approach

As revealed in current scientific studies [22, 33], regular patterns appear in human trajectories, implying the recurrence of route queries which allows *candidate approach* for online route services. Candidate approach pre-computes candidate paths (a solution pool) for users' recurrent queries (in an offline phase) and then update their travel times by live traffic information (during online operations). For example, three paths from v_1 to v_8 (p_1, p_2 and p_3) shown in Figure 2.3 are pre-computed offline in advance. They are re-ranked based on the live traffic updates online. The live travel times of p_1, p_2 and p_3 are 14, 13, and 12 respectively. The path (p_3) with the shortest travel time is returned as a result.

Chen et al. [11] in transportation community first proposed a candidate approach for route recommendation. Their approach takes drivers' preferences such as maximum path duration and minimum path reliability into considerations. Link reliability is estimated by analysis of historical data and microscopic



Live Traffic

Edge	Weight
(1,2)	3
(1,5)	4
(1,6)	2
(2,3)	6
(2,5)	1
(3,9)	5
(4,7)	2
(4,9)	4
(5,4)	5
(6,7)	3
(6,8)	4
(7,8)	3
(8,9)	6

Figure 2.3. An undirected road network for candidate approach

traffic simulation. They proposed a heuristic method with a weight increment procedure in order to compute a set of candidate paths which are partially disjoint and with acceptable reliability.

Malviya et al. [29] proposed some heuristics to generate candidate paths for online route services with bounded cost. Two representative heuristic methods are *K-variance* (K-VAR) and *Y-moderate* (Y-MOD). The idea of K-VAR is to build a Gaussian distribution $\mathcal{N} \sim (\mu, \sigma^2)$ based on the historic traffic data for each edge. Then, for each edge, it samples the travel time from the distribution $\mathcal{N}$ of the edge and computes a shortest path from v_s to v_t. It repeats the travel time sampling and the shortest path search after a fixed number of iterations in order to find k candidate paths. Y-MOD is a variant of Yen's algorithm [36] which computes k loop-free shortest paths. Its idea is to guarantee that the resulting k candidate paths have a limited fraction f of overlapping edges.

The heuristics in [11, 29] do not necessarily minimize the travel time error

between those candidate paths and the fastest paths in the historic data. In contrast, our TTP problem aims to minimize the historic travel time error in order to produce a set of candidate paths with resistance to the real-time traffic. Their work cannot be used for transportation reliability analysis as the heuristics may return some convoluted paths, which are not considered by transport planners.

2.3 Queries in Multi-criteria Road Networks

Multi-criteria road network is a road network whose edges are associated with multiple attributes. The attributes present different criteria/costs of a road segment such as driving time, walking time and physical length. A multi-criteria road network example is shown in Figure 2.4. The different criteria of a road segment in this network are driving time, gasoline consumption, and toll fee. The TTP problem is also related to queries in multi-criteria road networks since it associates each edge in a road network with a multidimensional cost vector. The criteria in the TTP problem are the travel times of edges at different time instants.

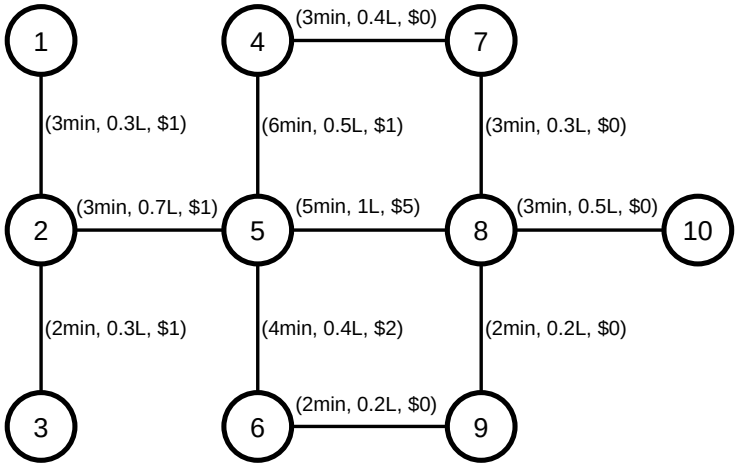


Figure 2.4. A multi-criteria undirected road network example

Queries in multi-criteria road networks have also been studied in data engineering discipline [30, 27] for transportation decision making. Mouratidis et al. [30] proposed skyline and top-k queries on a set of point of interests (POIs) in multi-cost road networks. Given a query location q, their skyline query com-

putes a skyline of POIs such that their cost vectors with respect to q are not dominated by one another. The top-k query takes a query location q and an increasingly monotonic aggregate function f as inputs and then returns k POIs with the highest value of f.

Meanwhile, Kriegel et al. [27] proposed route skyline queries in multi-cost road networks for multi-preference route planning. Given a SD pair (v_s, v_t), they aim to compute a set of skyline paths from v_s to v_t such that the paths can offer an optimal solution of any *arbitrary* preference functions.

Although the works above both focus on multi-cost road networks, they are different from the work in this thesis. First of all, our problem computes k paths of a given SD pair instead of a set of skyline or top-k POIs with respect to a query location [30]. Second, although the work in [27] and our work both regard paths as multidimensional points and exploit dominance property, our TTP problem is distinct from their skyline path computation since it aims to minimize the aggregate travel time error Ψ.

2.4 Reliable Route Recommendation

Transportation community has also studied multi-criteria road networks for reliable route recommendation [35, 37]. In view of traffic uncertainties, Yu et al. [37] adopted a *scenario* approach and first proposed absolute robust shortest path (ARSP) problem. They considered that travel times of road segments (and hence paths) vary with scenarios. A scenario is an instance of traffic conditions. Therefore, scenarios can be a set of representative times such as rainy day and sunny day, or can be a set of time instants like 9:00am - 9:15am of weekdays. ARSP problem aims to find a path that can minimize the maximum travel time in all scenarios. Let us consider the road network in Figure 2.2(a) again and Table 2.1 shows the travel times of four paths from v_1 to v_4 in four scenarios. p_2 is the optimal ARSP since its maximum travel time is the minimum among all four paths in all scenarios.

Xing et al. [35] re-visited ARSP problem and proposed percentile robust shortest path (PRSP) problem. PRSP is to compute a path that has the minimum α-percentile travel time. For instance, assume $\alpha = 75$, p_1 and p_2 are both α-optimal because their 75-percentile travel times, which are both 11, are minimum according to Table 2.1. They formulated ARSP and PRSP problems as mathematical optimization problems and solved them by Lagrangian relaxation approach. The above works compute only one reliable path whereas our TTP problem returns k paths which are more resistant to ever-changing traffic conditions.

Edge	**Scenarios**				**75-percentile**
	s_1	s_2	s_3	s_4	
$p_1\langle v_1, v_2, v_4\rangle$	9	13	11	10	**11**
$p_2\langle v_1, v_3, v_4\rangle$	9	10	11	12	**11**
$p_3\langle v_1, v_2, v_3, v_4\rangle$	18	18	21	21	21
$p_4\langle v_1, v_3, v_2, v_4\rangle$	10	11	13	13	13

Table 2.1. An example of ARSP and PRSP based on Figure 2.2(a)

Chapter 3

Problem Hardness

In this chapter, we prove the TTP problem is NP-hard by reduction from the Set-Cover problem, paving the way of devising heuristics in Section 4.2.

THEOREM 1 *The TTP problem is NP-hard.*

Proof. Before presenting a reduction scheme, we first express an instance of the TTP problem, $\langle G, v_s, v_t, k\rangle$, in another form which facilitates our proof. Recall that every path on G has a m-dimensional cost vector and the TTP problem aims to find an optimal combination of k paths among all possible paths from v_s to v_t. Let us assume there are n possible paths between v_s and v_t. Instead of expressing the problem from the perspective of graph, we can model it as a matrix like in Table 1.2. Each row represents the m-dimensional cost vector a path from v_s to v_t. Based on matrix-like representation, we can re-formulate an instance of the TTP problem as $\langle k, m, n, \{(i, j, \tau_j(p_i))\}\rangle$, where the set $\{(i, j, \tau_j(p_i))\}$ records the travel time of each path p_i at each time instant j, for $1 \leq i \leq n$ and $1 \leq j \leq m$.

Next, we present a *reduction scheme* that converts any given instance of the *Set-Cover problem* [12]. Let $\langle k, CS = \{S_i\}, U\rangle$ be an instance of Set-Cover, where k is an integer, U is a domain set of items, CS is a collection of subsets $S_i \subseteq U$. This problem asks whether there exists a size-k collection $CS' \subseteq CS$ such that they cover all items in U, i.e., $|\bigcup_{S_i \in CS'} S_i| = |U|$.

The reduction scheme is as follows:

- We set $m = |U|$ and $n = |CS|$. The value of k is the same in both problems.
- Without loss of generality, we rename the items in U as $1, 2, \cdots, m$ (in the Set-Cover instance).
- for each subset $S_i \in CS$ and each item $j \in U$, we set $\tau_j(p_i) = 0$ if $j \in S_i$, or set $\tau_j(p_i) = 1$ otherwise.

There exists a graph instance of the TTP problem to which any given instance of the *Set-Cover problem* corresponds. The graph instance, in which the paths connecting v_s and v_t $(p_1, p_2, ..., p_n)$ are disjoint, is shown in Figure 3.1. Note that the size of the constructed TTP instance $\langle k, m, n, \{(i, j, \tau_j(p_i))\}\rangle$ is polynomial to the size of the given Set-Cover instance $\langle k, CS = \{S_i\}, U\rangle$. Also, the construction process takes polynomial time.

Now, we show that a solution CS' of Set-Cover instance corresponds to a solution P of the corresponding TTP instance with aggregate score $\sum_{j=1}^{m} \min_{p \in P} \tau_j(p)$ equal to 0.

We first convert a given $CS' = \{S_{x_1}, S_{x_2}, \cdots, S_{x_k}\}$ to a corresponding $P = \{p_{x_1}, p_{x_2}, \cdots, p_{x_k}\}$ and then derive its aggregate score. By checking the

occurrences of items in S_{x_i} in the item-wise manner, we derive:

$$|\bigcup_{S_{x_i} \in CS'} S_{x_i}| = \sum_{j=1}^{m} \mathcal{B}(\bigvee_{i=1}^{k} (j \in S_{x_i}))$$

where the function $\mathcal{B}(\cdot)$ maps `true` to 1 and maps `false` to 0. Since CS' is a solution of Set-Cover, each item $j \in U$ must appear in some S_{x_i}. According to our reduction scheme, the corresponding $\tau_j(p_{x_i})$ in our constructed instance must be 0. Thus, we obtain: $\sum_{j=1}^{m} \min_{i=1}^{k} \tau_j(p_{x_i}) = 0$.

