A Landmark Algorithm for the Time-Dependent Shortest Path Problem

Guidance

Professor Hiroshi NAGAMOCHI
Associate Professor Liang ZHAO

Tatsuya OHSHIMA

2006 Graduate Course

in

Department of Applied Mathematics and Physics

Graduate School of Informatics

Kyoto University

February 2008

Acknowledgments

The author is heartily grateful to Professor Hiroshi Nagamochi for his continual guidance and invaluable suggestions to accomplish the thesis. He commented in detail on the whole work in the thesis, which significantly improved the accuracy of the arguments and the quality of the expression. Without his generous help, none of this work could have been completed. He is deeply grateful to Associate Professor Liang Zhao for his enthusiastic guidance, discussion and persistent encouragement. Many discussions with him were quite exciting and invaluable experience for me. Finally, he thanks all members in Professor Nagamochi's Laboratory including Mr. Hideki Hashimoto for many discussions on the area of this work and their warm friendship during the course.

Abstract

The shortest path problem is one of the most classical problem in combinatorial optimization problem which, given an edge-weighted graph and two vertices, asks to find a path between the two vertices of the minimum length. In this thesis, we consider a generalization of the shortest path problem in which the edge length is time-variable, which we call the time-dependent shortest path problem. This kind of problems has many applications in the fields of navigation systems and others.

Since the first algorithm was proposed by Cooke and Halsey in 1966, many studies have been done for this problem. Currently the fastest algorithm is due to Dreyfus and others (1969–1990) who proposed a straightforward generalization of the famous Dijkstra algorithm that is originally developed for the classical shortest path problem. In this thesis, we give an even faster algorithm at a small amount of extra preprocessing cost. The proposed algorithm is based on the ALT algorithm proposed by Goldberg and Harrelson (2005) for the shortest path problem, in which the main idea is to use pre-calculated landmarks in determining the interim distance labels for vertices. Experimental results show our algorithms is several times faster than the generalized Dijkstra algorithm.

Contents

1 Introduction

1.1 Background and motivation

The *shortest path problem* is a classical combinatorial optimization problem that, given an edge-weighted graph (or directed graph) and two vertices, asks to find a shortest path between the two vertices. There are many applications of this problem and many algorithms have been proposed and used in real situations (see, e.g., [29]). For a network with non-negative edge length, the Dijkstra algorithm [12] is the most well-known algorithm which can be implemented to have the running time of $O(m + n \log n)$ [14], where m and n denotes the number of edges and the number of vertices respectively.

In this thesis, we consider a generalization with time-variable edge length, i.e., the length of each edge is a function of the departure time at the starting vertex. Such a network with variable edge length is called a *time-dependent network*, and the shortest path problem in a time-dependent network is called the *time-dependent shortest path problem* (TDSP). Obviously the traditional shortest path problem is a special case of TDSP with invariable edge length. The follows describes the general TDSP problem (the precise definition will be given in Section 2).

Time-dependent shortest path problem (TDSP)

Input: A time-dependent network, a source s, a destination d and a departure time t.

Output: A shortest (s, d)-path with departure time t.

Problem TDSP arises frequently in our daily life where edge length denotes the traveling time. For example, the traveling time by cars may vary due to traffic jam and the arrival time of trains depends on the time schedules. See Figure 1 for an illustration.

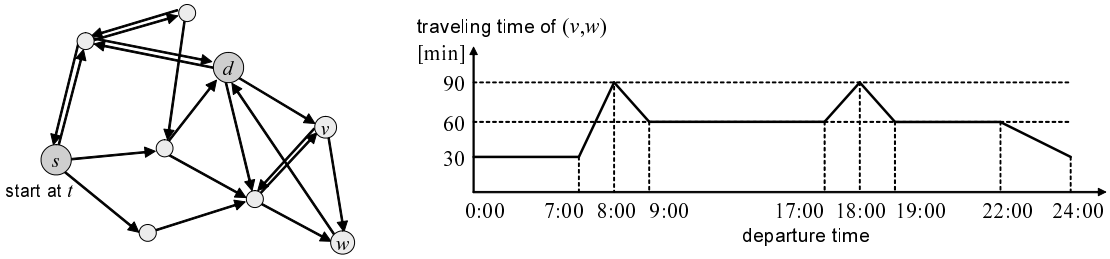


Figure 1: An illustration of TDSP with time-dependent edge length which denotes the traveling time during a day. Notice that there are two peaks representing the rush-hours.

However, comparing to the static case (i.e., the classic shortest path problem), there is little work on the time-dependent case. Cooke and Halsey [6] first considered TDSP and proposed an algorithm based on Dynamic Programming [3]. It can only treat integer time values and is not a polynomial time algorithm. Dreyfus [13] suggested a generalization of the Dijkstra algorithm, but his proposal was not complete until Halpern et al. [20, 25, 33] showed that it requires the network to satisfy the FIFO (First-In First-Out) property, where a network is said to satisfy the FIFO property if the *arrival time* function of each edge is nondecreasing (e.g., edge (v, w) in Figure 1 satisfies the FIFO property. Notice

that the arrival time is the sum of departure time and the corresponding traveling time). Supposing that the edge length can be obtained in $O(1)$ time for any departure time, the generalized Dijkstra algorithm has $O(m + n \log n)$ running time.

However, in many applications a much faster algorithm is required, and for that purpose, *preprocessing* may be used to get and store information that can be used to accelerate the calculation. For instance, in a navigation system, we can store a reasonable number of pre-calculated shortest paths to answer a query about shortest path faster (if hit). In fact, for the traditional shortest path problem, numerous studies on such kind of algorithms have been done in the recent decade, which we will give a survey in the following. Unfortunately, in spite of many trials, no algorithm for TDSP has been proposed before this study. In this thesis, we propose an algorithm that is much faster than the generalized Dijkstra algorithm at a small amount of preprocessing cost. The next table briefly summarizes the algorithms for the shortest path problem.

Table 1: A brief summary of algorithms for the shortest path problem.

Problem type	Algorithm without preprocessing	Algorithm with preprocessing
constant length	$O(m + n \log n)$ [12] etc.	[16] etc. (see Section 1.2)
TDSP	NP-hard [1, 4–6, 40] etc.	open
TDSP + FIFO	$O(m + n \log n)$ [13, 20, 25, 33]	this thesis

1.2 Previous works

We first review some algorithms for the shortest path problem.

The static case

Luby and Ragde [30] suggested *the bidirectional search algorithm* that is based on the Dijkstra algorithm but scans vertices not only in the forward direction from the source vertex but also in the backward direction from the destination vertex. It is reported that is the more practical than the Dijkstra algorithm. For a planar network, the complexity can be further reduced [15, 26].

Goldberg and Harrelson [16] proposed the *ALT algorithm*, which is based on the idea of the A^* algorithm [21] that decides the searching priority of a vertex by the sum of the interim distance (from the source) and an estimated value of the distance to the destination (the Dijkstra algorithm is a special case of the A^* algorithm with 0 estimation value). Goldberg and Harrelson suggested a method to find good estimations at small computation time. They considered to use a vertex subset called the *landmark set* and the idea is to compute an estimation by the triangle inequality of the values of shortest paths (see details in Section 2.2). Later Delling and Wagner [11] applied the ALT algorithm to a dynamic scenario where the topology of the network is supposed to be changeable.

Gutman [19] defined *reach* $r(v)$ for all $v \in V$ as the maximum value of $r(v, P)$ for all shortest paths P with $v \in P$, where $r(v, P)$ is the minimum length between the length from the source vertex of P to v and from v to the end vertex of P. The reach based

approach can prune all vertices which satisfies $r'(v) < h(v)$ from the search space, where $r'(v)$ is an upper bound on $r(v)$ and $h(v)$ is a lower bound on the length of the shortest path from v to the destination vertex. This idea is compatible with ALT algorithm, and Goldberg et al. [17, 18] showed a great improvement of running time by combining the reach and the ALT algorithm.

Wagner et al. [42, 43, 45] and Köhler, Möhring and Shilling [28, 31] focused on the coordinates of vertices (thus their algorithm can be applied only to Euclidean networks). Their approach separates vertices by coordinates into k vertex sets. A 0-1 k-vector $\mathbf{e} = (e_1, \ldots, e_k)$ for each edge e is called an edge flag and $e_i = 1$ if the i-th set includes reachable vertices through the shortest path that started from the edge e, otherwise $e_i = 0$. Suppose that a given destination vertex is in the i-th vertex set. Then Dijkstra algorithm can skip all edges with flag $e_i = 0$. There are many discussions on how to separate vertices or how many sets are necessary to reduce the search space. Möhring, Schilling et al. [31] proposed several separation methods such as the grid, quadtree, kd-tree and METIS partitions. They implemented the algorithm with these partitions and reported their experimental results. Moreover, Wagner, Willhalm and Zaroliagis [44] applied the edge flag algorithms to the dynamic scenario of topology-changeable networks.

Delling et al. [10, 27, 34, 35] made a sparse graph from the given network called a highway. The sparsity of the highway graph yields the reduction of the search space. The highway is constructed by removing all edges that do not appear in any shortest paths and each vertex of degree two is replaced by a shortcut edge. The algorithm scans the graph repeatedly to find the highway vertices. The highway can be constructed hierarchically and this approach can also be combined to the other techniques such as the bidirectional search, the A^* search and so on [24, 27].

Holzer et al. [9, 22, 23, 37, 38] proposed an overlay graph approach. The overlay graph is similar to highway, but the structure is quite different. An overlay graph consists of a set of vertices $V' \subseteq V$ which is called a *separator* and a set of edges $E' \subseteq V' \times V'$ such that $(v, w) \in E'$ if and only if there is no inner vertex of any (v, w)-shortest path belonging to V'. Holzer, Schults and Wagner [23] proposed an algorithm to make the smallest overlay graph with respect to the number of edges. Let C_s and C_d be two connected components of graph $(V - V', E)$ that includes the source s and the destination d, respectively. They showed that an (s, d)-shortest path can be found in the (s, d)-graph defined as $(V' \cup C_s \cup C_d, E' \cup E(C_s) \cup E(C_d))$. The overlay graph can be constructed hierarchically without loss of the optimality. The resulting graph may be smaller than the original one and it helpful to reduce the running time. This approach is also expected to be well combined with other techniques.

The transit node routing proposed by Bast et al. [2] uses a set of vertices called *transit nodes*. The transit nodes have the property that for every pair of vertices that are not too close to each other, the shortest path between transit nodes passes through at least one of these transit nodes. They reported that a set of transit nodes with about 10,000 vertices for a network with about 20,000,000 vertices performs well with respect to the running time.

Notice that the above existing algorithms do not treat the TDSP problem with time-dependent edge length. Sanders and Schultes have reviewed various algorithms of this kind and compared their performances. See [36].

The time-dependent case

Actually the general TDSP is NP-hard [40]. To solve some special cases (in which the FIFO property may not hold), the approach of *time-expansion* is often used which transforms the time-dependent network into a time-independent network [1, 4, 5, 39]. This approach requires to discretize the time value and for every time value, a copy of the vertex is maintained in the time-expanded network. The crucial problem here is the huge size of the new graph, which in general results in non-polynomial-time algorithms.

We remark that any time-dependent network can be transformed into an FIFO network if waiting at vertices is allowed. This was pointed out by Orda and Rom [33]. We will discuss it in Section 2.1.

On the other hand, there are studies on the TDSP problem with *arbitrary* departure time, i.e., given a traveling time (or an arrival time) function for each edge, the object is to find the optimal arrival-time function. Orda and Rom [33] proposed an exact single-source time-dependent shortest path algorithm with running time $O^f(mn)$ time, where a *functional complexity* denoted by $O^f(\cdot)$ is the complexity of functional operations such as additions, compositions and taking minimum between two functions. Dean [7, 8] gave a new algorithm for the FIFO networks where each edge length is a piecewise-linear function with a finite number of segments. Dean's algorithm runs in $O(mT \log n)$ time, where T is the sum of the number of segments of all functions. This is faster than the general algorithm by Dreyfus et al. [13, 20, 25, 33].

In this study, we generalize the A^* algorithm and the ALT algorithm to the TDSP problem in FIFO networks with non-negative edge length. We will also discuss generalization of other algorithms later in Section 5.

1.3 Structure of the thesis

The following of the thesis is organized as follows. In Section 2 we give some preliminaries. The proposed algorithm is shown in Section 3. In Section 4, we show an implementation of the proposed algorithm and show some computational results. Finally we conclude our result with remarks in Section 5.

2 Preliminaries and Definitions

In this section, we describe formulations of shortest path problems. Let $G = (V, E)$ be a *directed graph*, where $V = \{v_1, v_2, \ldots, v_n\}$ is a set of n vertices and $E = \{e_1, e_2, \ldots, e_m\} \subseteq V \times V$ is a set of m edges. Note that (v, w) and (w, v) represent distinct edges. The variable s denotes the *source* (starting) vertex, d denotes the *target* (destination) vertex and t denotes the *time*. Each edge $(v, w) \in E$ has a *length function* $l_{vw}(t)$ of time t. When edge length is a constant, it is denoted by l_{vw} for simplicity. An *arrival time function* $F_{vw}(t)$ of edge $(v, w) \in E$ is defined by $F_{vw}(t) = t + l_{vw}(t)$, which represents the arrival time at w when one leaves from v at time t. We assume that a function value can be obtained in $O(1)$ time. Throughout this thesis, we assume each edge length is non-negative, i.e., $l_{vw}(t) \geq 0$.