We then convert a given $P = \{p_{x_1}, p_{x_2}, \cdots, p_{x_k}\}$ to a corresponding $CS' = \{S_{x_1}, S_{x_2}, \cdots, S_{x_k}\}$. Since P is a solution, for each time instant j, there exists some path p_{x_i} such that $\tau_j(p_{x_i}) = 0$. According to our reduction scheme, for each item j, there is a corresponding S_{x_i} (in Set-Cover solution) that contains j. Therefore, CS' covers all items in U, and it is a solution of Set-Cover.

Since the Set-Cover problem is NP-hard [12], this proof implies that the TTP problem is also NP-hard. □

It is tempting to adapt existing heuristics for the Set-Cover problem to solve our TTP problem. Observe that, in the above reduction scheme, any given Set-Cover instance can be converted to a corresponding TTP instance with binary $\tau_j(p_i)$ (i.e., either 0 or 1). However, in a general TTP instance, the values of $\tau_j(p_i)$ can be real numbers. Such a TTP instance does not have a corresponding Set-Cover instance. Thus, existing heuristics for the Set-Cover problem are not applicable to our TTP problem.

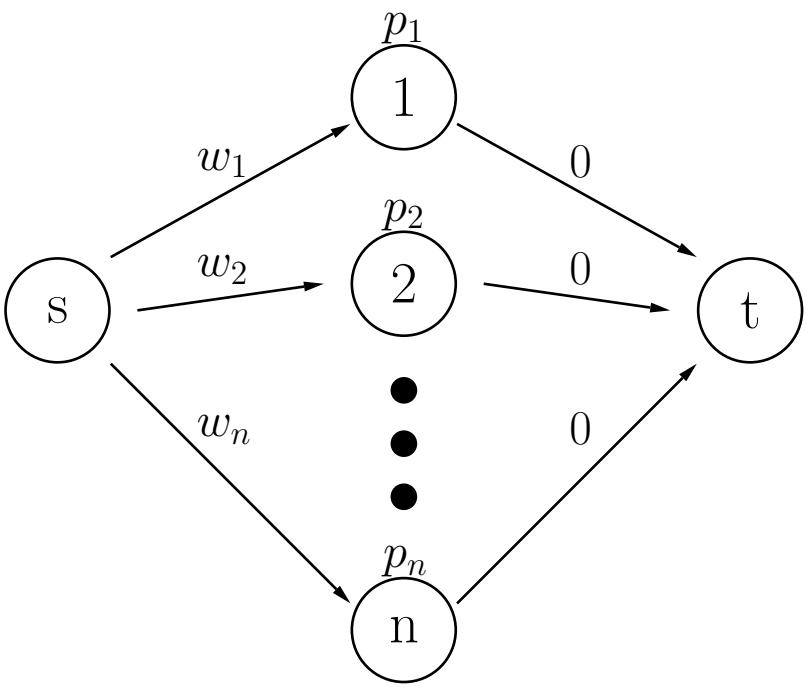


Figure 3.1. A graph instance corresponding to any given instance of the *Set-Cover problem*

Chapter 4

Algorithms

This chapter presents our technical contributions which are an exact algorithm and two heuristics for the TTP problem. They are explained with the aid of detailed examples. Finally, we are going to discuss two practical aspects of TTPs, namely selection of historic traffic data and updating TTPs in online route services.

4.1 Exact Method

In this section, we present an exact method for the TTP problem. Our exact method (Algorithm 1) consists of two phases: **(Phase I)** generating a set $\mathcal{C}$ of candidate paths and **(Phase II)** finding the optimal combination of k paths from the set $\mathcal{C}$.

A simple implementation is to enumerate all possible paths from v_s to v_t and then examine all size-k combinations of the paths. As an example, we assume

$k = 3$ and consider the SD pair (v_1, v_7) in the road network in Figure 1.1. Since there are $|\mathcal{C}| = 6$ possible paths (from v_1 to v_7) in the road network, we would enumerate $\binom{|\mathcal{C}|}{k} = \binom{6}{3} = 20$ size-3 combinations in total. However, this implementation does not scale well with a large road network.

In the light of this, we optimize the algorithms for both phases to reduce the search space $\binom{|\mathcal{C}|}{k}$. For Phase I, we develop pruning rules to eliminate unpromising paths that cannot contribute to the optimal solution (i.e., reducing the value $|\mathcal{C}|$). For Phase II, we adopt the branch-and-bound paradigm and design pruning rules to discard partial combinations that cannot lead to the optimal solution.

Algorithm 1 Exact (Node v_s, Node v_t, Integer k)

1: $\mathcal{C} \leftarrow$ GenerateCandidates (v_s, v_t, k) ▷ Phase I
2: $P_{opt}^k \leftarrow$ FindOptimal (k, $\mathcal{C}$) ▷ Phase II
3: **return** P_{opt}^k

4.1.1 Phase I: Generating Candidates

We face two challenges in generating candidate paths. First, the number of all possible paths from source to destination is incredibly large in a sizable road network. Exploring all of them is impractical. Second, many paths lead to long travel times, so those paths should not be included in the optimal solution.

To overcome the challenges, we prune unpromising paths by leveraging the dominance property. Since every edge in the road network has a cost vector $w(e)$ with size m, all possible paths p connecting a SD pair has a m-dimensional cost vector $\overrightarrow{p}$ also. As p represents a vector, we can define and exploit dominance of paths.

DEFINITION 5 (VECTOR AND PATH DOMINANCE) *Let $\overrightarrow{v}$ and $\overrightarrow{u}$ be two m-dimensional vectors. $\overrightarrow{v}$ is said to dominate $\overrightarrow{u}$ if and only if $\forall 1 \leq j \leq m, \overrightarrow{v}.j \leq \overrightarrow{u}.j$. We denote this as $\overrightarrow{v} \preceq \overrightarrow{u}$.*

Let p and p' be two paths from v_s to v_t. p' is said to dominate p if and only if $\overrightarrow{p'} \preceq \overrightarrow{p}$.

LEMMA 1 (DOMINANCE PROPERTY) *Given two vectors $\overrightarrow{v}$ and $\overrightarrow{u}$, if $\overrightarrow{v} \preceq \overrightarrow{u}$, then $\sum_{j=1}^{m} \overrightarrow{v}.j \leq \sum_{j=1}^{m} \overrightarrow{u}.j$.*

The concept of path dominance is visualized in Figure 4.1. A dimension is the travel time of a path at a particular time instant and a point represents a path from v_s and v_t. Every point corresponds to a dominance region. For example, p_3 with $\overrightarrow{p_3} = \langle 3, 4 \rangle$ dominates the points in the blue area , and p_4 with $\overrightarrow{p_4} = \langle 4, 2 \rangle$ dominates the paths in the red area.

Recall that the TTP problem is to minimize the aggregate value derived by the paths in a k-combination. Thus, we aim to retain those paths with short travel times over m time instants. By the concept of path dominance, if a path p' dominates another path p, this implies p' has smaller travel times than p at all time instants, so p should be pruned in Phase I. This dominance concept leads to the following pruning rule.

Table 4.1. Cost vector representations of paths and combinations

Type	**Cost Vector** $\overrightarrow{v} = \langle \overrightarrow{v}.1, \overrightarrow{v}.2, \cdots, \overrightarrow{v}.m \rangle$
Path p	$\overrightarrow{p} = \langle \cdots, \tau_j(p), \cdots \rangle$
Combination P	$\overrightarrow{P} = \langle \cdots, \min_{p \in P} \tau_j(p), \cdots \rangle$
Prefix path $\hat{p}$	$LB(\hat{p}) = \langle \cdots, LB_j(\hat{p}), \cdots \rangle$

PRUNING RULE 1 (PATH DOMINANCE PRUNING) *Given a path p, if there is a*

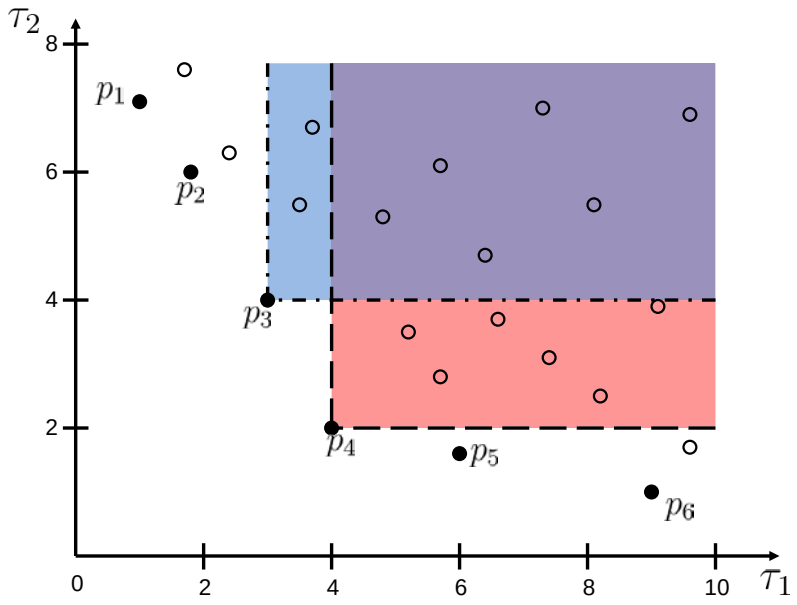


Figure 4.1. Concept of path dominance (τ_1 and τ_2 are historic travel times of paths)

path p' such that $\overrightarrow{p'} \preceq \overrightarrow{p}$, then p can be pruned.

Proof. For the sake of discussion, we let $p_1 = p$ and $p'_1 = p'$. Consider a k-combination that contains p_1, say $P = \{p_1, p_2, ..., p_k\}$. Suppose there is a path p'_1 such that $\overrightarrow{p'} \preceq \overrightarrow{p_1}$ and another k-combination is formed $P' = \{p'_1, p_2, ..., p_k\}$.

Consider the j-th dimension of the cost vectors and $\overrightarrow{P}$ and $\overrightarrow{P'}$. Observe that $\overrightarrow{P}.j = \min\{\tau_j(p_1), \tau_j(p_2), \cdots, \tau_j(p_m)\}$ and $\overrightarrow{P'}.j = \min\{\tau_j(p'_1), \tau_j(p_2), \cdots, \tau_j(p_m)\}$. Since $p'_1 \preceq p_1$, we have $\overrightarrow{P'}.j \leq \overrightarrow{P}.j$. By Lemma 1, we derive $\sum_{j=1}^{m} \overrightarrow{P'}.j \leq \sum_{j=1}^{m} \overrightarrow{P}.j$. Thus, $\Psi(P') \leq \Psi(P)$ and the lemma is proven. □

This pruning rule is applicable only when all possible paths between v_s and

v_t are known. However, enumerating all possible paths is expensive even in a medium-sized road network. Consequently, we aim to avoid exploring the entire search space during enumeration.

Observe that many paths share a common prefix among all possible paths from v_s to v_t. During path enumeration, we can safely disqualify a prefix path (before it reaches v_t) by computing the minimum possible travel time of the paths originating from a common prefix at each time instant j, denoted by $LB_j(\hat{p})$. It is a sum of two terms: (i) the exact travel time of a prefix at time instant j, i.e., $\tau_j(\hat{p})$ and, (ii) the *minimum* travel time from the end node of the prefix to v_t. A prefix path $\hat{p}$ and $LB_j(\hat{p})$ are defined as follows.

Definition 6 (Prefix Path) *Given a path* $p = \langle v_{a_1}, v_{a_2}, ..., v_{a_n} \rangle$, $\hat{p} = \langle v_{\hat{a}_1}, v_{\hat{a}_2}, ..., v_{\hat{a}_m} \rangle$ *is a prefix path of* p *if and only if* $m \leq n$ *and for* $1 \leq i \leq m, v_{a_i} = v_{\hat{a}_i}$.

Definition 7 (Travel Time and Cost Vector of Prefix Path) *Given a prefix path* $\hat{p}$, *for* $1 \leq j \leq m$, $LB_j(\hat{p})$ *is calculated as*

$$LB_j(\hat{p}) = \tau_j(\hat{p}) + \tau_j(sp_j^{last})$$

where $\tau_j(\hat{p}) = \sum_{e \in \mathcal{E}(\hat{p})} w_j(e)$ *denotes the exact travel time of a prefix at time instant* j *and* sp_j^{last} *denotes the shortest path from the last node of* $\hat{p}$ *to* v_t *at time instant* j.

The lower-bound cost vector of $\hat{p}$, *denoted by* $LB(\hat{p})$, *is defined as* $LB(\hat{p}) = \langle LB_1(\hat{p}), LB_2(\hat{p}), ..., LB_m(\hat{p}) \rangle$.

For example, let us consider $\hat{p} = \langle v_1, v_5 \rangle$ and $LB_2(\hat{p})$ in our sample network. We

have $\tau_2(\hat{p}) = 8$ and $\tau_2(sp_2^{last}) = 8$, where $sp_2^{last} = \langle v_5, v_6, v_7 \rangle$. Hence, $LB_2(\hat{p}) = (8 + 8) = 16$. By computing $LB_j(\hat{p})$ for $1 \leq j \leq m$, we obtain a cost vector $LB(\hat{p})$ that lower bounds the cost vector of any path sharing the common prefix $\hat{p}$. Similarly, we can apply the dominance concept and the pruning rule for $LB(\hat{p})$.

PRUNING RULE 2 (PREFIX PATH PRUNING) *Given a set of paths $\mathcal{D}$ and a prefix path $\hat{p}$, if there is a $p' \in \mathcal{D}$ such that $\overrightarrow{p'} \preceq LB(\hat{p})$, then every path p with the prefix $\hat{p}$ can be pruned.*

Algorithm 2 DepthFirstSearch *implements* GenerateCandidates

Algorithm DepthFirstSearch (Node v_s, Node v_t, Integer k)
1: Initialize $\hat{p} \leftarrow \langle v_s \rangle$
2: Initialize $\mathcal{C} \leftarrow \varnothing$
3: Compute the shortest path distances from $v \in V$ to v_t for each time instant j by backward Dijkstra's algorithm
4: $\mathcal{D} \leftarrow$ compute a set of paths by heuristics ▷ Section 4.1.1.1
5: RecurDFS (v_s, v_t, $\hat{p}$, $\mathcal{D}$, $\mathcal{C}$) ▷ candidates stored in $\mathcal{C}$
6: add $\mathcal{D}$ into $\mathcal{C}$ ▷ for correctness
7: **return** $\mathcal{C}$

Algorithm RecurDFS (Node v_s, Node v_t, Path $\hat{p}$, Path set $\mathcal{D}$, Path set $\mathcal{C}$)
1: $u_{last} \leftarrow \mathcal{T}(\hat{p})$
2: **if** $u_{last} \neq v_t$ **then**
3: **if** $\forall p \in \mathcal{D}, LB(\hat{p})$ is not dominated by $\overrightarrow{p}$ **then** ▷ pruning rule
4: **for** each node v adjacent to u_{last} **do**
5: **if** $v \notin \hat{p}$ **then**
6: append v to $\hat{p}$
7: RecurDFS (v_s, v_t, $\hat{p}$, $\mathcal{D}$, $\mathcal{C}$)
8: remove v from $\hat{p}$
9: **else**
10: $\mathcal{C} \leftarrow \mathcal{C} \cup \{\hat{p}\}$

We present the implementation of Phase I in Algorithm 2. First, we compute $\tau_j(sp_j^{last})$ using backward Dijkstra's algorithm from v_t and apply a heuristic to find a set of pruning paths $\mathcal{D}$ (Line 3 and 4). All of them will be used to

support our pruning rules. We will discuss how to select the set $\mathcal{D}$ shortly (Section 4.1.1.1). Next, we initialize an empty prefix path and an empty candidate set $\mathcal{C}$. Then, we apply a DFS-like procedure to find all qualified candidates. The procedure recursively constructs different prefixes stemming from v_s and compares them against $\mathcal{D}$. If a prefix is dominated by a path in $\mathcal{D}$, the algorithm discards and stops expanding the prefix. Otherwise, the recursion keeps expanding the prefix by visiting the neighbors of its end node until reaching v_t. Each completely expanded path (from v_s to v_t) becomes a candidate for the next phase.

THEOREM 2 *Only non-dominated paths can contribute to the optimal solution of the TTP problem.*

Proof. Based on Pruning Rule 1, all dominated paths are disqualified to form an optimal combination since they can be replaced by the paths which dominate them. Hence, only non-dominated paths may belong to the optimal solution. □

THEOREM 3 *The size of $\mathcal{C}$ is $O(|V|!)$*

Proof. Let us consider a completely connected graph $G(V, E)$. Given any two nodes in G, there are $\sum_{i=0}^{|V|-2} P_i^{|V|-2} = (|V|-2)!(\sum_{i=0}^{|V|-2} \frac{1}{i!})$ paths between them, where P_k^n denotes permutation. If the number of dimensions of cost vectors is large enough such that all paths connecting any two nodes in G are non-dominated among one another, the size of $\mathcal{C}$ is factorial in $|V|$. □

4.1.1.1 Selection of the Pruning Path Set $\mathcal{D}$

In the previous section, we mentioned that every path p corresponds to distinct dominance regions. The selection of $\mathcal{D}$ is critical for pruning efficiency. Including too many paths in $\mathcal{D}$ may actually cripple performance because pruning becomes too slow (we need to check every element against a prefix path one by one). We have empirically found that $\mathcal{D}$ yields the best performance in most cases when it includes the $k + 1$ paths described below. Note that this does not affect correctness, but only determines the trade-off between the effectiveness of pruning and its processing overhead.

We first introduce a new notation of edges, $w_s(e)$. It represents the weight of an edge dedicated to shortest path search and is used to find $\mathcal{D}$. $w_s(e)$ is introduced in order *not* to mix with $w_j(e)$. Initially, for every edge, we add up the travel time of the edge of all m time instants, i.e.,

$$w_s(e) = \frac{1}{m} \sum_{j=1}^{m} w_j(e) \tag{4.1}$$

Then, we compute the shortest path between v_s and v_t using $w_s(e)$ and insert it into $\mathcal{D}$. The intuition is that a path is not bad at all time instants.

Next, in the road network, we set the weight of each edge e to $w_s(e) = \min_{j=1}^{m} w_j(e)$. We then execute the following steps for k iterations. In each iteration, we perform a shortest path search from v_s to v_t, obtain a shortest path sp_i, and insert it into $\mathcal{D}$. Then, we update $w_s(e) = \max_{j=1}^{m} w_j(e)$ for each edge e on sp_i. We hope that this would force the shortest path search in subsequent iterations to find other paths that are significantly different from sp_i.

4.1.2 Phase II: Finding Optimal Combination

In this section, we present the implementation for Phase II, i.e., enumerating k-combinations of paths from the candidate set $\mathcal{C}$, in order to find the optimal k-combination. Since we have obtained the cost vectors of candidate paths in Phase I, the road network is no longer required in this phase.

The number of k-combinations of paths is $\binom{|\mathcal{C}|}{k}$, so it is expensive to generate them, especially when $|\mathcal{C}|$ is large. Thus, we develop pruning rules to prevent exploring unnecessary k-combinations. The key idea of the pruning rules is to early stop extending any *partial combination* $\hat{P}$ (which contain fewer than k candidates) by computing its lower bound aggregate value.

Table 1.3 lists out all 3-combinations and their aggregate travel time errors. Observe that some of the combinations share common paths. For example, $\{p_1, p_2, p_3\}$, $\{p_1, p_2, p_4\}$, $\{p_1, p_2, p_5\}$, and $\{p_1, p_2, p_6\}$ have $\{p_1, p_2\}$ as common paths. Suppose that $P_{best} = \{p_1, p_2, p_3\}$ is the best combination found so far and $\Psi(P_{best})$ is 58. Let us consider a partial combination $\hat{P} = \{p_1, p_2, p_*\}$, where $\{p_1, p_2\}$ are *fixed paths* $\hat{P}_f$ and p_* is a *variable* path selected from $\hat{P}_v = \{p_4, p_5, p_6\}$. We can derive the lower bound aggregate value of $\hat{P}$, denoted by $\Psi_{lb}(\hat{P})$, using Definition 8. $\Psi_{lb}(\hat{P})$ is calculated as $\sum_{j=1}^{m} \min\{\min_{p \in \{p_1, p_2\}} \tau_j(p), \min_{p_* \in \{p_4, p_5, p_6\}} \tau_j(p_*)\} = 15 + 16 + 12 + 14 + 8 = 65$ according to Table 1.3. Since $\Psi_{lb}(\hat{P}) > \Psi(P_{best})$, we can safely discard three combinations derived from $\hat{P}$, namely $\{p_1, p_2, p_4\}$, $\{p_1, p_2, p_5\}$, and $\{p_1, p_2, p_6\}$, by Pruning Rule 3.