When all edge lengths in G are constant, the time-dependent network is simply called a *static network* or a network and denoted by (G, l), where l is the set of edge length. An edge (v, w) satisfies the *FIFO property* if $F_{vw}(t_1) \leq F_{vw}(t_2)$ holds for any $0 \leq t_1 \leq t_2$. A function satisfying the FIFO property is called an *FIFO function* and a time-dependent network is called an *FIFO network* if all edge functions satisfy the FIFO property.

A vertex sequence $P = (v_1, v_2, \ldots, v_k)$ is called a *path* if $(v_i, v_{i+1}) \in E$, $i = 1, \ldots, k-1$. When there are no indexes $i \neq j$ with $v_i = v_j$, the path is called simple. A path from a source s to a target d is called an (s, d)-path. For the static case, the length $l(P)$ of an (s, d)-path $P = (s = v_1, \ldots, v_k = d)$ is defined by

$$l(P) = \sum_{i=1}^{k-1} l_{v_i v_{i+1}}.$$

An (s, d)-path P that minimizes $l(P)$ is called an (s, d)-shortest path and the length is denoted by $\mathrm{dist}^*(s, d)$.

For the time-dependent case, an (s, d)-path P with a specified starting time t_s is called an (s, d, t_s)-path and is denoted by (P, t_s). The *arrival time* $A(P, t_s)$ of an (s, d, t_s)-path (P, t_s) is defined by

$$A(P, t_s) = t_s + \sum_{i=1}^{k-1} l_{v_i v_{i+1}}(t_i),$$

where

$$\begin{aligned} &t_1 = t_s, \\ &t_{i+1} = F_{v_i v_{i+1}}(t_i), \quad i = 1, \ldots, k-1. \end{aligned}$$

The value $A(P, t_s) - t_s$ is called the *length* of an (s, d, t_s)-path (P, t_s). An (s, d, t_s)-path (P, t_s) that minimizes the arrival time $A(P, t_s)$ is called an (s, d, t_s)-shortest path and the arrival time $A(P, t_s)$ is called the *earliest arrival time* and its length is denoted by $\mathrm{dist}^*(s, d, t_s)$.

A 3-tuple (s, d, t_s) of a source $s \in V$, a target $d \in V$ and a starting time $t_s \in \mathbb{R}$ is called a *query*.

Given a time-dependent network (G, F) and a query (s, d, t), the time-dependent shortest path problem asks to find an (s, d, t_s)-shortest path and the arrival time. We first show that the FIFO property is important in calculation a shortest path.

Theorem 1 (Halpern [20], Kaufman and Smith [25], Orda and Rom [33]) For an FIFO network (G, F) and a query (s, d, t), there exists an (s, d, t)-shortest path, and it is simple. □

On the other hand, if the FIFO property does not hold, then the problem is NP-hard [40]. In fact, there may exist an (s, d, t)-shortest path with an unlimited number of edges if the FIFO property does not hold. Figure 2 shows an example.

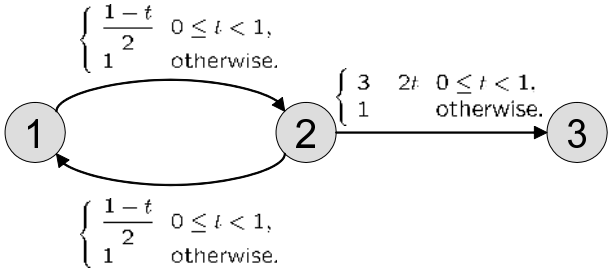


Figure 2: An example of a shortest path with an infinite number of edges. The edge labels denote the edge length functions. The $(1, 3, 0)$-shortest path $(1, 2, 1, 2, \ldots, 1, 2, \ldots, 3)$ has length 2 but the number of edges is infinite. On the other hand, any path with a finite number of edge has length larger than 2.

In the following, we will only deal with FIFO networks (and non-negative edge length) unless stated otherwise and the TDSP problem is described as follows.

TDSP

Input: An FIFO network (G, F) and a query (s, d, t).

Output: An (s, d, t)-shortest path (P, t) and the earliest arrival time $A(P, t)$.

2.1 Modelling

In this section, we discuss the arrival time functions. As stated before, whether the arrival time function satisfies the FIFO property or not is crucial to the complexity of the problem. Orda and Rom [33] indicated that any function can be considered as an FIFO function if we allow unlimited waiting at the vertices. Generally, waiting is considered to be allowed at the vertices, but there can be waiting costs. The costs can be used to disallow waiting at all by using huge values.

Orda and Rom defined the optimal waiting time. Let $l_{vw}(t)$ be a length function of edge (v, w) and $D_{vw}(t, \tau)$ be the function defined by $D_{vw}(t, \tau) = t + \tau + l_{vw}(t + \tau)$, i.e., $D_{vw}(t, \tau)$ represents the arrival time with waiting time τ. $D_{vw}(t)$ is defined by

$$D_{vw}(t) = \min_{\tau \ge 0} \{D_{vw}(t, \tau)\}$$

and then this $D_{vw}(t)$ is an FIFO function and such τ minimizing $D_(t + \tau)$ is called the optimal waiting time. See Figure 3. Without waiting, we arrive at time t_4 when the starting time is t_1. But waiting until time t_2 yields an arrival time $t_3 < t_4$. Figure 4 illustrates how to find $D_{vw}(t)$. The point t_2 can be characterized as the rightmost intersection point

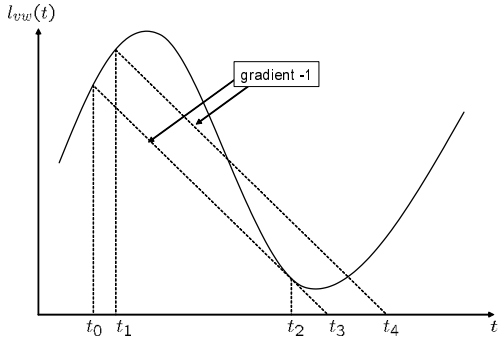


Figure 3: Anyone who started in $[t_0, t_2]$ can arrive at time t_3 by waiting until time t_2 and it achieves the earliest arrival time. Without waiting, starting at time $t_1 \in (t_0, t_2)$ yields an arrival at time $t_4 > t_3$.

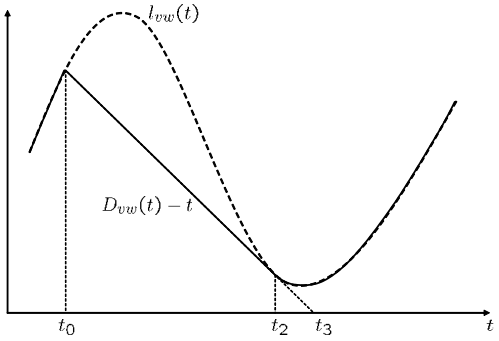


Figure 4: An illustration of $D_{vw}(t)$. The solid line and dashed line represent $D_{vw}(t) - t$ and $l_{vw}(t)$, respectively. It is observed that such a $D_{vw}(t)$ achieves the earliest arrival times for each time.

Note: The x-axis represents a starting time and the y-axis represents an edge length of time.

in $[t_1, t_1 + l_{vw}(t_1)]$ between $l_{vw}(t)$ and the 45° cord closest to the origin. Then anyone starting in $[t_0, t_2]$ should arrive at t_3 to achieve the earliest arrival time. See detail in Orda and Rom [32, 33].

As shown above, FIFO networks can be prepared naturally if there are some time-dependent network data. (The word "naturally" means that waiting at the vertex is a common idea to arrive earlier. We usually wait for the next train of express services instead of the current one of local services). In the rest of this section, we describe how to consider the data of traffic or train timetables.

For the road data, the delay or changing of the arrival time is mainly caused by traffic jams. Usually, for each edge, we assume that we cannot reach the next cross earlier than any car running in front of us. It shows immediately that such a traffic data is an FIFO network.

For the train timetables, overtaking often occurs that breaks the FIFO property. It means that we cannot compute the shortest paths in train networks without modifying these data. However we can consider each edges as multiple edges, where each edge functions satisfies the FIFO properties. Such an edge decomposition exists, and each edge satisfies the FIFO property. Moreover, when we have train timetable data of each services such as express, local etc., each train service satisfies the FIFO property. To compute the shortest path in a time-dependent network with multiple edges, the algorithm suffices to take the earliest arrival times among all multiple edges at each step. (We choose the fastest service when we take trains).

2.2 ALT algorithm

Our approach is based on Goldberg and Harrelson's ALT algorithm [16] for the static problem. In this section, we describe it. ALT algorithm is an efficient algorithm for the

shortest path problem with preprocessed data, which is based on the A^* algorithm. The A^* algorithm is one of the searching algorithms like BFS and DFS. An *estimator* is a value assigned to each vertex v, which is expected to estimate the length of a (v, d)-shortest path for a given target d. The main idea of the A^* algorithm is to use the sum of the interim distance from the source the vertex v and an estimator of v as the priority in searching sequence.

First, we explain the A^* algorithm for the shortest path problem. Consider the shortest path problem from a source vertex s to a target vertex d in a network (G, l) and suppose that we have a function $h : V \to \mathbb{R}$ such that $h(v)$ gives an estimator of a vertex v. We have a function $g : V \to \mathbb{R}$ and a priority f defined by $f(v) = g(v) + h(v)$. Especially, $g^*(v)$ and $h^*(v)$ represent the length of the (s, v)-shortest path and the (v, d)-shortest path, respectively. This means $g^*(v) = \mathrm{dist}^*(s, v)$ and $h^*(v) = \mathrm{dist}^*(v, d)$, and $f^*(v)$ is defined by $f^*(v) := g^*(v) + h^*(v)$. It can be observed that $f^*(v)$ has the same value (i.e., the length of a (s, d)-shortest path) for any vertex that belongs to an (s, d)-shortest path. Let Q and R be a set of vertices. Then the A^* algorithm is described as follows.

A^* algorithm

Input: A network (G, l), a source s, a target d and an estimator $h : V \to \mathbb{R}$.

Output: An (s, d)-shortest path and the length.

1. For all $v \in V - \{s\}$, let $g(v) := \infty$, $\mathrm{prev}(v) :=$ NULL. Let $g(s) := 0$, $Q := \{s\}$ and $R := \emptyset$.
2. `while` $d \notin R$ `do`
   (a) Let $u := \arg\min\{f(v) \mid v \in Q\}$. Remove u from Q and insert it into R.
   (b) For all neighbors v of u with $v \notin R$, `if` $g(v) > g(u) + l_{uv}$, `then` set $g(v) := g(u) + l_{uv}$, $f(v) := g(u) + l_{uv} + h(v)$ and $\mathrm{prev}(v) := u$ and insert v into Q.
3. Output the (s, d)-path by tracing $\mathrm{prev}(v)$ and the length $g(d)$.

For a general estimator, the A^* algorithm does not guarantee the optimality. An estimator h is called *valid* to a given network (G, l) if the A^* algorithm with the estimator h outputs an (s, d)-shortest path for any pair of vertices (s, d).

Definition 1 (*Metric estimator*) For a network (G, l), an estimator h is said *metric* if it satisfies $l_{vw} - h(v) + h(w) \geq 0$ for all edges $(v, w) \in E$.

Theorem 2 (Goldberg and Harrelson [16]) For a network (G, l), an estimator h, a source vertex s and a target vertex d, an (s, d)-shortest path can be found by the A^* algorithm in $O(m + n \log n)$ time if the estimator h is metric. □

The converse of Theorem 2 is not true in general. There exists a network where the A^* algorithm with any estimator h can find an (s, d)-shortest path for any pair of vertices (s, d). See Figure 5 to get an example. The A^* algorithm is equivalent to Dijkstra algorithm when $h(v) = 0$ for all vertices. However, the search space (the size of R when the algorithm terminates) can be worse than Dijkstra algorithm. For example, if an estimator is given as $h(d) = \infty$ and $h(v) = 0$ for all $v \in V - \{d\}$, the A^* algorithm scans all vertices before it reaches d. The following lemma shows that a *bounded* and *non-negative* estimator surely reduces the search space than Dijkstra algorithm.

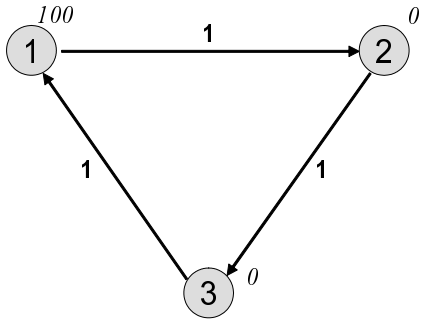


Figure 5: An example of a network where the A^* algorithm with any estimator h outputs an (s, d)-shortest path for any pair of vertices (s, d). The edge labels in bold Roman denote the edge length and the numbers in Italics in the upper right in each vertex denote an example of a non-metric estimator.