Definition 8 (Lower Bound Aggregate Value of Partial k-combination) *Given a partial k-combination $\hat{P}$ with its fixed path set $\hat{P}_f$ and variable path set*

$\hat{P}_v$, $\Psi_{lb}(\hat{P})$ *is defined as*

$$\sum_{j=1}^{m} \min\{\min_{p \in \hat{P}_f} \tau_j(p), \min_{p_* \in \hat{P}_v} \tau_j(p_*)\} \tag{4.2}$$

With the above lower bound cost equation, we have the following pruning rule.

PRUNING RULE 3 (PARTIAL k-COMBINATION PRUNING) *Let P_{best} be a known k-combination. Given a partial combination $\hat{P}$, if $\Psi_{lb}(\hat{P}) > \Psi(P_{best})$, then the derived combinations of $\hat{P}$ can be pruned.*

4.1.2.1 Efficient computation of $\Psi_{lb}(\hat{P})$ by $\mathcal{TC}$ matrix

It is expensive to compute $\Psi_{lb}(\hat{P})$ by Definition 8 directly because the term $\min_{p_* \in \hat{P}_v} \tau_j(p_*)$ takes $O(|\hat{P}_v|)$ computation time. We propose a structure called $\mathcal{TC}$ matrix to support retrieving term $\min_{p_* \in \hat{P}_v} \tau_j(p_*)$ in constant time.

The $\mathcal{TC}$ matrix is a $n \times m$ matrix, where m is the number of time instants and n is the size of $\mathcal{C}$. Its entries are used to store the lower bound travel times of some variable path sets, i.e., $\min_{p_* \in \hat{P}_v} \tau_j(p_*)$ in Equation 4.2. Let $\mathcal{C} = \{c_1, c_2, ..., c_n\}$ and the entry $\mathcal{TC}[i, j]$ is defined as the lower bound travel time of $c_i, ..., c_n$ at the j-th time instant:

$$\mathcal{TC}[i, j] = \min_{x=i}^{n} \tau_j(c_x) \tag{4.3}$$

Table 4.2 illustrates the $\mathcal{TC}$ matrix for the paths in Table 1.2. Intuitively, the i-th row of $\mathcal{TC}$, denoted by $\mathcal{TC}[i, \cdot]$, is the lower bound cost vector of the candidates $c_i, ..., c_n$ and supports our partial k-combination pruning. By con-

structing the $\mathcal{TC}$ matrix incrementally in descending order of i, we can achieve $O(n \cdot m)$ construction time.

Next, by using $\mathcal{TC}$ matrix in Table 4.2, we illustrate how to compute the lower bound cost of a partial combination $\hat{P}$. Suppose $k = 3$ and consider $\hat{P}$ with $\hat{P}_f = \{p_4, p_5\}$ and $\hat{P}_v = \{p_6, p_2, p_1\} = \{c_4, c_5, c_6\}$. To calculate $\Psi_{lb}(\hat{P})$, we first fetch the cost vectors of p_4, p_5 from Table 1.2, which are $\langle 19, 16, 20, 21, 8\rangle$ and $\langle 17, 30, 23, 21, 9\rangle$. Next, we fetch the lower bound cost vector $\langle 15, 20, 12, 14, 11\rangle$ of $\hat{P}_v$, i.e., $\mathcal{TC}[4, \cdot]$, from Table 4.2. Note that we need to compute this lower bound cost vector from scratch if we do not maintain $\mathcal{TC}$ matrix. Then, we combine these three cost vectors by taking the minimum value in each dimension to obtain $\langle 15, 16, 12, 14, 8\rangle$, and calculate the lower bound cost as: $\Psi_{lb}(\hat{P}) = (15 + 16 + 12 + 14 + 8) = 65$. If $P_{best} = \{p_2, p_3, p_4\}$ with $\Psi(P_{best}) = 54$, we do not examine all 3-combinations derived from $\hat{P}$ (i.e., $\{p_4, p_5, p_6\}$, $\{p_4, p_5, p_2\}$ and $\{p_4, p_5, p_1\}$) since $\Psi(P_{best}) < \Psi_{lb}(\hat{P})$.

Table 4.2. $\mathcal{TC}$ matrix for paths in Table 1.2, ordered by $\tau_{min}(p)$

Content of $\mathcal{C}$	**Path**	τ_1	τ_2	τ_3	τ_4	τ_5
c_1	p_3	15	10	6	14	8
c_2	p_4	15	16	12	14	8
c_3	p_5	15	20	12	14	9
c_4	p_6	15	20	12	14	11
c_5	p_2	18	20	14	14	12
c_6	p_1	19	20	14	15	16

4.1.2.2 Enumeration Algorithm

We adopt *branch-and-bound* paradigm together with partial k-combination pruning to enumerate the optimal solutions efficiently. Algorithm 3 shows

the pseudocode for enumerating path combinations and finding the optimal k-combination P_{opt} with the aid of $\mathcal{TC}$ matrix. First, we compute the $\mathcal{TC}$ matrix of $\mathcal{C}$. Next, we execute *RecurBranch* recursively to insert the i-th candidate c_i into $\hat{P}$. If $\hat{P}$ contains fewer than k paths, then we compare its lower bound cost ($\Psi_{lb}(\hat{P})$) with that of the best combination found so far (stored in P_{opt}). If $\hat{P}$ has a smaller cost, then we call *RecurBranch* to further expand it. When $\hat{P}$ contains k paths, we can compute its exact cost and check whether it is better than the current P_{opt}.

The effectiveness of partial combination pruning depends on how *early* we can obtain a complete k-combination with a high pruning power, i.e., close to the optimal solution. In order to achieve early discovery, we insert the candidates to $\hat{P}$ in ascending order of $\tau_{min}(p) = \min_{j=1}^{m} \tau_j(p)$ as shown in Table 4.2 and Algorithm 3. This order intuitively allows early discovery of a good k-combination.

THEOREM 4 *The time and space complexities of Algorithm 3 are* $O(m \cdot \binom{|\mathcal{C}|}{k})$ *and* $O(|\mathcal{C}| \cdot m)$ *respectively.*

Proof. At the worst case, if the pruning rules do not work, Algorithm 3 enumerates all $\binom{|C|}{k}$ combinations to compute the optimal solution and takes $O(m)$ time to compute Ψ. Therefore, it takes $O(m \cdot \binom{|\mathcal{C}|}{k})$ time. We need to maintain the cost vectors of the candidates (e.g., Table 1.2) and their corresponding $\mathcal{TC}$ matrix. Both of them require $O(|\mathcal{C}| \cdot m)$ space and hence Algorithm 3 requires $O(|\mathcal{C}| \cdot m)$ space also. □

Algorithm 3 BranchEnumerate *implements* FindOptimal

Algorithm BranchEnumerate (Integer k, Paths $\mathcal{C}$)
1: Initialize $P_{opt} \leftarrow \varnothing$ ▷ optimal solution in P_{opt}
2: $\mathcal{C}^s \leftarrow$ sort $\mathcal{C}$ in ascending order of τ_{min}
3: compute the matrix $\mathcal{TC}$ by using $\mathcal{C}^s$
4: **for** $i \leftarrow 1$ to $|\mathcal{C}^s| - k + 1$ **do**
5: $\quad \hat{P} \leftarrow \{c_i^s\}$
6: $\quad$ RecurBranch (k, $\hat{P}$, $\mathcal{C}^s$, $\mathcal{TC}$, P_{opt})
7: **return** P_{opt}

Algorithm RecurBranch (Integer k, Path set $\hat{P}$, Path set $\mathcal{C}^s$, Matrix $\mathcal{TC}$, k-combination P_{opt})
1: **if** $|\hat{P}| < k$ **then**
2: $\quad z \leftarrow$ largest index of candidates in $\hat{P}$
3: $\quad$ **for** $i \leftarrow z + 1$ to $|\mathcal{C}^s| - (k - |\hat{P}|) + 1$ **do**
4: $\quad\quad \hat{P} \leftarrow \hat{P} \cup \{c_i^s\}$
5: $\quad\quad$ **if** $\Psi_{lb}(\hat{P}) < \Psi(P_{opt})$ **then** ▷ by using $\mathcal{TC}$
6: $\quad\quad\quad$ RecurBranch (k, $\hat{P}$, $\mathcal{C}^s$, $\mathcal{TC}$, P_{opt})
7: $\quad\quad \hat{P} \leftarrow \hat{P} \setminus \{c_i^s\}$
8: **else**
9: $\quad P \leftarrow \hat{P}$ ▷ becomes a k-combination
10: $\quad$ **if** $\Psi(P) < \Psi(P_{opt})$ **then**
11: $\quad\quad P_{opt} \leftarrow P$

4.2 Heuristic Methods

We have shown the hardness of our problem and the number of candidates can be exponential in $|V|$. It is tempting to devise heuristics which can bound the number of candidates $|\mathcal{C}|$. In this section, we present two heuristic methods to meet this requirement and find a low-cost solution efficiently. First, we propose a method (TP) that reduces the cost of candidate generation by a heuristic (Phase I). Second, we develop a method (ATP) that makes 'best-effort' to find a low-cost solution within a given time limit.

4.2.1 Top-Picker Algorithm

Observe that, in our exact algorithm, the candidate set $\mathcal{C}$ is large even if we apply pruning rules in Phase I. Thus, this would lead to a huge number of path combinations in Phase II. In order to reduce the total computation cost, Top-Picker Algorithm (TP) generates a bounded number of candidates by a heuristic (Phase I) and reuses the combination enumeration of the exact method (Phase II).

The idea of TP is to limit the size of the candidate set $\mathcal{C}$, say, to at most m. It computes the shortest path sp_j (from v_s to v_t) at each time instant, and then inserts these paths into the candidate set $\mathcal{C}$. Note that TP always returns the optimal solution when $k \geq m$.

Let us use Table 1.2 as an example with $k = 3$. First, we find the shortest paths at each time instant. They are p_6, p_3, p_3, p_2, and p_4 respectively. Then, we insert them into the candidate set $\mathcal{C} = \{p_2, p_3, p_4, p_6\}$. In Phase II, we apply Algorithm 3 to enumerate all 3-combinations of $\mathcal{C}$, such as $\{p_2, p_3, p_4\}, \{p_2, p_3, p_6\}, \{p_2, p_4, p_6\}, \{p_3, p_4, p_6\}$, etc. Finally, we compute their costs and return $P_{opt} = \{p_2, p_3, p_4\}$.

THEOREM 5 *The time complexity of Top-Picker Algorithm is* $O(m \cdot (D + \binom{m}{k}))$ *where* D *is the time complexity of the shortest path algorithm. Its space complexity is* $O(\max\{|V| + |E|, m^2\})$.

Proof. In Phase I, TP executes shortest path search m times to find the candidate set $\mathcal{C}$, resulting in time complexity $O(m \cdot D)$, where D is the time complexity of the shortest path algorithm. Since the size of $\mathcal{C}$ is at most m, Phase II may

examine at most $\binom{m}{k}$ path combinations and take $O(m)$ to derive Ψ. As a result, the time complexity of Top-Picker Algorithm is $O(m \cdot (D + \binom{m}{k}))$.

We need to maintain the road network for shortest path search in Phase I, which requires $O(|V| + |E|)$ space. In Phase II, we can solely maintain the cost vectors of all candidates and $\mathcal{TC}$ matrix, which requires $O(m^2)$ space. Therefore, its space complexity is $O(\max\{|V| + |E|, m^2\})$. □

Algorithm 4 GeneratePaths-TP (Node v_s, Node v_t)

```
1: Initialize C ← ∅
2: for j ← 1 to m do
3:     sp_j ← the shortest path from v_s to v_t at instant j
4:     if sp_j is not in C then
5:         C ← C ∪ {sp_j}
6: return C
```

4.2.2 Anytime Top-Picker Algorithm

In some applications, the query user (e.g., transportation planner) is fine with approximate solution and specifies a time limit T_{limit} for finding the solution. To support this requirement, we propose the Anytime Top-Picker Algorithm (ATP), which attempts to find a good approximate solution for the TTP problem within time limit.

In order to allocate the time fairly on (i) finding candidates and (ii) enumerating path combinations, we interleave both phases I and II in this algorithm. For this purpose, we implement an incremental function for Phase I, called GetNextSP(), as shown in Algorithm 5. It iteratively returns distinct shortest paths across the given m time instants until exhausting all of them. Algorithm 6 shows

the pseudocode of ATP. Initially, ATP fetches k distinct shortest paths over m instants and stores them into a candidate set $\mathcal{C}$. These k paths are used to initialize the best combination found so far P_{opt}. If there is time left, then it retrieves another shortest path sp. Subsequently, a new k-combination P is formed by combining sp with each $(k-1)$-combinations of $\mathcal{C}$, and replaces the current optimal solution if it has a better score. Upon reaching the time limit or exhausting all of the shortest paths, the algorithm returns the best combination found so far P_{opt} as the result.

Algorithm 5 GetNextSP(Node v_s, Node v_t, Hash Table $\mathcal{H}$)

```
1: for each unscanned instant j do
2:     sp ← the shortest path from v_s to v_t at instant j
3:     if sp is not in H then
4:         Insert sp to H
5:         return sp
6: return ∅
```

Algorithm 6 AnytimeTP (Node v_s, Node v_t, Integer k, Time T_{limit})

```
1: C ← ∅, P_opt ← ∅
2: Initialize a hash table H
3: for i ← 1 to k do
4:     sp ← GetNextSP ( v_s, v_t, H )
5:     C ← C ∪ {sp}
6: P_opt ← C
7: while sp ← GetNextSP ( v_s, v_t, H ) do
8:     for each (k − 1)-combination P^{k−1} of C do
9:         if T_limit is used up then
10:            return P_opt
11:        P_cur ← P^{k−1} ∪ {sp}
12:        if Ψ(P_cur) < Ψ(P_opt) then
13:            P_opt ← P_cur
14:    C ← C ∪ {sp}
15: return P_opt
```

We illustrate how ATP works on the example in Table 1.2. Assume $k = 3$ again. Table 4.3 shows the detailed execution steps of ATP. First, we fetch $k = 3$ paths (i.e., p_6, p_3, p_2) and insert them into the candidate set $\mathcal{C}$. Next, we initialize $P_{opt} = \mathcal{C} = \{p_6, p_3, p_2\}$, whose score is $\Psi(P_{opt}) = 55$. If there is time left, then we fetch the next distinct shortest path (p_4). Then, we enumerate all 2-combinations of $\mathcal{C}$, such as $\{p_6, p_3\}$, $\{p_3, p_2\}$ and $\{p_6, p_2\}$, and combine each of them with p_4 to form a new 3-combination. If a new combination has a smaller score, then it becomes the current optimal result. Finally, the algorithm reports $\{p_2, p_3, p_4\}$ with $\Psi = 54$ as the answer.

Table 4.3. Execution steps of ATP on Table 1.2

Procedure	Checked $P^3_{1,7}$	$\Psi(P^3_{1,7})$	Current P_{opt}
Initialize	$\{p_6, p_3, p_2\}$	55	$\{p_6, p_3, p_2\}$
GetNextSP : p_4	$\{p_4\} \cup \{p_6, p_3\}$	55	$\{p_6, p_3, p_2\}$
	$\{p_4\} \cup \{p_6, p_2\}$	65	$\{p_6, p_3, p_2\}$
	$\{p_4\} \cup \{p_3, p_2\}$	**54**	$\{p_2, p_3, p_4\}$

4.3 Discussion: TTPs in Practice

In the previous sections, we presented an exact algorithm (Section 4.1) and two heuristics (Section 4.2) to solve the TTP problem. However, in order to apply traffic-tolerant paths in practice, we need to further consider two issues, namely *how to select a suitable set of historic traffic data* (Section 4.3.1) and *how to update TTP in online route services* (Section 4.3.2).

4.3.1 Selection Policy on Historic Traffic Data

Up to the previous section, we implicitly assume that we are given a set of traffic data to compute TTPs. In practice, the set of traffic data may be extracted from a large traffic data repository. For example, users (e.g., transportation planners and analysts) are interested in only a small portion of the data (e.g., the most recent) since the volume of entire traffic data is huge.

Therefore, in this section, we discuss some guidelines for selecting a subset of traffic data and candidates computation. We call the set of guidelines a *selection policy* which is suited to the *application scenarios* we discussed in Section 1.1.1. It contains two parameters, namely the number of days (D) and the number of time instants per day (L). The following discussion on selection policy is in the context of *online route services*. But the selection policy for transportation analysis can be defined similarly.

Intuitively, D determines how many historic traffic records are used for pre-computation. Including all traffic recorded several years ago from the time of recurrent route queries may not be useful since the traffic patterns may change over time. In contrast, only considering the traffic data collected few weeks or months ago from the time of the queries may be more meaningful as they reflect the recent traffic of the road network.

Besides, the collection period of historic traffic data is equally important. If the journey starts in the morning, it is not reasonable to include the traffic data recorded at midnight since the traffic conditions during these two periods are expected to be different. Additionally, we expect that there is a periodicity in traffic and the traffic at the same 'hours' on different dates has similar patterns.

For example, the traffic between 8:00am and 9:00am on weekdays in a month is expected to be similar since the majority of citizens go for work in these peak hours and the traffic in this period may be useful to a route which starts, say, at 8:00am. We consider this daily time window for path pre-computation as another selection parameter (L) - the number of time instants per day.

To sum up, the selection policy determines a subset of historic traffic data for pre-computation. It is decided by the system administrators or transportation analysts who have knowledge on the road network and traffic of their cities. We are going to evaluate the effect of the two parameters on the accuracy of TTPs in our experiments (Chapter 5). However, the art of designing and customizing an appropriate selection policy is orthogonal to our approach.

4.3.2 Updating TTPs in Online Route Services

In the context of online route services, TTPs act as candidate paths of candidate approach (cf. Section 2.2.3). We recommend to update TTPs offline periodically (e.g., weekly) in order to reflect the recent traffic patterns of road networks. The update frequency of can be determined by systems administrators.

For the online phase, we propose the following adaptation to cope with situations like sudden changes in traffic conditions (e.g., traffic accidents, protests, and congestions). First, we check whether there are any segments on TTPs suffering serious congestions. Then, we replace those congested segments with other segments offering shorter travel times.

For instance, we have a traffic-tolerant path $p = \langle v_s, ..., v_u, ..., v_w, ..., v_t \rangle$ and find that the segment $s_{old} = \langle v_u, ..., v_w \rangle$ is undergoing a traffic jam. We search

the fastest path between v_u and v_w (sp) based on the current traffic and replace s_{old} with sp. Since v_u and v_w are close to each other, the cost of fastest path search is small. This adaption not only preserves the quality of traffic-tolerant paths but also maintains reasonable query time.

Chapter 5

Experiments

In this chapter, we conduct various experiments to evaluate our proposed algorithms by using real and synthetic data sets. We first present the experimental setup and then discuss the experimental results.

5.1 Road Network and Traffic Data

Table 5.1 lists the information of road networks used in our experiments. They are United Kingdom (**UK**) and Colorado (**COL**), which are available at [3] and [1] respectively.

For UK, we downloaded real and historic traffic data from [5] from January to March of 2013. Each traffic record is linked to each road segment in UK based on the unique identifiers of roads. The traffic data were recorded every 15 minutes and hence there are 96 traffic records per day for each road segment.

Since the historic traffic data of COL are *not* available, we synthetically generate traffic data in the following way. We generate the travel times for m time instants. At each time instant j, we pick a random number from $\{-1, 1\}$ as $sign_j$ to determine if the travel times of all edges *increase* or *decrease*. Then, for each edge, we select a random number from $[0..X]$ and $w_j(e)$ is calculated as $\frac{l(e)}{s} \times (1 + sign_j \cdot X\%)$, where $l(e)$ is the road segment length and s is the vehicle speed. We set s as 60km/h in our experiments. The values of X are shown in Table 5.2.

5.2 Experimental setup

In the introduction, we suggest two applications of our TTP problem, namely transportation planning and online route services. The former requires fast computation time, whereas the latter focuses on the travel time error of the precomputed paths against real-time traffic. Thus, in our experiments, we measure the computation time and travel time error.