Definition 2 (*Bounded estimator*) For a network (G, l) and a target vertex d, an estimator h is *bounded* if it satisfies $h(v) \leq \mathrm{dist}^*(v, d)$ for all vertices $v \in V$.

Lemma 1 For a network (G, l), a bounded estimator h, a source s and a target d, let R be a set of vertices in the A^* algorithm. Then $f(v) \leq g^*(d)$ holds for all $v \in R$.

Proof. Proved in Section 3 as a generalized version. □

This lemma says that the algorithm never scans vertices with $f(v) > g^*(d)$. When an estimator h is bounded and non-negative, the search space is no more than that need by Dijkstra algorithm. For example, if all estimators have the exact (v, d)-shortest path length, the algorithm never scans vertices which does not appear in some (s, d)-shortest path. By these observations, it can be expected to reduce the running time in practice by the A^* algorithm than Dijkstra algorithm if a metric, bounded and non-negative estimator is given or it can be obtained quickly enough.

ALT algorithm

The most remarkable point of ALT algorithm by Goldberg and Harrelson [16] is that ALT algorithm can compute a metric, bounded and non-negative estimator in small time and furthermore it requires only $O(n)$ spaces to store the preprocessed data.

The obvious metric, bounded and non-negative estimator of v to d is the (v, d)-shortest path length. However, this estimator requires $O(n^2)$ space to store all pairs' shortest paths. The size of the space is not a crucial issue if the network size is small, but for a large network, we cannot prepare such a large space. For example, for a network with 23,000,000 vertices 58,000,000 edges in a navigation system, the input size is 2 GB but n^2 is 4×10^9 GB, which is impossible to prepare now.

The ALT algorithm uses vertices called *landmarks*. Let $\mathbb{L} = \{L_1, \ldots, L_k\}$ be a set of vertices. Based on the optimality of the shortest path, it holds

$$\mathrm{dist}^*(u, w) \leq \mathrm{dist}^*(u, v) + \mathrm{dist}^*(v, w) \text{ for any 3-tuple of vertices } (u, v, w).$$

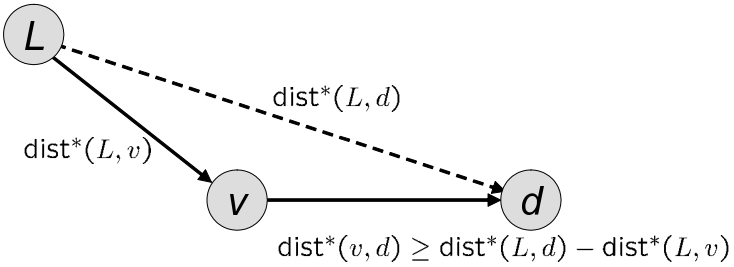


Figure 6: An illustration of the landmark estimator. By optimality, the length of the solid path is no less than the dashed path. In particular, when the solid path corresponds to an (L, d)-shortest path, the landmark estimator gives the (v, d)-shortest path length.

Goldberg and Harrelson defined an estimator $h'_l(v, L)$ of v with respect to a landmark $L \in \mathbb{L}$ by $h'_l(v, L) = \mathrm{dist}^*(L, d) - \mathrm{dist}^*(L, v)$. By definition, it holds

$$h'_l(v, L) = \mathrm{dist}^*(L, d) - \mathrm{dist}^*(L, v) \leq \mathrm{dist}^*(v, d) \text{ for any vertex } v.$$

The estimator h'_l is metric since

$$\begin{aligned} l_{vw} - h'_l(v, L) + h'_l(w, L) &= l_{vw} - \mathrm{dist}^*(L, d) + \mathrm{dist}^*(L, v) + \mathrm{dist}^*(L, d) - \mathrm{dist}^*(L, w) \\ &= l_{vw} + \mathrm{dist}^*(L, v) - \mathrm{dist}^*(L, w) \\ &\geq \mathrm{dist}^*(L, v) + \mathrm{dist}^*(v, w) - \mathrm{dist}^*(L, w) \\ &\geq 0 \end{aligned}$$

holds for any edge $(v, w) \in E$.

Figure 6 illustrates the triangle inequality for the landmark estimator $h'_l(v, L)$. To compute a larger bounded estimator, one can take the maximum estimator for all landmarks $L \in \mathbb{L}$. Then for a set of landmarks $\mathbb{L}$ and each vertex v, the *landmark estimator* is defined by

$$h_l(v) = \max\{0, \max\{\mathrm{dist}^*(L, d) - \mathrm{dist}^*(L, v) \mid L \in \mathbb{L}\}\}. \tag{1}$$

Clearly, an estimator which always returns zero is metric, bounded and non-negative. For the landmark estimator h_l, the following lemma holds.

Lemma 2 (Goldberg and Harrelson [16]) For a network (G, l), a target vertex d and a set of landmarks $\mathbb{L} \subseteq V$, the landmark estimator h_l defined by (1) is metric, bounded and non-negative. □

3 Time-dependent ALT algorithm

In the previous section, we described the ALT algorithm which can be applied only to the static shortest path problem. In this section, we generalize the A^* algorithm and the ALT algorithm to the time-dependent shortest path problem.

3.1 Time-dependent A^* algorithm

We first define the *time-dependent A^* algorithm* for the time-dependent shortest path problem. Basic idea of the algorithm is similar to the A^* algorithm, except for the time dependency of the estimators.

An *time-dependent estimator* is a value assigned to each vertex v and time $t \geq 0$, which is expected to estimate the length of a (v, d, t)-shortest path for a given target d. Consider the time-dependent shortest path problem for a query (s, d, t_s) in an FIFO network (G, F). Suppose that we have a function $h : (V, \mathbb{R}) \to \mathbb{R}$ such that $h(v, t)$ gives a time-dependent estimator of a vertex v and a time $t \geq 0$. A function g and a priority f are defined by $g : V \to \mathbb{R}$ and $f(v) = g(v) + h(v, g(v))$, respectively. Especially, $g^*(v)$ and $h^*(v, t)$ represent the earliest arrival time of an (s, v, t_s)-shortest path and the length of a (v, d, t)-shortest path, respectively. This means $g^*(v) = t_s + \mathrm{dist}^*(s, v, t_s)$ and $h^*(v, t) = \mathrm{dist}^*(v, d, t)$, and $f^*(v)$ is defined by $f^*(v) := g^*(v) + h^*(v, t)$. We define the total ordering $\prec$ among all $f(v)$ by the following definition. For a time-dependent estimator h and any vertices $u, v \in V$,

$$f(u) \prec f(v) \Longleftrightarrow \begin{cases} g(u) + h(u, g(u)) < g(v) + h(v, g(v)), \\ g(u) + h(u, g(u)) = g(v) + h(v, g(v)) \ \text{ and } \ g(u) < g(v). \end{cases}$$

The equation is defined by the followings.

$$f(u) = f(v) \Longleftrightarrow g(u) + h(u, g(u)) = g(v) + h(v, g(v)) \ \text{ and } \ g(u) = g(v).$$

Let Q and R be a set of vertices. Then the time-dependent A^* algorithm is described as follows.

Time-dependent A^* algorithm

Input: An FIFO network (G, F), a query (s, d, t_s) and a time-dependent estimator $h : (V, \mathbb{R}) \to \mathbb{R}$.

Output: An (s, d, t_s)-shortest path and the earliest arrival time.

1. For all $v \in V - \{s\}$, let $g(v) := \infty$, $\mathrm{prev}(v) :=$ NULL. Let $g(s) := t_s$, $Q := \{s\}$ and $R := \emptyset$.
2. `while` $d \notin R$ `do`
   (a) Let $u \in Q$ such that $f(u) \preceq f(v)$ for any $v \in Q$. Remove u from Q and insert it into R.
   (b) For all neighbors v of u with $v \notin R$, `if` $g(v) > F_{uv}(g(u))$, `then` set $g(v) := F_{uv}(g(u))$, $f(v) := F_{uv}(g(u)) + h(v, F_{uv}(g(u)))$ and $\mathrm{prev}(v) := u$ and insert v into Q.
3. Output the (s, d, t_s)-path by tracing $\mathrm{prev}(v)$ and the length $g(d)$.

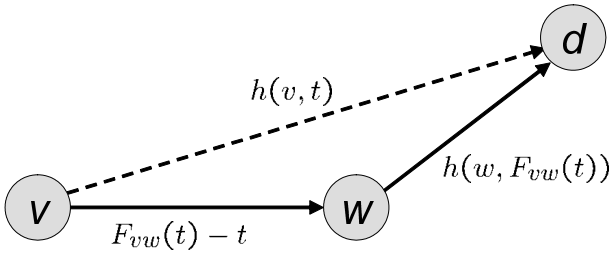


Figure 7: An illustration of a metric time-dependent estimator. Consider values $h(v,t)$, $h(w, F_{vw}(t))$ and $F_{vw}(t) - t$ as "length", then these lengths satisfy the triangle inequality, i.e., the length of the dashed path is no more than the length of the solid path.

For a general time-dependent estimator, the time-dependent A^* algorithm does not guarantee the optimality. A time-dependent estimator h is called *valid* to a given time-dependent network (G, F) if the time-dependent A^* algorithm with the time-dependent estimator h outputs an (s, d, t_s)-shortest path for any query (s, d, t_s). Let us describe it in details. First, we define a *metric estimator* and an *FIFO estimator*.

Definition 3 (*Metric estimator*) For an FIFO network (G, F), a time-dependent estimator h is called a *metric* estimator if it satisfies $F_{vw}(t) - t - h(v,t) + h(w, F_{vw}(t)) \geq 0$ for all edges $(v, w) \in E$ and any time $t \geq 0$.

Figure 7 illustrates the triangle inequality of a metric estimator. It shows immediately that $\mathrm{dist}(v, w, t) - h(v,t) + h(w, t + \mathrm{dist}(v, w, t)) \geq 0$ holds for any (v, w, t)-path, where $\mathrm{dist}(v, w, t)$ is the length of the (v, w, t)-path.

Definition 4 (*FIFO estimator*) For an FIFO network (G, F), a time-dependent estimator h is called an *FIFO estimator* if it satisfies $t_1 + h(v, t_1) \leq t_2 + h(v, t_2)$ for all vertices $v \in V$ and any time $0 \leq t_1 \leq t_2$.

Theorem 3 For an FIFO network (G, F), a query (s, d, t_s) and a time-dependent estimator h, an (s, d, t_s)-shortest path can be found by the time-dependent A^* algorithm in $O(m + n \log n)$ time if the estimator h is a metric FIFO estimator. □

Proof. Let Q and R be a set of vertices in the time-dependent A^* algorithm Step (2). It suffices to show that each vertex $w \in R$ satisfies $g(w) = g^*(w)$.

Clearly $g(s) = g^*(s)$. Assume $g(w) = g^*(w)$ for all $w \in R$. Let u be the vertex chosen by Step (2a) so that $f(u) \preceq f(v)$ holds for an $v \in Q$. We want to show $g(u) = g^*(u)$. Let P be an (s, u, t_s)-shortest path and $v \in P$ be the first vertex such that $v \notin R$. There exists such a vertex v since at least $u \in P$. See Figure 8. Since F satisfies the FIFO property and each vertex on P from s to v is in R, $g(v) = g^*(v)$ holds. Notice that $g^*(u) = g^*(v) + \mathrm{dist}^*(v, u, g^*(v))$ can be achieved by the path P. Since $f(u) \preceq f(v)$ holds for any $v \in Q$, it holds

$$g(u) + h(u, g(u)) < g^*(v) + h(v, g^*(v)), \text{ or} \tag{2}$$

$$g(u) + h(u, g(u)) = g^*(v) + h(v, g^*(v)) \text{ and } g(u) \leq g^*(v). \tag{3}$$

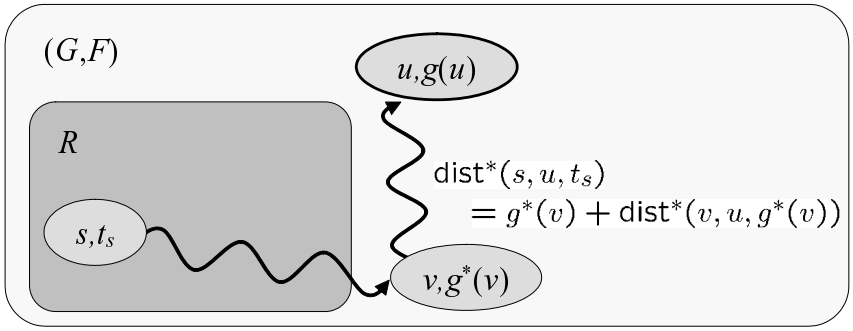


Figure 8: An illustration of the proof of Theorem 3. A pair of values in each oval denotes the vertex and the arrival time, respectively. The solid line represents the (s, u, t_s)-shortest path P.

If (2) holds, then

$$\begin{aligned} g(u) + h(u, g(u)) &< g^*(v) + h^*(v, g^*(v)) = g^*(u) - \mathrm{dist}^*(v, u, g^*(v)) + h(v, g^*(v)) \\ &\leq g^*(u) - h(v, g^*(v)) + h(u, g(u)) + h(v, g^*(v)) = g^*(u) + h(u, g^*(u)) \end{aligned}$$

implies $g(u) < g^*(u)$ since h satisfies the FIFO property, a contradiction. Hence only (3) holds. Now we have $g(u) = g^*(u)$ since $g(u) \leq g^*(v) \leq g^*(u)$ and $g(u) \geq g^*(u)$ hold by definition. For each $w \in R$, $g(w)$ does not change, therefore the correctness is proved.