Training and testing sets of traffic data: In order to evaluate the travel time error of the methods, we divide the traffic data into *training* and *testing* sets. We take the training set for computing TTP solutions, and take the testing set for testing the travel time error of such solutions. The data on testing sets were collected after those in the training sets. For UK, training and testing sets of traffic data are governed by the equation $m = D \times L$, where D is the number of training (testing) days and L is the number of training (testing) time instants per day. L specifies the granularity of time periods, (e.g., in UK, a time period 08:00-09:00 implies $L = 4$ since the traffic data of UK are recorded every 15

minutes).

In our experiments, we vary D and L of training sets. As for testing sets, we fix D of the testing set as traffic data collected during 16 - 31 March, 2013 (i.e., $D = 16$) but directly use L of the training sets. For instance, we take the traffic data between 08:00-08:30 ($L = 2$) and 15 days before the testing period (i.e., 1 - 15 March, 2013) for training. Then, we use the traffic data collected between 08:00-08:30 during 16 - 31 March, 2013 for testing.

For COL, we simply generate two sets of synthetic traffic data according to Section 5.1. One set is for training while another set is for testing. The number of time instants (m) for training and testing is shown in Table 5.2.

Travel time error measures: For each method, we use the training data to compute its resulting path combination P. Then, we measure the travel time error of P by using the testing data. Specifically, we apply Equation 1.1, substitute P into $P^k_{s,t}$, and calculate $\tau_j(p)$ and $\tau_j(sp_j)$ based on time instants j in the testing data.

Query: For generation of source-destination pairs, we pick 100 pairs uniformly at random from the road network. The *average* time errors of these pairs are reported in the figures in the subsequent sections. For the value of k, we test k from 1 to 10 and choose $k = 5$ by default, which is the same as [29].

Methods: Our proposed methods are: the exact method (which is shown as TTP in figures) and two heuristic methods (TP and ATP). Our competitors are two representative heuristic methods proposed in [29]: *K-variance* (K-VAR) and *Y-moderate* (Y-MOD). The additional parameters used by them are configured according to [29]. Their details have been discussed in Chapter 2.

We implemented all methods in C++ and evaluate them in subsequent experiments. All the experiments were run on a PC with 3.4 GHz Intel® Core™ i7 CPU and 8 GB RAM in Linux environment.

Table 5.1. Road network size

Network	#Nodes	#Edges	Traffic
United Kingdom (**UK**)	2,321	4,996	Real [5]
Colorado (**COL**)	435,666	1,042,400	Synthetic

Table 5.2. Experiment parameters

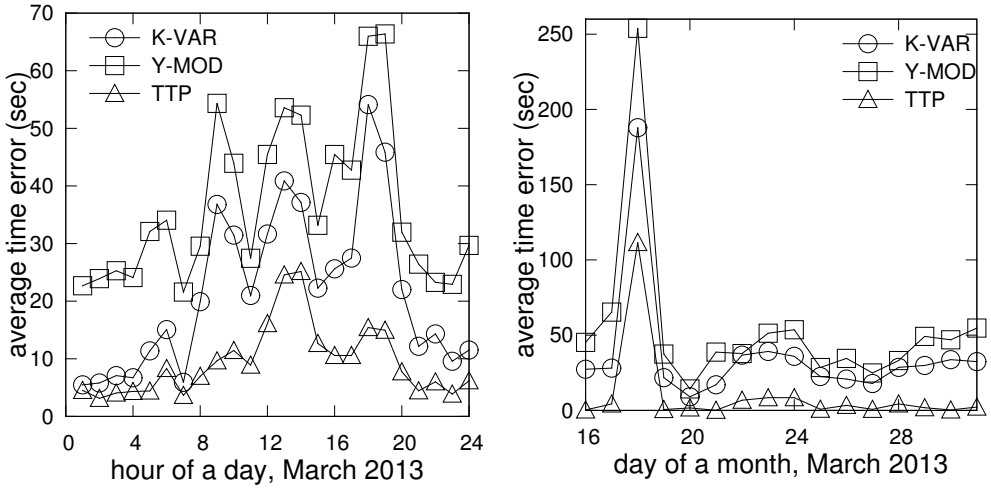
Parameters	Values	Default
k	1,2,...,10	5
No. of days (D)	15,30,45,60	15
No. of time instants per day (L)	1,2,3,4	1
m for synthetic traffic	15,30,45,60	30
X% for synthetic traffic	5,10,15,20	10

5.3 Real Traffic Data

In this section, we present the travel time error and the efficiency of our proposed methods in various perspectives including (i) different hours of a day, (ii) different days of a month, (iii) increasing the number of paths (k), and (iv) increasing the number of time instants (m). In this section, TTP in the figures refers to the results generated by the exact algorithm.

Different hours of a day. First, we evaluate how the average time error varies with different time of a day since this reveals some traffic patterns of UK. Figure 5.1(a) shows the average time errors of TTP and two competitors at different hours of a day. TTP achieves a smaller time error throughout the day and outperforms the others especially in rush hours (i.e., 08:00 and 17:00) by at least 3 times. Although there is a spike between 13:00 and 14:00 for TTP, its time

error is still smaller than that of others. The figure also reveals that the traffic of UK fluctuates in three periods, namely 08:00-09:00, 13:00-14:00, and 17:00-18:00. The default time period in the subsequent experiments is 08:00-09:00.

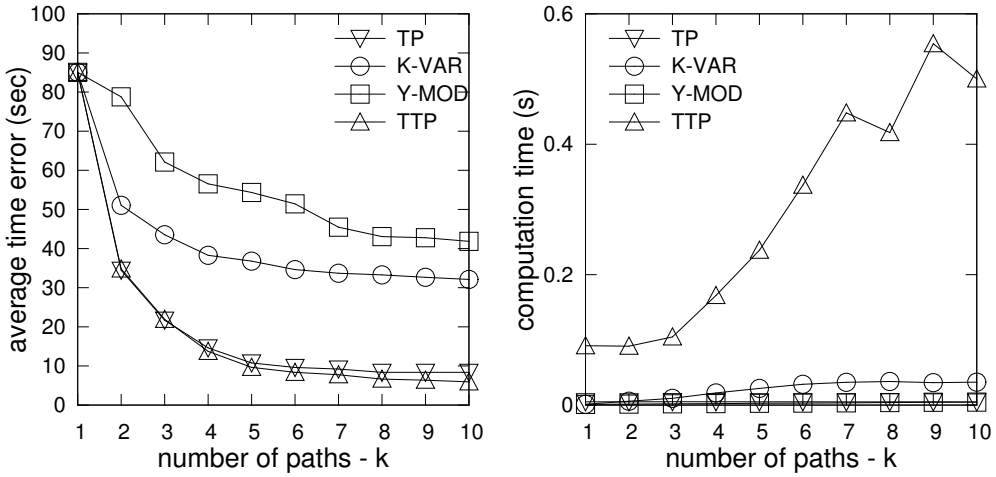


(a) varying time of a day (b) varying day of a month

Figure 5.1. $k = 5$**, March 2013, UK**

Different days of a month: Figure 5.1(b) shows the average time errors of three algorithms across different testing days of March, between 08:00 and 09:00. It can be seen that TTP consistently has smaller errors than the others. Although all of them suffer from a sudden rise in time error on 18 March, 2013, TTP can still obtain a lower average time error of about 2 minutes while K-VAR and Y-MOD have average time errors of about 3 minutes and 4 minutes respectively.

Varying k: As shown in Figure 5.2(a), the average time errors of all algorithms diminish with increasing k and start to converge when $k > 5$. This is because including more paths implies it is more probable to have a path, out of k paths, with a smaller travel time error at a particular time instant. Obviously,

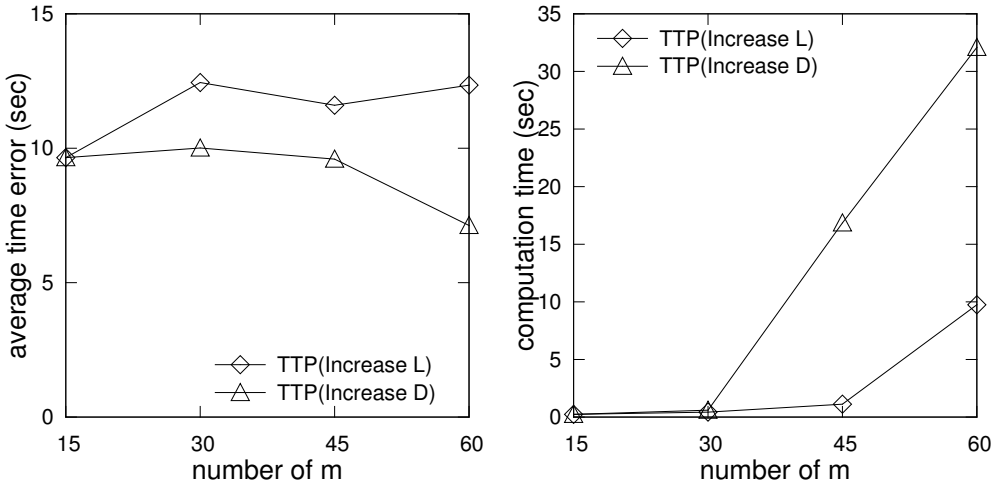


(a) error (b) computation time

Figure 5.2. Varying k, 08:00-08:15 in March 2013, UK

the average time error of TTP decreases more rapidly than the others, indicating a higher marginal decrease in travel time error. We also measure the computation time of TTP as shown in Figure 5.2(b). It increases with k because the number of k-combinations enumerated also increases. Although TTP can finish within a second in UK network, it cannot scale well in medium-sized networks (cf. Figure 5.6).

Varying D: In this experiment, we set the number of time instants per day to one ($L = 1$) but increase the number of training days (D) from 15 to 60 days in order to examine its effect on TTP. The time error and computation time of TTP are shown in Figure 5.3. In general, the time error of TTP drops with increasing D, implying that more training days of a time instant is more preferable, but the trade-off is the drastic increase in computation cost. The reason is that dominance loses selectivity with increasing number of dimensions.



(a) average time error (b) computation time

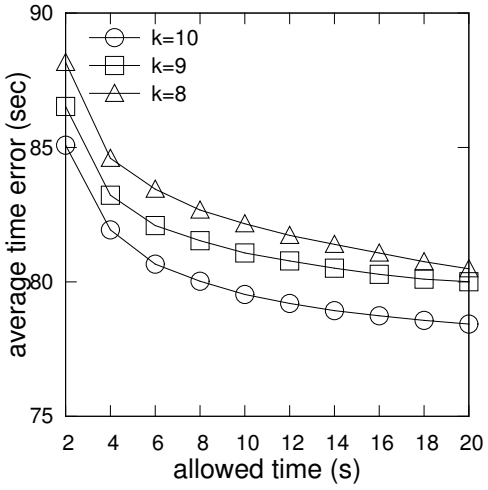
Figure 5.3. Varying m, fix $k = 5$, UK

Varying L: Conversely, we fix the number of training day to 15 days but vary the number of time instants per day (L) from 1 to 4. The same measurements are made and also presented in Figure 5.3. The time error of TTP rises when including more time instants. This may be because more time instants introduce higher traffic variability. Same as before, the computation cost of TTP increases proportionally to the number of dimensions. To sum up, one time instant per day (e.g., 08:00 - 08:15) with 15 to 30 training days may be a feasible choice for TTP in UK road network.

5.4 Synthetic Traffic Data

Since the exact solution is not scalable in large road networks, we proposed two heuristics - TP and ATP algorithms. In this section, we evaluate mainly the error and efficiency of our proposed heuristics by varying the following parameters

- (i) the percentage change of traffic time ($X\%$), (ii) the number of paths (k), and (iii) the number of time instants (m).

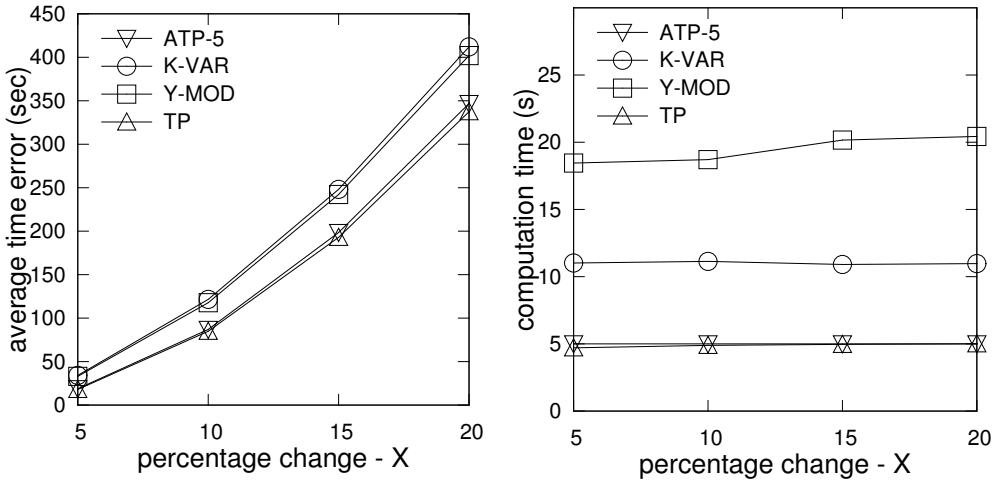


average time error

Figure 5.4. Varying allowed time of ATP, fix $X\% = 10$ and $m = 30$, COL

Varying allowed time of ATP: Before showing the results of varying the mentioned parameters, we first show that the average time error of ATP drops initially and converges as allowed running time increases. This finding justifies the reason of proposing ATP. We only show the case of $k = 8, 9, 10$ because the computation times of TP in these cases are more than 30 seconds according to Figure 5.4. In the following discussions, we set the default time limit of ATP to 5 seconds and this is denoted by ATP-5 in figures.

Varying X: In this experiment, we increase the traffic fluctuation and measure the changes of average time errors. The results are shown in Figure 5.5. It can be seen that the average time errors of all methods rise with increasing the percentage change of travel times. Nevertheless, the errors of TP and ATP are still smaller than the others, implying that they are more resistant to the

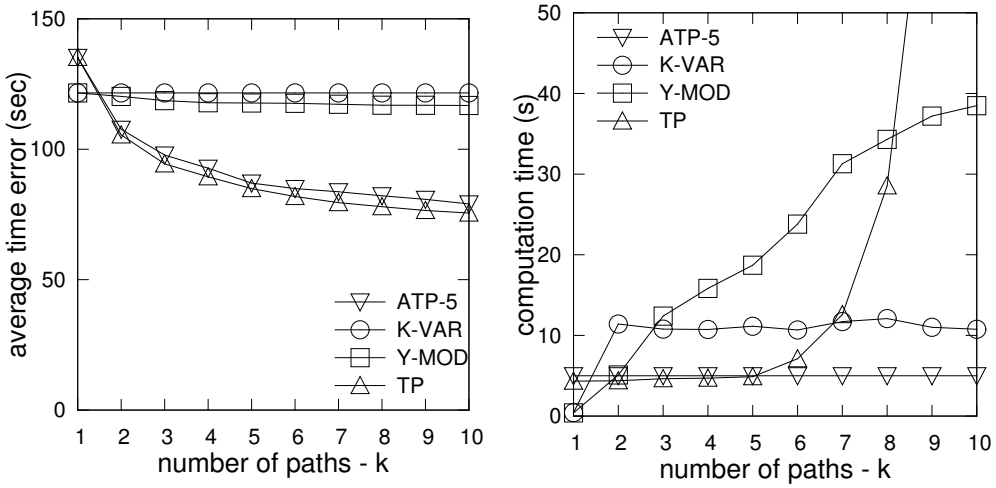


(a) average time error (b) computation time

Figure 5.5. Varying $X\%$, fix $k = 5$ and $m = 30$, COL

traffic fluctuation. We also show their computation times, which are intuitively insensitive to the traffic changes, for reference.

Varying k: Figure 5.6 shows the average time errors and computation times of all methods, and reveals their important properties. The computation cost of K-VAR becomes steady after k is larger than 1 because it is controlled by the number of iterations, which is fixed throughout the experiment and configured according to [29]. Its time error remains unchanged with k because it can retrieve only one path throughout the iterations after our investigation. The computation time of Y-MOD increases linearly with k since it is a variant of Yen's K shortest path algorithm [36] whose complexity is linearly dependent on k. However, its time error drops slightly with increasing k. This may be because of the inherent property of Yen's algorithm, which generates a set of K paths with a large portion of overlapping. As for TP, its computation time rises rapidly when k is larger than 5. The running times for $k = 9$ and 10 are 75 and

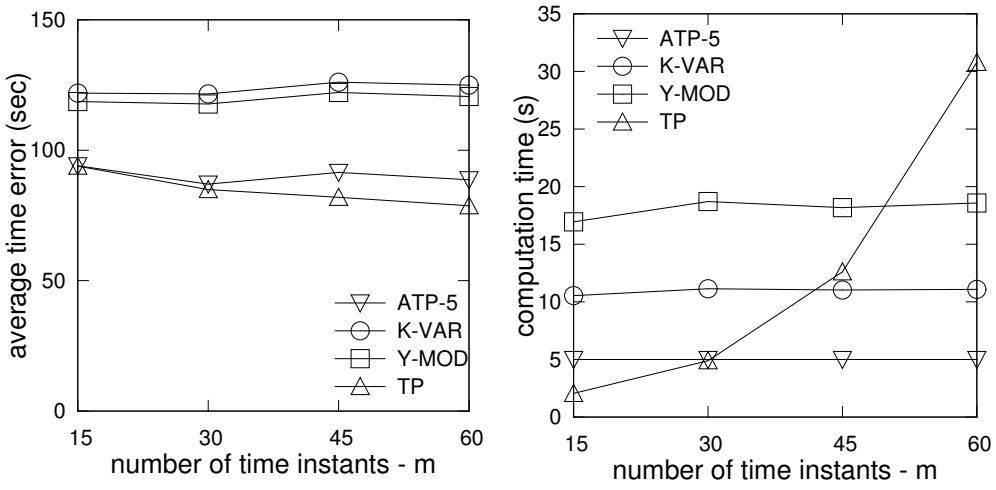


(a) average time error (b) computation time

Figure 5.6. Varying k, fix $X\% = 10$ and $m = 30$, COL

148 seconds respectively. The rise in computation cost is due to the exponential number of enumerated combinations. The exponential growth in computation time of TP also implies that the computation time of the exact algorithm (TTP) has the similar and even more rapid growth. This further justifies the our proposed heuristics. Despite this, the error of TP diminishes with k continuously and outperforms the others. ATP exhibits a similar diminishing trend to TP but it offers constant computation cost. To summarise, ATP achieves better trade-off between error and computation cost.

Varying m: Figure 5.7 shows the average time errors and computation times of the methods versus the number of time instants. As m increases, the average time errors and computation times of Y-MOD and K-VAR are insensitive to m since they simply aggregate (i.e., average) the travel times of all time instants to a single travel time for candidates generation. When m increases, the average time error of TP drops slightly while its computation cost increases



(a) average time error (b) computation time

Figure 5.7. Varying m, fix $X\% = 10$ and $k = 5$, COL

exponentially due to enumeration of combinations. As for ATP, its average time error decreases initially but increases afterwards. This is because it cannot further explore more paths at other time instants due to a given time limit, i.e., 5 seconds.

Chapter 6

Conclusion and Future Work

In this chapter, we conclude the work in this thesis and discuss the future direction of the work.

6.1 Conclusion

This thesis proposes and studies a novel problem called the traffic-tolerant path (TTP) problem in road networks, which takes a SD pair and historic traffic information as input and returns k paths $P_{s,t}^k$ that minimize the aggregate historic travel time $\Psi(P_{s,t}^k)$. Its applications include transportation reliability analysis and efficient route-search services. By reducing a well-known NP-hard problem, Set-Cover problem, to the TTP problem, we prove the NP-hardness of this problem.

Then, we propose a two-phase exact enumeration algorithm. In Phase I, given a SD pair (v_s, v_t), we regard all possible paths from v_s to v_t as a high di-

mensional vector and leverage dominance property to prune unpromising paths, resulting in a set of candidate paths $\mathcal{C}$. In Phase II, in order to reduce search space, we adopt a branch-and-bound algorithm to enumerate the optimal k-combination $P^k_{s,t}$. In the view of the NP-hardness, we also devise two heuristics (TP and ATP) which offers "low-cost" solutions efficiently. Finally, the experiments conducted on real and synthetic data show that our proposed algorithms achieve much higher accuracy than existing approaches and their efficiency is comparable to that of competitors.