We apply the Fibonacci heaps [14] to maintain the set of vertices Q. At each iteration, the time-dependent A^* algorithm inserts one vertex into R and no vertex $w \in R$ is removed from R. Therefore the number of iterations is at most n. The algorithm requires at most n DELETEMIN-operations, n INSERT-operations and m DECREASEKEY-operations. Therefore the complexity is $O(m + n \log n)$. □

The converse of Theorem 3 is not true in general. There exists an FIFO network (G, F) where the time-dependent A^* algorithm with any time-dependent estimator h outputs an (s, d, t)-shortest path for any query (s, d, t). See Figure 9 to get an example.

The time-dependent A^* algorithm is equivalent to the generalized Dijkstra algorithm when $h(v, t) = 0$ for all vertices and time. As shown above, any metric FIFO estimator guarantees the optimality of the time-dependent A^* algorithm. However, the search space (the size of R when the algorithm terminates) may be worse than generalized Dijkstra algorithm. (An example can be shown in the same way as the static version in Section 2.2). The following lemma shows a *bounded* and *non-negative* estimator surely reduces the search space than the generalized Dijkstra algorithm.

Definition 5 (*Bounded estimator*) For an FIFO network (G, F) and a target vertex d, a time-dependent estimator h is called a *bounded estimator* if it satisfies $h(v, t) \leq \mathrm{dist}^*(v, d, t)$ for all vertices $v \in V$ and any time $t \geq 0$.

Lemma 3 For an FIFO network (G, F), a query (s, d, t_s) and a bounded time-dependent estimator h, let R be a set of vertices in the time-dependent A^* algorithm. Then $f(v) \leq g^*(d)$ holds for all $v \in R$.

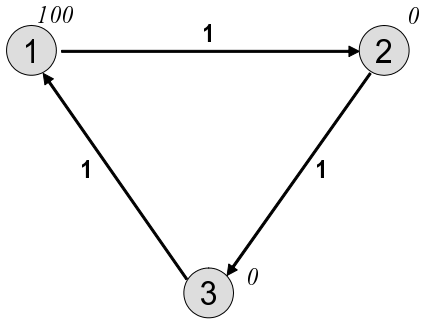


Figure 9: An example of an FIFO network where the time-dependent A^* algorithm with any estimator h outputs an (s, d, t)-shortest path for any query (s, d, t). The edge labels in bold Roman denote the edge length and the numbers in Italics in the upper right in each vertex denote an example of a non-metric time-dependent estimator.

Proof. We show that each vertex v inserted by Step (2a) satisfies $f(v) \leq g^*(d)$. Let Q be a set of vertex in the algorithm Step (2). Let P be an (s, d, t_s)-shortest path, w be the first vertex along P such that $w \notin R$ and u be the vertex chosen by Step (2a) so that $f(u) \preceq f(v)$ holds for any $v \in Q$. Hence $f(u) \leq f(w)$ holds. Since F satisfies the FIFO property and each vertex on P from s to w is in R, $g(w) = g^*(w)$ holds. Then

$$\begin{aligned} f(u) \leq f(w) &= g(w) + h(w, g(w)) = g^*(w) + h(w, g^*(w)) \\ &\leq g^*(w) + \mathrm{dist}^*(w, d, g^*(w)) = g^*(d). \end{aligned}$$

For each $v \in R$, $g(v)$ is no longer changed. Therefore the proof has been done. □

This proof also shows Lemma 1. This lemma says that the algorithm never scans vertices with $f(v) > g^*(d)$. When a time-dependent estimator h is bounded and non-negative, the search space is no more than the generalized Dijkstra algorithm. For example, if all estimators have the exact (v, d, t_s)-shortest path length, the algorithm never scans vertices which does not appear in some (s, d, t_s)-shortest path. By these observations, it can be expected to reduce the running time than the generalized Dijkstra algorithm in practice if a metric FIFO, bounded and non-negative estimator is given or it can be obtained quickly enough. In the next subsection, we will show such a time-dependent estimator, that is a time-dependent estimator satisfying the next property. For any vertex $v \in V$, edge $(v, w) \in E$ and time $t \geq 0$, $t_1 \leq t_2$,

- $F_{vw}(t) - t - h(v, t) + h(w, F_{vw}(t)) \geq 0$ (metric),
- $t_1 + h(v, t_1) \leq t_2 + h(v, t_2)$ (FIFO property),
- $0 \leq h(v, t) \leq \mathrm{dist}^*(v, d, t)$ (bounded and non-negative).

3.2 Construction of the time-dependent ALT algorithm

In this section, we explain how to construct a *time-dependent ALT algorithm*. The framework of the algorithm is similar to the ALT algorithm except for the time-dependent

estimators. The most crucial point in designing a time-dependent ALT algorithm is how to construct metric FIFO, bounded and non-negative estimators. For a straightforward extension of the ALT algorithm, the estimator used in Section 2.2 would require to prepare the shortest path lengths for all starting times from each landmark. This motivated us to find a good estimator which does not require such expensive samplings.

We define metric FIFO, bounded and non-negative estimators based on the triangle inequality. We call our new estimators the *min-length estimator*, the *sample-time estimator* and the *combined estimator*. In the rest of this section, we describe how to calculate these estimators and prove that they are both metric FIFO, bounded and non-negative estimators. First, we show an important lemma.

Lemma 4 The maximum function of two metric FIFO and bounded estimators is also a metric FIFO and bounded estimator.

Proof. Let h_1 and h_2 be two metric FIFO and bounded estimators. Denote the maximum function of h_1 and h_2 by h. For an vertex $v \in V$, it is obviously bounded by definition. For each edge $(v, w) \in E$ and a time $t \geq 0$, let h_i be the estimator with $h_i(v, t) \geq h_k(v, t)$ for $k = 1, 2$ and h_j be the estimator with $h_j(w, F_{vw}(t)) \geq h_k(w, F_{vw}(t))$ for $k = 1, 2$. Then we have

$$\begin{aligned} F_{vw}(t) - t - h(v, t) + h(w, F_{vw}(t)) &= F_{vw}(t) - t - h_i(v, t) + h_j(w, F_{vw}(t)) \\ &\geq F_{vw}(t) - t - h_i(v, t) + h_i(w, F_{vw}(t)) \geq 0, \end{aligned}$$

which shows that $h(v, t)$ is metric. At last, for any time $0 \leq t_1 \leq t_2$, let h_i be the estimator with $h_i(v, t_1) \geq h_k(v, t_1)$ for $k = 1, 2$ and h_j be the estimator with $h_j(w, t_2) \geq h_k(w, t_2)$ for $k = 1, 2$. Then we have

$$\begin{aligned} h(v, t_2) + t_2 - h(v, t_1) - t_1 &= h_j(v, t_2) + t_2 - h_i(v, t_1) - t_1 \\ &\geq h_i(v, t_2) + t_2 - h(v, t_1) - t_1 \geq 0, \end{aligned}$$

which shows that $h(v, t)$ satisfies the FIFO property. □

Min-length estimator

Let $l' : E \to \mathbb{R}$ be the *minimum edge length* function defined by

$$l'_{vw} = \min\{F_{vw}(\tau) - \tau \mid \tau \geq 0\}.$$

Let $\mathrm{dist}'(v, w)$ be the length of a (v, w)-shortest path with respect to l'. Clearly it holds

$$\mathrm{dist}'(u, w) \leq \mathrm{dist}'(u, v) + \mathrm{dist}'(v, w) \text{ for any 3-tuple of vertices } (u, v, w).$$

We define a time-dependent estimator h_M of a vertex v and a time $t \geq 0$ with respect to L by $h'_M(v, t, L) = \mathrm{dist}'(L, d) - \mathrm{dist}'(L, v)$. By definition, it holds

$$\begin{aligned} h'_M(v, t, L) &= \mathrm{dist}'(L, d) - \mathrm{dist}'(L, v) \\ &\leq \mathrm{dist}'(v, d) \leq \mathrm{dist}^*(v, d, t) \text{ for any vertex } v \text{ and time } t. \end{aligned}$$

The followings show that the estimator h'_M is a metric and FIFO estimator.

Lemma 5 For an FIFO network (G, F) and a landmark $L \in V$, the time-dependent estimator h'_M is a metric estimator.

Proof. For any edge $(v, w) \in E$ and time $t \geq 0$, it holds

$$\begin{aligned} &F_{vw}(t) - t - h'_M(v, t, L) + h'_M(w, F_{vw}(t), L) \\ &\quad = F_{vw}(t) - t - \mathrm{dist}'(L, d) + \mathrm{dist}'(L, v) + \mathrm{dist}'(L, d) - \mathrm{dist}'(L, w) \\ &\quad = F_{vw}(t) - t + \mathrm{dist}'(L, v) - \mathrm{dist}'(L, w) \\ &\quad \geq \mathrm{dist}'(L, v) + \mathrm{dist}'(v, w) - \mathrm{dist}'(L, w) \geq 0. \end{aligned}$$

□

Lemma 6 For an FIFO network (G, F) and a landmark $L \in V$, the estimator h'_M is an FIFO estimator.

Proof. By definition, it holds $t_1 + h'_M(v, t_1, L) = t_2 + h'_M(v, t_2, L)$ for any vertex v and time $0 \leq t_1 \leq t_2$. □

Then for a set of landmarks $\mathbb{L}$, each vertex v and time $t \geq 0$, the *min-length estimator* $h_M(v, t)$ is defined by

$$h_M(v, t) = \max\{0, \max\{\mathrm{dist}'(L, d) - \mathrm{dist}'(L, v) \mid L \in \mathbb{L}\}\}.$$

Clearly, a time-dependent estimator which always returns zero is a metric FIFO, bounded and non-negative estimator. By Lemma 4, the min-length estimator is a metric FIFO, bounded and non-negative estimator.

Sample-time estimator

Let $T_p = \{t_1, \ldots, t_p\}$ be p samplings of time such that $t_i < t_{i+1}$ for each $1 \leq i \leq p - 1$. Let $\alpha_{L,v}(t)$ be a function of time t with respect to a landmark L and a vertex v defined as follows.

$$\alpha_{L,v}(t) = \max\{\tau \mid \tau \in T_p, \mathrm{dist}^*(L, v, \tau) + \tau \leq t\}. \tag{4}$$

A time-dependent estimator $h'_S(v, t, L)$ of a vertex v and a time t with respect to L is defined by

$$h'_S(v, t, L) = \begin{cases} \max\{0, \mathrm{dist}^*(L, d, \alpha_{L,v}(t)) + \alpha_{L,v}(t) - t\}, & \text{if } \mathrm{dist}^*(L, v, t_1) + t_1 \leq t, \\ 0, & \text{otherwise.} \end{cases}$$

We are now going to show $h'_S(v, t, L)$ is a metric and FIFO estimator.

Lemma 7 For an FIFO network (G, F) and a landmark $L \in V$, the time-dependent estimator h'_S is a metric estimator.

Proof. For any time $t \geq 0$, there are tree considerable cases, that

1. $t_1 + \mathrm{dist}^*(L, v, t_1) \leq t$ and $t_1 + \mathrm{dist}^*(L, w, t_1) \leq F_{vw}(t)$,
2. $t_1 + \mathrm{dist}^*(L, v, t_1) > t$ and $t_1 + \mathrm{dist}^*(L, w, t_1) \leq F_{vw}(t)$,
3. otherwise.

Note that $\alpha_{L,w}(F_{vw}(t)) \geq \alpha_{L,v}(t)$ holds for any $t \geq t_1 + \mathrm{dist}^*(L, v, t_1)$. See Lemma 8 to prove. For simplicity, let $t' = \alpha_{L,v}(t)$ and $t'' = \alpha_{L,w}(F_{vw}(t))$. If the case 1, we have to consider two cases, that (i) $h'_S(v, F_{vw}(t), L) > 0$ and (ii) $h'_S(v, F_{vw}(t), L) = 0$. For the case (i), it holds

$$\begin{aligned} F_{vw}(t) - t - h'_S(v, t, L) + h'_S(w, F_{vw}(t), L) \\ &= F_{vw}(t) - t - \mathrm{dist}^*(L, d, t') - t' + t + \mathrm{dist}^*(L, d, t'') + t'' - F_{vw}(t) \\ &= \mathrm{dist}^*(L, d, t'') + t'' - \mathrm{dist}^*(L, d, t') - t' \geq 0, \end{aligned}$$

since $t'' \geq t'$. For the case (ii), $h'_S(v, F_{vw}(t), L) = 0$ implies $\mathrm{dist}^*(L, d, t'') + t'' - F_{vw}(t) \leq 0$. Then it holds

$$\begin{aligned} F_{vw}(t) - t - h'_S(v, t, L) + h'_S(w, F_{vw}(t), L) &\geq F_{vw}(t) - t - h'_S(v, t, L) \\ &\geq \mathrm{dist}^*(L, d, t'') + t'' - (\mathrm{dist}^*(L, d, t') + t') \\ &\geq 0, \end{aligned}$$

since $t'' \geq t'$. If the case 2, it holds

$$\begin{aligned} F_{vw}(t) - t - h'_S(v, t, L) + h'_S(w, F_{vw}(t), L) &= F_{vw}(t) - t + h'_S(w, F_{vw}(t), L) \\ &\geq F_{vw}(t) - t \geq 0. \end{aligned}$$

If the case 3, $t_1 + \mathrm{dist}^*(L, w, t_1) > F_{vw}(t)$ implies $t_1 + \mathrm{dist}^*(L, v, t_1) > t$. Then it holds

$$F_{vw}(t) - t - h'_S(v, t, L) + h'_S(w, F_{vw}(t), L) = F_{vw}(t) - t \geq 0.$$

Therefore the time-dependent estimator h'_S is metric. □

Lemma 8 For an FIFO network (G, F) and a landmark $L \in V$, it holds $\alpha_{L,w}(F_{vw}(t)) \geq \alpha_{L,v}(t)$ for any edge $(v, w) \in E$ and time $t \geq \mathrm{dist}^*(L, v, t_1) + t_1$.

Proof. Let $t' = \alpha_{L,v}(t)$ and $t'' = \alpha_{L,w}(F_{vw}(t))$ for simplicity. Consider an (L, w, t')-path P which consists of (L, v, t')-shortest path and an edge (v, w). Along the path P, $\mathrm{dist}^*(L, v, t') + t' \leq t$ holds and $F_{vw}(\mathrm{dist}^*(L, v, t') + t') \leq F_{vw}(t)$ by the FIFO property. Notice that $t'' \in T_p$ is the maximum such that $\mathrm{dist}^*(L, w, t'') + t'' \leq F_{vw}(t)$. Therefore $t'' \geq t'$ holds. See Figure 10. □

Next we show the time-dependent estimator h'_S satisfies the FIFO property.

Lemma 9 For an FIFO network (G, F) and a landmark $L \in V$, the time-dependent estimator h'_S is an FIFO estimator.

Proof. If $\tau_1 < t_1 + \mathrm{dist}^*(L, v, t_1)$ holds, it is obviously that $h'_S(L, v, \tau_2) + \tau_2 \geq h'_S(L, v, \tau_1) + \tau_1$ holds. Then assume that $\tau_1 \geq t_1 + \mathrm{dist}^*(L, v, t_1)$ holds. It is clear that $\alpha_{L,v}(\tau_1) \leq \alpha_{L,v}(\tau_2)$ holds. Hence it holds

$$\begin{aligned} &h'_S(v, \tau_2, L) + \tau_2 - h'_S(v, \tau_1, L) - \tau_1 \\ &\quad = \mathrm{dist}^*(L, d, \alpha_{L,v}(\tau_2)) + \alpha_{L,v}(\tau_2) - \tau_2 + \tau_2 - \mathrm{dist}^*(L, d, \alpha_{L,v}(\tau_1)) - \alpha_{L,v}(\tau_1) + \tau_1 - \tau_1 \\ &\quad = \mathrm{dist}^*(L, d, \alpha_{L,v}(\tau_2)) + \alpha_{L,v}(\tau_2) - \mathrm{dist}^*(L, d, \alpha_{L,v}(\tau_1)) - \alpha_{L,v}(\tau_1) \geq 0. \end{aligned}$$

Therefore the time-dependent estimator h'_S satisfies the FIFO property. □

Finally, we show the time-dependent estimator h'_S is bounded and non-negative.

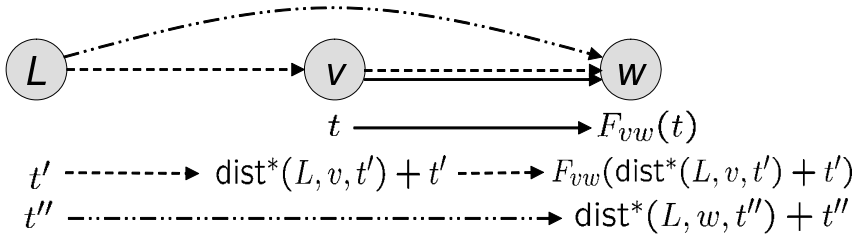


Figure 10: An illustration of proof of Lemma 8. For simplicity, $\alpha_{L,v}(t)$ is denoted by t' and $\alpha_{L,w}(F_{vw}(t))$ is denoted by t''.

Lemma 10 For an FIFO network (G, F) and a landmark $L \in V$, the time-dependent estimator h'_S is bounded and non-negative for any vertex v and time $t \geq 0$.

Proof. Clearly, h'_S is non-negative by definition. For simplicity, let $t' = \alpha_{L,v}(t)$. By definition, $t' + \mathrm{dist}^*(L, v, t') \leq t$ holds for any time $t \geq t_1 + \mathrm{dist}^*(L, v, t_1)$. For any vertex v and time $t \geq t_1 + \mathrm{dist}^*(L, v, t_1)$, it holds

$$\begin{aligned} h'_S(v, t, L) &= \mathrm{dist}^*(L, d, t') + t' - t \\ &= \mathrm{dist}^*(L, d, t') - \mathrm{dist}^*(L, v, t') + \mathrm{dist}^*(L, v, t') + t' - t \\ &\leq \mathrm{dist}^*(v, d, \mathrm{dist}^*(L, v, t') + t') + \mathrm{dist}^*(L, v, t') + t' - t \\ &\leq \mathrm{dist}^*(v, d, t) + t - t = \mathrm{dist}^*(v, d, t). \end{aligned}$$

For any vertex v and time $t < t_1 + \mathrm{dist}^*(L, v, t_1)$, it is obviously bounded and non-negative. Therefore the time-dependent estimator h'_S is bounded and non-negative. □

Then for a set of landmarks $\mathbb{L}$, a set of p samplings T_p, each vertex v and time $t \geq 0$, the *sample-time estimator* $h_S(v, t)$ is defined by

$$h_S(v, t) = \max\{0,\ \max\{\mathrm{dist}^*(L, d, \alpha_{L,v}(t)) + \alpha_{L,v}(t) - t \mid L \in \mathbb{L}\}\}.$$

Clearly, an estimator which always returns zero is a metric FIFO, bounded and non-negative estimator. By Lemma 4, the sample-time estimator h_S is a metric FIFO, bounded and non-negative estimator for any vertex v and time $t \geq 0$.

Combine the two estimators

The two estimators defined above can be combined and used to obtain a larger bounded estimator. Let $\mathbb{L} \subseteq V$ be a set of landmarks and h_M and h_S be the min-length estimator and the sample-time estimator, respectively. Then for a set of landmarks $\mathbb{L}$, a set of p samplings T_p, each vertex v and time $t \geq 0$, the *combined estimator* $h_C(v, t)$ is defined by

$$h_C(v, t) = \max\{h_M(v, t),\ h_S(v, t)\}$$

It is clear that the combined estimator is a metric FIFO and bounded estimator by Lemma 4. And by definition of h_M and h_S, it is non-negative. By Lemma 3, a larger bounded estimator surely reduces the search space. In Section 4, we will compare the proposed time-dependent ALT algorithm with the generalized Dijkstra algorithm with respect to the running time and the search space.

4 Implementations

In this section, we show the detail of the implementation and report the results of the proposed time-dependent ALT algorithm. First, we explain how to select landmarks from given time-dependent networks and we then describe about how to assign the arrival time functions in Section 4.2. In the section 4.3, we report the experimental results for instances provided by DIMACS [41]. Many results for the (traditional) ALT algorithm and the related works have been reported [11, 16–18].

Each instance provided by DIMACS [41] represents a directed graph with constant integer edge length. Each vertex is embedded in a plane and given its (x, y)-coordinate. (It represents the longitude and the latitude). The (x, y)-coordinate of each vertex helps to decide a set of landmarks, but it is not used in our time-dependent ALT algorithm.

4.1 Landmark selection

In this section, we explain how to select landmarks. Let k denote the number of landmarks we would like to choose. According to Goldberg and Harrelson [16], we can use the *farthest* selection and the *planar* selection, and in addition, we use the *grid* selection. Now let the time-dependent network be embedded in the 2-dimensional surface and suppose each vertex is embedded in a plane and has an (x, y)-coordinate.

The farthest selection is a greedy method, which works as follows. Pick a start vertex and find a vertex v_1 that is the farthest away from v_1 with respect to the Euclidean distance between their coordinates. Add v_1 to the set of landmarks. Proceed iterations; in each iteration find a vertex that is the farthest away from the current set of landmarks and adding the vertex to the set. The procedure terminates when the size of landmark set is equal to k. This algorithm can be viewed as an approximation to the problem of selecting a set of k vertices so that the minimum distance between a pair of selected vertices is maximized.

The planar selection works as follows. First, find a vertex c closest to the center of the embedding. Divide the embedding into k pie-slice sectors centered at c, each contains approximately the same number of vertices. For each sector, pick a vertex farthest away from the center with the starting time $t = 0$. This algorithm locates landmarks at the boundary.

The grid selection works as follows. Separate a (x, y)-plane in which each vertex is embedded into k regions (see Figure 11) and select k vertices arbitrarily from each region. This method can be also viewed as an approximation to the problem of selecting a set of k vertices so that the minimum distance between a pair of selected vertices is maximized.

We evaluate these selections by comparing the running times with each of them. By definition of our estimators, the landmark L seems to contribute a strong estimation of a vertex v if the vertex v is approximately on the way from L to d. It is also that landmarks which is allocated close to each other may not contribute a strong estimation even if there are many landmarks.

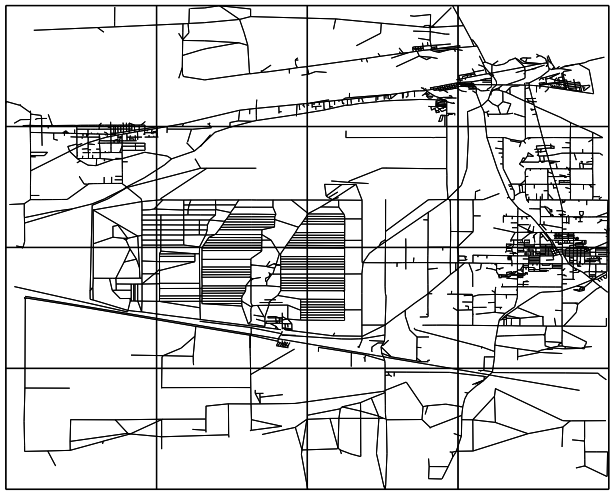


Figure 11: An example of a 4×4 grid partitioning of the graph.

Table 2: Instances. The column of "Length distribution" shows the number of edges whose length is in the range.

			Length distribution [#edge]			
Name	#vertex	#edge	0–1800	1801–3600	3601–	Remarks
NW	1,207,745	2,840,208	1,584,224	507,964	748,020	North West area
BAY	321,270	800,172	609,406	129,686	61,080	BAY area
S1	19,292	47,278	29,896	8,580	8,802	trimmed BAY
S2	3,931	9,744	6,656	1,518	1,570	trimmed BAY

4.2 Instances

In this section, we explain about instances. Each network with constant edge length is provided by DIMACS web site [41].

We implement our algorithm for four graphs denoted by NW, BAY, S1 and S2, respectively. Both NW and BAY are original data from DIMACS web site, and S1 and S2 are generated by trimming and eliminating vertices from BAY to be strongly connected. Table 2 shows the number of vertices and edges of each instance.

Each of these instances has constant edge lengths. To make these instances time-dependent, we now assign edge length functions generated as follows. Consider we have the edge length denoted by l_{vw} for each edge $(v, w) \in E$, and every edge functions $l_{vw}(t)$ will be defined in $(-\infty, \infty)$ as periodic functions with period $T = 86400$ [second], i.e., $l_{vw}(t) = l_{vw}(t + T)$ for any time t. We prepare two kind of edge functions called *random* and *practical*.

The random function is generated as follows. Consider to make a piecewise linear function by choosing segment points randomly. Let k be the number of segments and $r \geq 1$ be a coefficient that the maximum value of each edge length is restricted by rl_{vw}. First, let $x_0 = 0$ and $x_k = T$ and select $k-1$ x-values $(x_1, x_2, \ldots, x_{k-1})$ arbitrarily in $(0, T)$. Choose a random value from $[l_{vw}, rl_{vw}]$ as y_0 and set (x_0, y_0) as the first point. Proceed

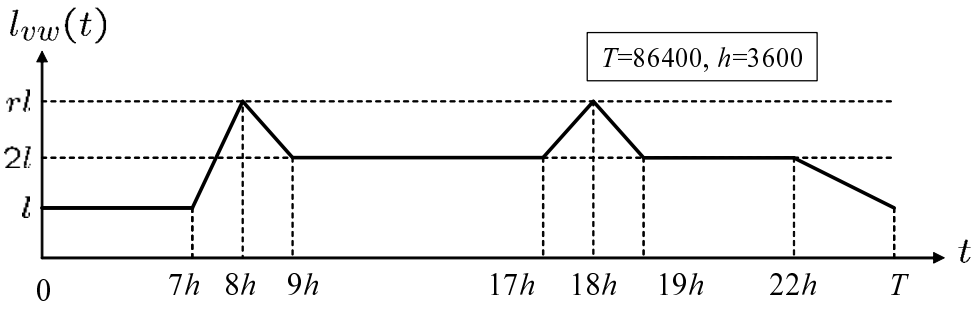


Figure 12: An example of a practical function. The coefficient r depends on l_{vw} to satisfy the FIFO property.

iterations; in each iteration i choose the value of y_i such that $\max\{x_{i-1}+y_{i-1}-x_i,\, l_{vw}\} \leq y_i \leq \min\{y_0+T,\, rl_{vw}\}$. Such y_i satisfies the FIFO property, and we let (x_i, y_i) be the i-th point. In this thesis, we use $k = 8$ and $r = 4$ to compare the computational results with the results with the practical functions.