6.2 Future Work

The TTP problem is a combinatorial optimization problem and there are some intriguing future directions on this problem. First, it is interesting to devise an algorithm with provable guarantees for this problem in our future work. Although we propose TP heuristics due to its hardness, it gives the optimal solution only when $k \geq m$ and it does not provide any approximation guarantees in the worst case.

Second, it is tempting to extend the TTP problem to *time-dependent spatial networks* since Demiryurek et al. [15] showed that time-dependent shortest paths offer shorter travel times (about 36%) than static shortest paths which considers constant travel times of edges, demonstrating the applicability of time-dependent paths.

In time-dependent spatial networks, the travel time of an edge e is modeled as a function of arrival time to the edge. By extending the TTP problem to

time-dependent spatial networks, we associate each edge with m *time-varying functions* instead of a m-dimensional travel time vector. Those time-varying functions can represent the daily variation of travel times like Figure 1.3 but also can denote traffic patterns in different scenarios such as weekday, holiday, traffic under blizzards, and so on.

As a result, the *time-dependent* TTP problem can be formulated as follows.

Definition 9 (Time-dependent Road Network with Historic Traffic) *A road network is modeled as a directed and weighted graph $G(V, E, T_m)$, where V is the set of road junctions, E is the set of road segments, and T_m is a mapping from edges to m time-varying functions $\mathcal{T}_1, \mathcal{T}_2, ..., \mathcal{T}_m$.*

Given an edge $e \in E$ and time t, we denote the travel time of e starting at time t with respect to j-th time-varying function by $\mathcal{T}_j(e, t)$.

Given a path $p = \langle v_{a_1}, v_{a_2}, ..., v_{a_n} \rangle$ and start time t_s, the time-dependent travel time of p for j-th time-varying function, denoted by $\mathcal{T}_j(p, t_s)$, is defined as

$$\mathcal{T}_j(p, t_s) = \sum_{i=1}^{n-1} \mathcal{T}_j((v_{a_i}, v_{a_{i+1}}), t_i) \tag{6.1}$$

where $t_1 = t_s, t_{i+1} = t_i + \mathcal{T}_j((v_{a_i}, v_{a_{i+1}}), t_i)$ for $i = 1, ..., n-2$.

Definition 10 (Time-dependent K Traffic-tolerant Paths Query) *Given a road network $G(V, E, T_m)$, $v_s, v_t \in V$, a positive integer k and a start time t_s, Time-dependent Traffic-tolerant Paths Query TDTTP(v_s, v_t, k, t_s) returns a k-combination $P^k_{s,t}$ such that for any possible $P'^k_{s,t}$, $\Psi(P^k_{s,t}, t_s) \leq \Psi(P'^k_{s,t}, t_s)$,*

where Ψ *of any* $P^k_{s,t}$ *is defined as*

$$\Psi(P^k_{s,t}, t_s) = \sum_{j=1}^{m} \min_{p \in P^k_{s,t}} \mathcal{T}_j(p, t_s) \tag{6.2}$$

Our existing exact algorithm and TP heuristics can be extended to solve the TDTTP problem defined above. The key modification of our exact algorithm (Phase I) is to expand the time-dependent paths from v_s to v_t. Conceptually, we can still make use of the idea of dominance property to prune unpromising paths. Nevertheless, we cannot directly apply prefix path pruning in time-dependent networks since the computation of lower bound travel time of a prefix path, i.e., $LB_j(\hat{p})$, is non-trivial and requires further investigation. For TP heuristics, we can directly apply the state-of-the-art time-dependent shortest path algorithm, e.g., [16], to find the candidates. Finally, it is intriguing to investigate if time-dependent TTPs offers better performance in our future work.

Bibliography

[1] 9th DIMACS Implementation Challenge - Shortest Paths. http://www.dis.uniroma1.it/challenge9/.

[2] Caltrans PeMS. http://pems.dot.ca.gov/.

[3] GB Road Traffic Counts. http://data.gov.uk/dataset/gb-road-traffic-counts/.

[4] Google Maps. http://maps.google.com/.

[5] Highways Agency Network Journey Time and Traffic Flow Data. http://data.gov.uk/dataset/dft-eng-srn-routes-journey-times/.

[6] TomTom - At the Heart of the Journey. http://www.tomtom.com/.

[7] Travel Time Reliability: Making It There On Time, All The Time. url-http://ops.fhwa.dot.gov/publications/tt_reliability/index.htm, 2006.

[8] Ittai Abraham, Daniel Delling, Andrew V. Goldberg, and Renato Fonseca F. Werneck. Hierarchical Hub Labelings for Shortest Paths. In *ESA*, pages 24–35, 2012.

[9] Holger Bast, Stefan Funke, Domagoj Matijevic, Peter Sanders, and Dominik Schultes. In Transit to Constant Time Shortest-path Queries in Road Networks. In *ALENEX*, 2007.

[10] Reinhard Bauer, Daniel Delling, Peter Sanders, Dennis Schieferdecker, Dominik Schultes, and Dorothea Wagner. Combining Hierarchical and Goal-directed Speed-up Techniques for Dijkstra's Algorithm. *ACM Journal of Experimental Algorithmics*, 15, 2010.

[11] Yanyan Chen, Michael G. H. Bell, and Klaus Bogenberger. Reliable pretrip multipath planning and dynamic adaptation for a centralized road navigation system. *IEEE Transactions on Intelligent Transportation Systems*, 8(1):14–20, 2007.

[12] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition.* The MIT Press, 3rd edition, 2009.

[13] Gianlorenzo D'Angelo, Daniele Frigioni, and Camillo Vitale. Dynamic arc-flags in road networks. In *SEA*, pages 88–99, 2011.

[14] Daniel Delling and Dorothea Wagner. Landmark-based Routing in Dynamic Graphs. In *Proceedings of the 6th International Conference on Experimental Algorithms*, WEA'07, pages 52–65, 2007.

[15] Ugur Demiryurek, Farnoush Banaei Kashani, and Cyrus Shahabi. A Case for Time-dependent Shortest Path Computation in Spatial Networks. In *ACM GIS*, pages 474–477, 2010.

[16] Ugur Demiryurek, Farnoush Banaei Kashani, Cyrus Shahabi, and Anand Ranganathan. Online Computation of Fastest Path in Time-dependent Spatial Networks. In *SSTD*, pages 92–111, 2011.

[17] Edsger W. Dijkstra. A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

[18] Robert Geisberger, Peter Sanders, Dominik Schultes, and Daniel Delling. Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks. In *Proceedings of the 7th International Conference on Experimental Algorithms*, WEA'08, pages 319–333, 2008.

[19] Robert Geisberger, Peter Sanders, Dominik Schultes, and Christian Vetter. Exact Routing in Large Road Networks Using Contraction Hierarchies. *Transportation Science*, 46(3):388–404, 2012.

[20] Andrew V. Goldberg and Chris Harrelson. Computing the Shortest Path: A* Search Meets Graph Theory. In *SODA*, pages 156–165, 2005.

[21] Hector Gonzalez, Jiawei Han, Xiaolei Li, Margaret Myslinska, and John Paul Sondag. Adaptive Fastest Path Computation on a Road Network: A Traffic Mining Approach. In *VLDB*, pages 794–805, 2007.

[22] Marta C. González, César A. Hidalgo, and Albert-László Barabási. Understanding individual human mobility patterns. *Nature*, 453(7196):779–782, June 2008.

[23] Ron Gutman. Reach-Based Routing: A New Approach to Shortest Path Algorithms Optimized for Road Networks. In *ALENEX*, pages 100–111, 2004.

[24] Ming Hua and Jian Pei. Probabilistic Path Queries in Road Networks: Traffic Uncertainty Aware Path Selection. In *EDBT*, pages 347–358, 2010.

[25] Evangelos Kanoulas, Yang Du, Tian Xia, and Donghui Zhang. Finding Fastest Paths on a Road Network with Speed Patterns. In *ICDE*, pages 10–10, 2006.

[26] Ekkehard Köhler, Rolf H. Möhring, and Heiko Schilling. Fast Point-to-point Shortest Path Computations with Arc-flags. In *9TH DIMACS IMPLEMENTATION CHALLENGE*, 2006.

[27] Hans-Peter Kriegel, Matthias Renz, and Matthias Schubert. Route Skyline Queries: A Multi-Preference Path Planning Approach. In *ICDE*, pages 261–272, 2010.

[28] Tim Lomax, David Schrank, Shawn Turner, and Richard Margiotta. Selecting travel reliability measures. *Texas Transportation Institute monograph (May 2003)*, 2003.

[29] Nirmesh Malviya, Samuel Madden, and Arnab Bhattacharya. A Continuous Query System for Dynamic Route Planning. In *ICDE*, pages 792–803, 2011.

[30] Kyriakos Mouratidis, Yimin Lin, and Man Lung Yiu. Preference Queries in Large Multi-cost Transportation Networks. In *ICDE*, pages 533–544, 2010.

[31] Peter Sanders and Dominik Schultes. Highway Hierarchies Hasten Exact Shortest Path Queries. In *ESA*, pages 568–579, 2005.

[32] Peter Sanders and Dominik Schultes. Engineering Highway Hierarchies. In *ESA*, pages 804–816, 2006.

[33] Chaoming Song, Zehui Qu, Nicholas Blumm, and Albert-László Barabási. Limits of predictability in human mobility. *Science*, 327(5968):1018–1021, 2010.

[34] Lingkun Wu, Xiaokui Xiao, Dingxiong Deng, Gao Cong, Andy Diwen Zhu, and Shuigeng Zhou. Shortest Path and Distance Queries on Road Networks: An Experimental Evaluation. *PVLDB*, 5(5):406–417, 2012.

[35] Tao Xing and Xuesong Zhou. Reformulation and Solution Algorithms for Absolute and Percentile Robust Shortest Path Problems. *IEEE Transactions on Intelligent Transportation Systems*, 14(2):943–954, 2013.

[36] Jin Y. Yen. Finding the K Shortest Loopless Paths in a Network. *Management Science*, 17(11):712–716, 1971.

[37] Gang Yu and Jian Yang. On the robust shortest path problem. *Computers and Operations Research*, 25:457–468, 1998.

[38] Andy Diwen Zhu, Hui Ma, Xiaokui Xiao, Siqiang Luo, Youze Tang, and Shuigeng Zhou. Shortest Path and Distance Queries on Road Networks: Towards Bridging Theory and Practice. In *SIGMOD*, pages 857–868, 2013.