The practical function is based on a practical traffic models. The number of segments is fixed by 8 and each segment is defined as shown in Figure 12. There are two peaks in a period $[0, T]$, which represents traffic jams in the rush-hour, and at the peak the edge length becomes r times of the base length l_{vw}. The coefficient r is set to 4 if $0 \leq l_{vw} \leq 1800$ and is set to 3 if $1800 < l_{vw} \leq 3600$ to satisfy the FIFO property. When $l_{vw} > 3600$, the edge function is assigned as a constant, $l_{vw}(t) = l_{vw}$. The distribution of edge lengths of each instance is shown in Table 2.

4.3 Experimental results

We report our experimental results in this section. We compare the performance of the generalized Dijkstra algorithm with our time-dependent ALT algorithm with the min-length estimator and the combined estimator. The performance for each instance is evaluated by the average *time efficiency* and the average *search space efficiency* of 200 queries. The measures are defined as follows:

Time efficiency: the average running time of the generalized Dijkstra algorithm divided by that of our time-dependent ALT algorithm.

Search space efficiency: the average number of vertices scanned by the generalized Dijkstra algorithm divided by that scanned by our time-dependent ALT algorithm.

All experiments are run on a PC with a 3.2GHz processor and 2GB RAM. Note that the time efficiency and search space efficiency are machine-independent. The instance, the algorithms and the landmark selection methods are denoted by follows.

Instances

***Name*-P:** the FIFO network with practical edge length functions.

***Name*-R:** the FIFO network with random edge length functions.

Algorithms

GD: the generalized Dijkstra algorithm.

M: the time-dependent ALT algorithm with the min-length estimator.

C-p: the time-dependent ALT algorithm with the combined estimator and p samplings.

We report the results of the time-dependent ALT algorithm for the instance NW-P and NW-R in Table 3, BAY-P and BAY-R in Table 4, S1-P and S1-R in Table 5 and S2-P and S2-R in Table 6. We implement the time-dependent ALT algorithm with 1,4,9 and 16 landmarks and the min-length estimator or the combined estimator with 1,2 and 4 samplings, respectively.

Table 3, 4, 5 and 6 show the result of the time-dependent ALT algorithm with the various number of landmarks, samplings and the various landmark selections. Each table shows the time efficiency in Roman and the search space efficiency in Italics. The running time [s] and the search space [#edge] is shown under the name of the instance for reference. Figure 13 shows the relationship between the number of landmarks and the time efficiency for each instance. For this, the number of samplings is fixed to 2. Figure 14 shows the relationship between the number of samplings and the time efficiency. The number of landmarks is fixed to 9. Figure 15 shows the relationship between the *Dijkstra rank* and the time efficiency. The Dijkstra rank is a measure which used widely. For a query $(s, d, 0)$, the Dijkstra rank of vertex v is the number of vertices in the priority queue before v is reached [9].

At last, we show an example of the reduction of the search space for the instance NW-P. The number of landmarks is 16 and landmarks are selected by the planar selection. We use the combined estimator with 2 samplings. See Figure 16. At this example, the time-dependent ALT algorithm yields the speed-up to 7.4 and reduces the search space about 18.2 times than the generalized Dijkstra algorithm.

Table 3: Results for instance NW. The time efficiency is in Roman and the search space efficiency is in Italics. The numbers under the name of instance are the running time [s] of "GD" in Roman and the search space [#vertex] in Italics.

Instance	Selection	$\|\mathbb{L}\|$	M	C-1	C-2	C-4
NW-P	Farthest	1	0.95	0.90	0.91	0.89
			1.24	*1.28*	*1.29*	*1.29*
1.153		4	2.03	2.30	2.35	2.28
613512			*2.66*	*3.27*	*3.32*	*3.35*
		9	2.67	3.82	3.88	3.85
			3.75	*6.12*	*6.37*	*6.59*
		16	2.77	3.54	3.60	3.42
			4.06	*7.15*	*7.55*	*7.77*
	Planar	1	1.05	0.99	0.99	0.97
			1.41	*1.48*	*1.49*	*1.50*
		4	2.01	2.56	2.59	2.56
			2.70	*3.68*	*3.73*	*3.75*
		9	2.73	3.75	3.78	3.70
			3.76	*6.09*	*6.27*	*6.39*
		16	2.89	3.87	3.86	3.73
			4.16	*7.71*	*8.06*	*8.43*
	Grid	1	1.04	0.96	0.97	0.93
			1.38	*1.44*	*1.44*	*1.44*
		4	2.12	2.48	2.50	2.44
			2.79	*3.49*	*3.56*	*3.59*
		9	2.38	2.86	2.89	2.81
			3.21	*4.66*	*4.83*	*4.90*
		16	2.75	3.34	3.35	3.18
			3.89	*6.66*	*6.94*	*7.15*

Instance	Selection	$\|\mathbb{L}\|$	M	C-1	C-2	C-4
NW-R	Farthest	1	0.84	1.01	1.02	1.01
			1.20	*1.48*	*1.49*	*1.49*
1.166		4	1.21	2.76	2.74	2.71
615602			*1.72*	*3.99*	*3.99*	*3.99*
		9	1.28	3.58	3.51	3.39
			1.89	*5.99*	*6.05*	*6.07*
		16	1.26	3.61	3.49	3.26
			1.93	*7.59*	*7.63*	*7.66*
	Planar	1	1.89	0.98	0.95	0.97
			1.89	*1.47*	*1.47*	*1.47*
		4	1.14	2.06	2.04	2.04
			1.61	*3.06*	*3.06*	*3.06*
		9	1.27	3.62	3.59	3.36
			1.87	*6.13*	*6.13*	*6.14*
		16	1.27	3.54	3.40	3.18
			1.95	*7.45*	*7.46*	*7.46*
	Grid	1	0.91	0.97	0.98	0.96
			1.23	*1.44*	*1.44*	*1.44*
		4	1.18	2.38	2.36	2.34
			1.67	*3.48*	*3.49*	*3.49*
		9	1.24	2.73	2.79	2.65
			1.79	*4.74*	*4.75*	*4.76*
		16	1.23	3.15	3.03	1.55
			1.88	*6.60*	*6.62*	*6.62*

Table 4: Results for instance BAY. The time efficiency is in Roman and the search space efficiency is in Italics. The numbers under the name of instance are the running time [s] of "GD" in Roman and the search space [#vertex] in Italics.

Instance	Selection	$\|\mathbb{L}\|$	M	C-1	C-2	C-4
BAY-P	Farthest	1	0.91	0.89	0.90	0.90
			1.31	*1.41*	*1.43*	*1.44*
0.233		4	1.50	1.84	1.90	1.98
154419			*2.22*	*3.02*	*3.21*	*3.37*
		9	1.62	1.97	2.14	2.21
			2.49	*3.85*	*4.26*	*4.61*
		16	1.54	1.99	2.15	2.17
			2.67	*4.82*	*5.42*	*5.85*
	Planar	1	0.88	0.83	0.85	0.85
			1.28	*1.33*	*1.35*	*1.35*
		4	1.34	1.37	1.46	1.47
			1.95	*2.36*	*2.51*	*2.59*
		9	1.60	1.95	2.04	2.03
			2.45	*3.80*	*4.11*	*4.28*
		16	1.63	1.86	2.00	1.99
			2.63	*4.48*	*5.02*	*5.34*
	Grid	1	0.92	0.84	0.86	0.86
			1.28	*1.34*	*1.36*	*1.37*
		4	1.30	1.43	1.47	1.53
			1.88	*2.37*	*2.52*	*2.59*
		9	1.44	1.51	1.55	1.57
			2.16	*2.97*	*3.16*	*3.30*
		16	1.56	1.89	2.00	1.98
			2.54	*4.58*	*5.04*	*5.36*

Instance	Selection	$\|\mathbb{L}\|$	M	C-1	C-2	C-4
BAY-R	Farthest	1	0.88	0.94	0.94	0.93
			1.20	*1.45*	*1.45*	*1.45*
0.236		4	1.09	1.75	1.76	1.77
153640			*1.57*	*2.93*	*2.96*	*2.98*
		9	1.09	1.86	1.86	1.85
			1.64	*3.59*	*3.70*	*3.81*
		16	1.04	1.86	1.87	1.75
			1.70	*4.50*	*4.64*	*4.71*
	Planar	1	0.84	0.88	0.87	0.85
			1.17	*1.34*	*1.34*	*1.34*
		4	0.92	1.22	1.20	1.19
			1.37	*2.01*	*2.02*	*2.03*
		9	1.10	1.98	1.97	1.87
			1.66	*3.84*	*3.91*	*3.92*
		16	1.07	1.78	1.71	1.59
			1.69	*4.25*	*4.29*	*4.32*
	Grid	1	0.84	0.90	0.91	0.90
			1.17	*1.35*	*1.36*	*1.36*
		4	0.96	1.36	1.36	1.36
			1.39	*2.23*	*2.28*	*2.30*
		9	1.02	1.44	1.42	1.39
			1.54	*2.80*	*2.87*	*2.89*
		16	1.03	1.59	1.57	1.42
			1.64	*3.84*	*3.96*	*4.06*

Table 5: Results for instance S1. The time efficiency is in Roman and the search space efficiency is in Italics. The numbers under the name of instance are the running time [s] of "GD" in Roman and the search space [#vertex] in Italics.

Instance	Selection	$\|\mathbb{L}\|$	M	C-1	C-2	C-4
S1-P	Farthest	1	0.88	0.78	0.78	0.77
			1.21	*1.23*	*1.24*	*1.25*
0.011		4	1.45	1.23	1.33	1.39
9093			*2.15*	*2.36*	*2.54*	*2.70*
		9	1.85	1.45	1.56	1.62
			2.86	*3.30*	*3.66*	*4.05*
		16	1.69	1.17	1.28	1.29
			2.79	*3.42*	*3.88*	*4.20*
	Planar	1	0.89	0.77	0.79	0.77
			1.21	*1.22*	*1.23*	*1.24*
		4	1.39	1.16	1.20	1.20
			2.05	*2.21*	*2.32*	*2.43*
		9	1.73	1.36	1.46	1.50
			2.69	*3.12*	*3.45*	*3.74*
		16	1.89	1.31	1.43	1.50
			3.12	*3.78*	*4.34*	*4.83*
	Grid	1	0.87	0.77	0.77	0.77
			1.20	*1.21*	*1.23*	*1.24*
		4	1.49	1.24	1.28	1.34
			2.16	*2.34*	*2.47*	*2.61*
		9	1.64	1.25	1.37	1.42
			2.52	*2.87*	*3.21*	*3.44*
		16	1.68	0.84	1.27	1.28
			2.74	*3.34*	*3.77*	*4.16*

Instance	Selection	$\|\mathbb{L}\|$	M	C-1	C-2	C-4
S1-R	Farthest	1	0.85	0.95	0.96	0.97
			1.15	*1.44*	*1.46*	*1.49*
0.011		4	1.01	1.32	1.35	1.31
9078			*1.48*	*2.42*	*2.52*	*2.52*
		9	1.01	1.35	1.35	1.28
			1.53	*2.99*	*3.07*	*3.14*
		16	1.01	1.30	1.31	1.22
			1.62	*3.61*	*3.83*	*3.99*
	Planar	1	0.82	0.79	0.79	0.76
			1.11	*1.20*	*1.20*	*1.20*
		4	0.97	1.06	1.09	1.04
			1.41	*1.96*	*2.01*	*2.01*
		9	1.03	1.28	1.26	1.23
			1.55	*2.85*	*2.97*	*3.00*
		16	1.00	1.19	1.18	1.10
			1.61	*3.29*	*3.42*	*3.45*
	Grid	1	0.82	0.77	0.77	0.76
			1.11	*1.19*	*1.19*	*1.19*
		4	0.97	1.26	1.26	1.26
			1.42	*2.31*	*2.36*	*2.40*
		9	0.97	1.22	1.28	1.26
			1.47	*2.73*	*2.93*	*3.01*
		16	0.96	1.11	1.15	1.12
			1.54	*3.15*	*3.39*	*3.56*

Table 6: Results for instance S2. The time efficiency is in Roman and the search space efficiency is in Italics. The numbers under the name of instance are the running time [s] of "GD" in Roman and the search space [#vertex] in Italics.

Instance	Selection	$\|\mathbb{L}\|$	M	C-1	C-2	C-4
S2-P	Farthest	1	1.01	0.87	0.90	0.89
			1.28	*1.30*	*1.31*	*1.32*
0.002		4	1.49	1.16	1.20	1.20
1973			*2.04*	*2.13*	*2.20*	*2.30*
		9	1.62	1.15	1.19	1.20
			2.40	*2.62*	*2.78*	*2.99*
		16	1.68	1.02	1.08	1.03
			2.65	*2.98*	*3.30*	*3.62*
	Planar	1	1.03	0.87	0.90	0.89
			1.30	*1.31*	*1.33*	*1.35*
		4	1.41	1.08	1.10	1.10
			1.96	*2.01*	*2.06*	*2.13*
		9	1.56	1.11	1.10	1.15
			2.29	*2.47*	*2.59*	*2.80*
		16	1.66	1.03	1.06	1.07
			2.55	*2.94*	*3.18*	*3.51*
	Grid	1	0.94	0.83	0.82	0.80
			1.20	*1.21*	*1.22*	*1.23*
		4	1.54	1.22	1.29	1.32
			2.10	*2.24*	*2.37*	*2.50*
		9	1.56	1.08	1.12	1.11
			2.24	*2.44*	*2.56*	*2.75*
		16	1.71	1.09	1.14	1.20
			2.68	*3.10*	*3.45*	*3.89*

Instance	Selection	$\|\mathbb{L}\|$	M	C-1	C-2	C-4
S2-R	Farthest	1	0.88	0.91	0.95	1.00
			1.18	*1.36*	*1.42*	*1.50*
0.002		4	1.06	1.05	1.15	1.16
1972			*1.52*	*1.98*	*2.13*	*2.30*
		9	1.07	1.10	1.17	1.18
			1.63	*2.52*	*2.77*	*3.01*
		16	1.02	0.93	1.07	1.11
			1.65	*2.76*	*3.35*	*3.74*
	Planar	1	0.87	0.83	0.80	0.83
			1.15	*1.23*	*1.23*	*1.23*
		4	1.00	0.94	0.95	0.92
			1.43	*1.74*	*1.75*	*1.79*
		9	1.00	0.90	0.88	0.90
			1.48	*2.00*	*2.11*	*2.26*
		16	1.01	0.86	0.92	0.94
			1.62	*2.53*	*2.84*	*3.13*
	Grid	1	0.85	0.77	0.77	0.77
			1.13	*1.16*	*1.16*	*1.19*
		4	1.01	1.08	1.12	1.19
			1.45	*1.93*	*2.10*	*2.30*
		9	1.03	0.95	1.04	1.01
			1.53	*2.17*	*2.45*	*1.62*
		16	0.92	1.02	1.06	1.06
			2.70	*3.11*	*3.52*	*2.66*

4.4 Remarks

In this section, we investigate the experimental results in details.

First, we compare the edge length function structures. Fixing the number of landmarks and samplings and the landmark selection, the comparison of the time efficiency shows that the result for the practical function outperforms the others where many landmarks are given. It shows that the time-dependent ALT algorithm is sensitive to the function structures. By definition of the estimators, it is observed that the function close to static gives a larger bounded estimator. The practical function is expected to be closer to static than the random function since there are few changes of the length in the practical function except for two peaks. By the way, it is difficult to say which is better where few landmarks are given. With a few landmarks, each estimator is not large enough so that the performance is not relate to the type of functions.

Next we study the relationship between the time efficiency and the number of landmarks. By Lemma 3, a larger bounded estimator surely reduces the search space and more landmarks are expected to have the higher time efficiency. Seeing Table 3, 4, 5 and 6, it can be observed that more landmarks surely yields the higher search space efficiency. However, there is the trade-off between the number of landmarks and the time efficiency. See Figure 13. In our study, the best number of landmarks is 9, though Goldberg and Harrelson reported the best number of landmarks is 16 for many instances [16]. The difference between theirs and ours is considered to be derived from the overhead of the samplings. On the other hand, it is observed that not all algorithms outperforms the generalized Dijkstra algorithm. The generalized Dijkstra algorithm is enough fast in small networks with about 30,000 vertices so that the overhead of the time-dependent ALT algorithm goes over the search space efficiency.

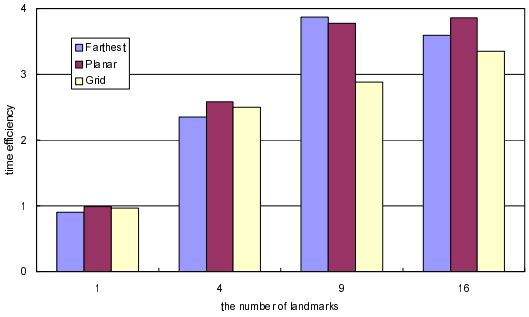
The landmark selection is as important as the number of landmarks. As described in Section 4.1, the landmark L such that the vertex v is approximately on the way L and d seems to contribute a strong estimation and it is considered to prefer to be far from each other landmark. Due to Goldberg and Harrelson [16], the planar selection is the best in their study. The results in Table 3, 4, 5 and 6 show our algorithm also follows it.

We give data for the choice of 9 landmarks in Figure 14. Consider the number of sampling of the min-length estimator is 0. Similarly to the comparison of the number of landmarks, more samplings surely reduce the search space. (See Table 3, 4, 5 and 6). Our time-dependent ALT algorithm has to use $O((p+1)|\mathbb{L}|)$ time additionally, where p is the number of samplings and $\mathbb{L}$ is a set of landmarks, for the computation of the time-dependent estimator which is not needed for the generalized Dijkstra algorithm. Then our results show the 4 samplings is the best for the time efficiency.

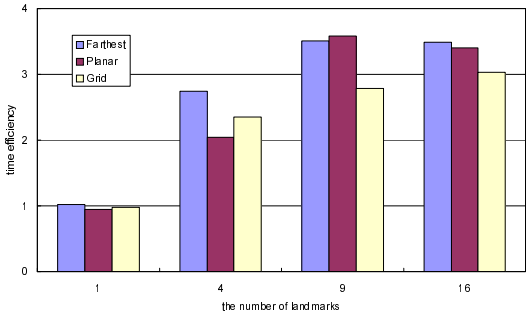
Let us summarize the results. We confirm the function close to static averagely gives a larger bounded estimation than the random function. We obtain the results that strongly depends on the number of landmarks and samplings and our study finds the best number of landmarks and samplings are 9 and 2, respectively.

Furthermore, we give the result of the relationship between the *Dijkstra rank* and the time efficiency. The Dijkstra rank is a measure which used widely. For a query $(s, d, 0)$, the Dijkstra rank of vertex v is the number of vertices in the priority queue before v is reached [9]. See Figure 15 which represents the relationship between the average Dijkstra ranks and the time efficiency. The number of landmark is fixed to 9 and the selection is the

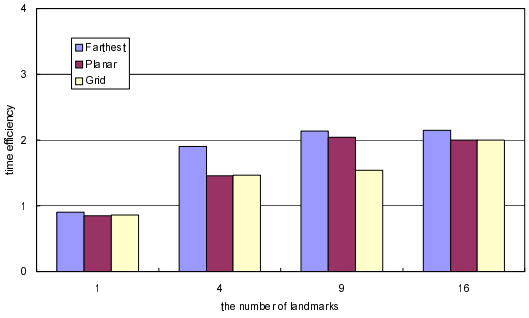
planar. Figure 15 shows the performance of the algorithm is also depends on the Dijkstra ranks. Focusing on the results of the min-length and C-4 for the instance NW-P, we can find an interest feature. For the high Dijkstra ranks, we obtain the highest speed-ups up to 4 by the combined estimator, however, for the low Dijkstra ranks, we sometimes obtain the highest speed-ups by the min-length estimator. This result also says, as described above, that for the low Dijkstra ranks (for the small networks), too many landmarks and samplings do not contribute the time efficiency.



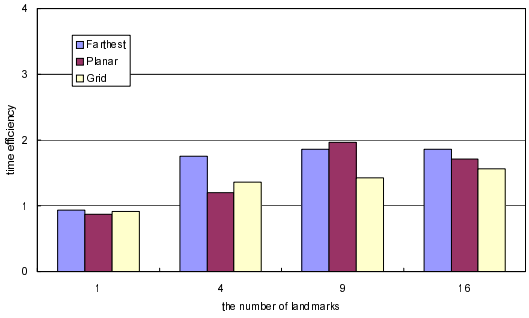
For the instance NW-P.



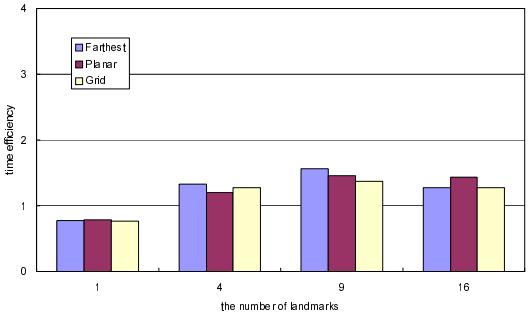
For the instance NW-R.



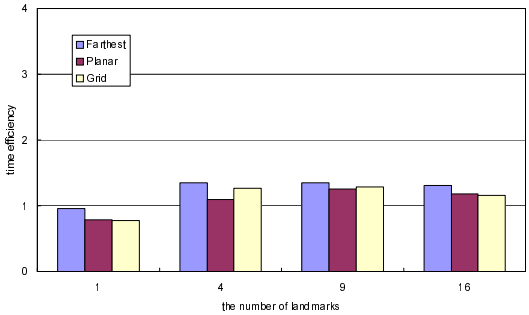
For the instance BAY-P.



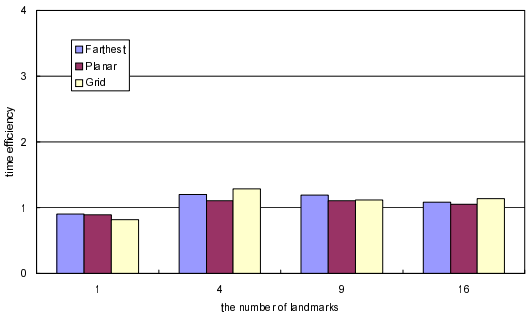
For the instance BAY-R.



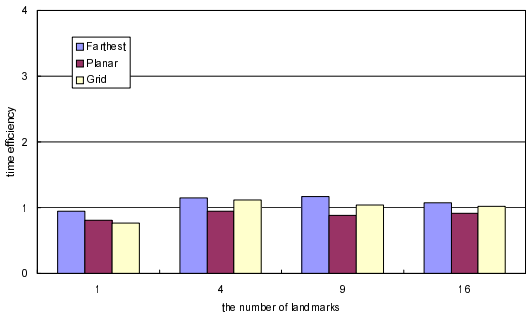
For the instance S1-P.



For the instance S1-R.

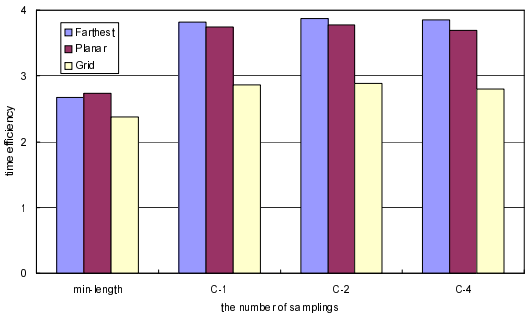


For the instance S2-P.

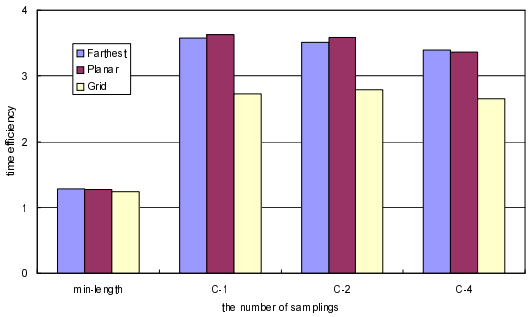


For the instance S2-R.

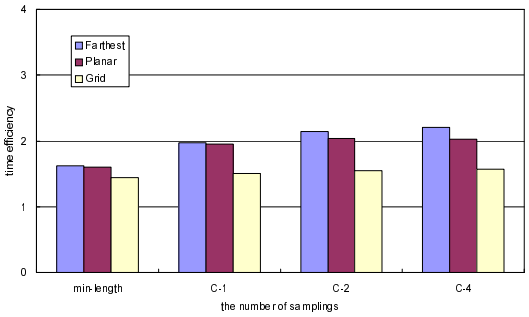
Figure 13: Relationship between the time efficiency and the number of landmarks. The number of samplings is fixed to 2. The x-axis represents the number of landmarks and the y-axis represents the time efficiency. The left, the middle and the right bar represent the results of the farthest, the planar and the grid landmark selections, respectively.



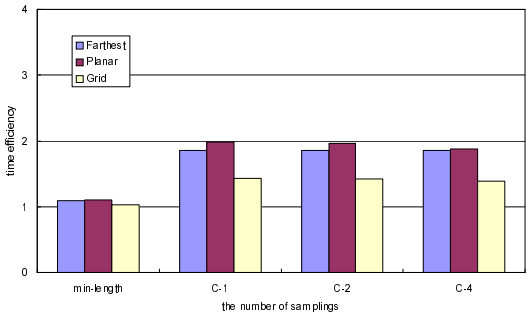
For the instance NW-P.



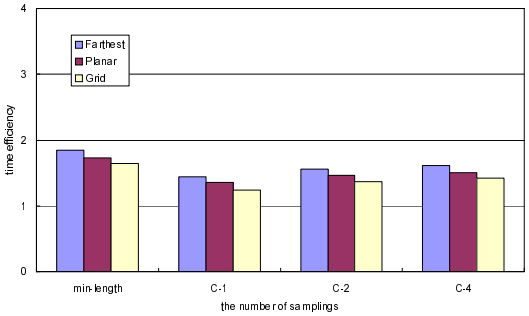
For the instance NW-R.



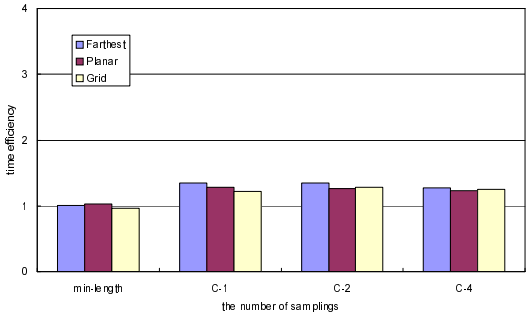
For the instance BAY-P.



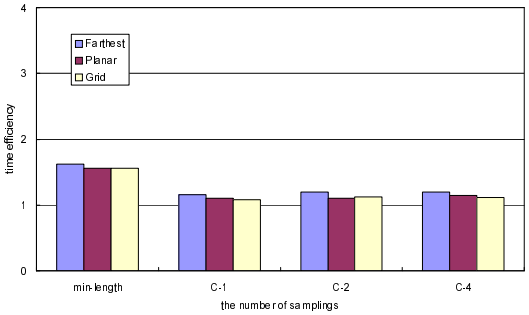
For the instance BAY-R.



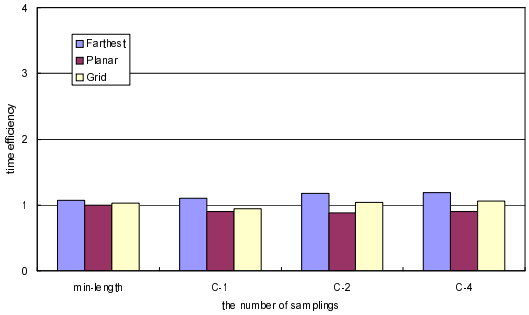
For the instance S1-P.



For the instance S1-R.

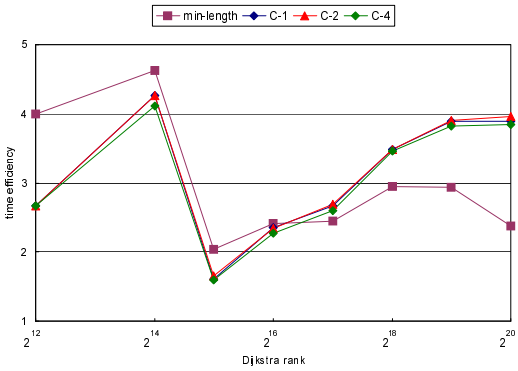


For the instance S2-P.

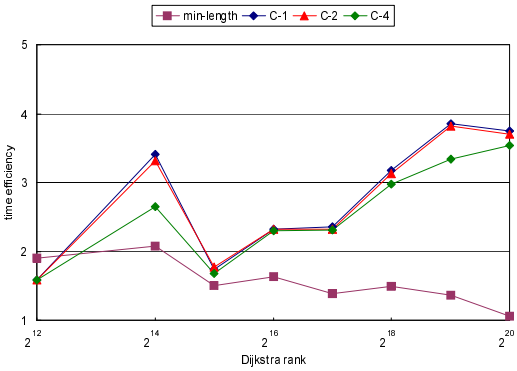


For the instance S2-R.

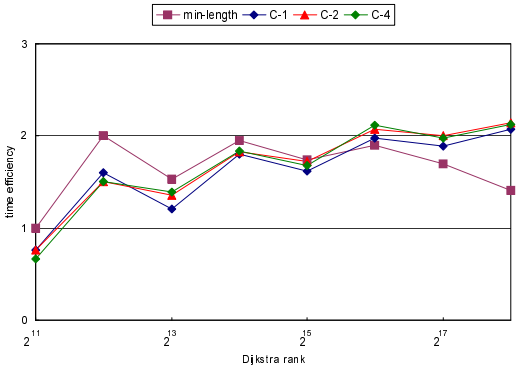
Figure 14: Relationship between the time efficiency and the number of samplings. The number of landmarks is fixed to 9. The x-axis represents the algorithm with min-length, C-1, C-2 and C-4. The y-axis represents the time efficiency. The left, the middle and the right bar represent the results of the farthest, the planar and the grid landmark selections, respectively.



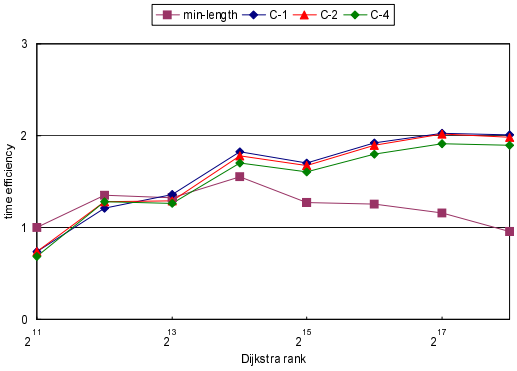
For the instance NW-P.



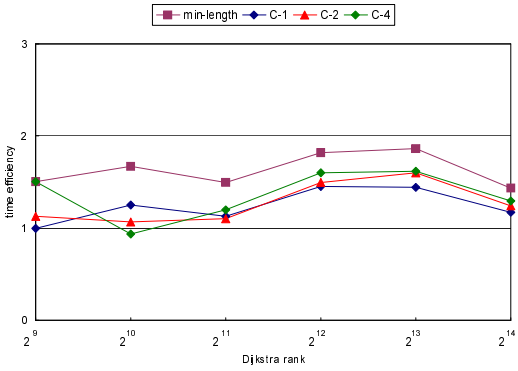
For the instance NW-R.



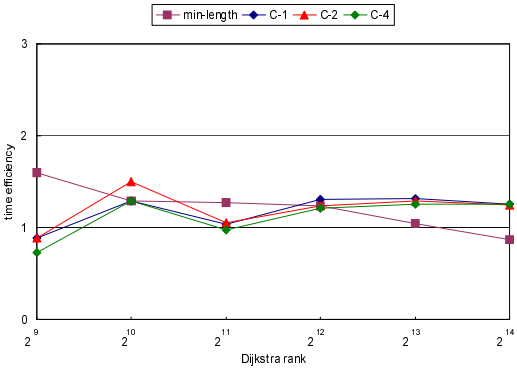
For the instance BAY-P.



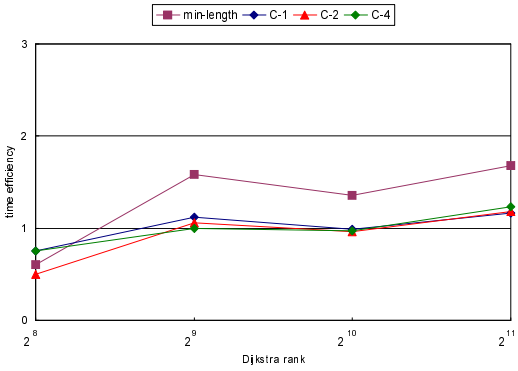
For the instance BAY-R.



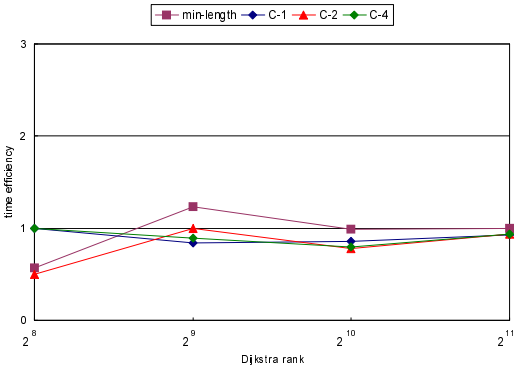
For the instance S1-P.



For the instance S1-R.



For the instance S2-P.



For the instance S2-R.

Figure 15: Relationship between the time efficiency and the Dijkstra rank. The x-axis represents the average Dijkstra rank and the y-axis represents the time efficiency. To compare, we show min-length, C-1, C-2 and C-4, respectively.

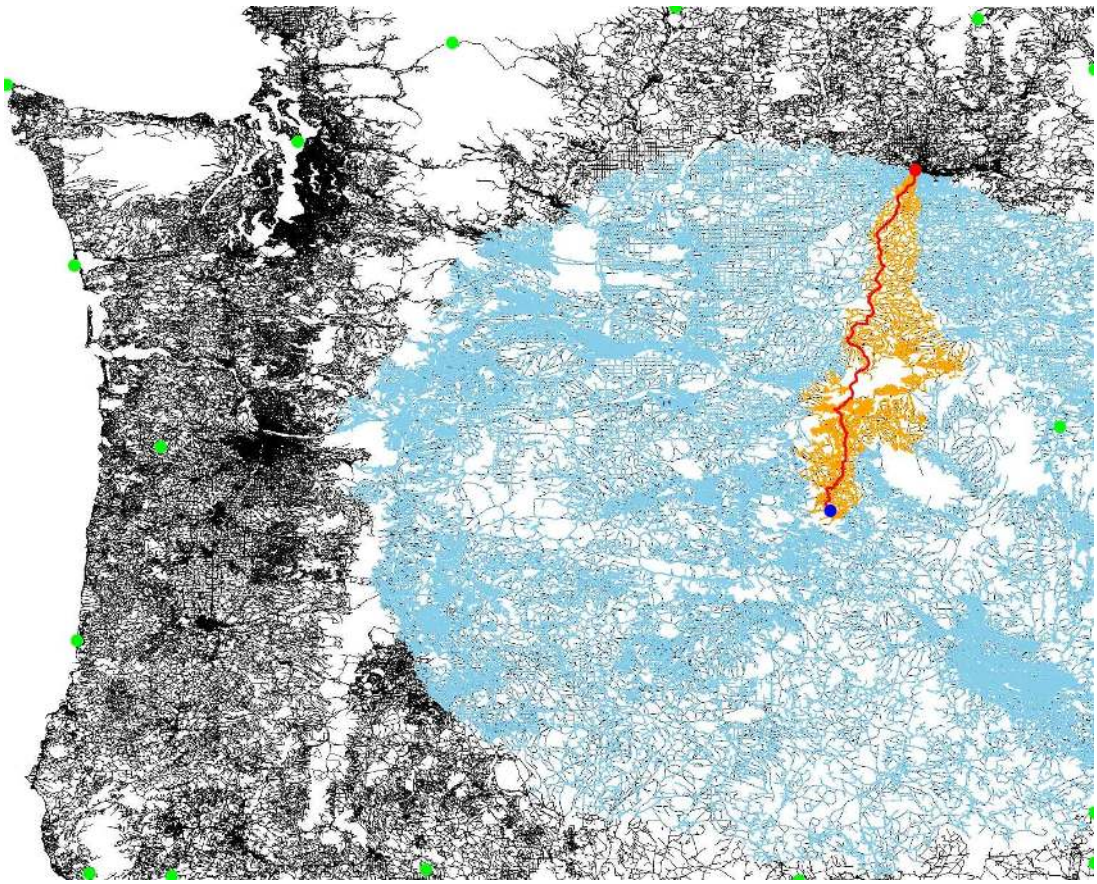


Figure 16: An example of the result of the time-dependent ALT algorithm. The "blue" circle is the source s and the "red" circle is the target d. The "red" path is the $(s, d, 0)$-shortest path and the Dijkstra rank of the query $(s, d, 0)$ is $377096 \approx 2^{18.5}$. The set of green circles is a set of landmarks selected by the planar selection. The number of samplings is 2. The set of "skyblue" vertices represents the search space of the time-dependent ALT algorithm. The union of the "skyblue" vertices and "orange" vertices represents the search space of the generalized Dijkstra algorithm. The time efficiency is about 7.4, the search space efficiency is about 18.2.

5 Conclusions and future work

In this thesis, we considered the time-dependent shortest path problem and proposed a new algorithm called the time-dependent ALT algorithm based on the ALT algorithm [16]. The most significant point is that our algorithm does not require to save all shortest paths of all starting times which all of the obvious extensions of previous algorithms for static networks requires. We proposed the time-dependent A^* algorithm and showed that the algorithm guarantees the optimality if a given estimator is a *metric FIFO* estimator and that the algorithm with a bounded and non-negative estimator scans no more vertices than the generalized Dijkstra algorithm. We also gave two time-dependent estimators called the *min-length estimator* and the *sample-time estimator* which is a metric FIFO, bounded and non-negative estimator. Also we implemented the time-dependent ALT algorithm and compared it with the generalized Dijkstra algorithm [13, 20, 25, 33] with respect to the running time and the search space. We obtained about 4 times speed-ups than the generalized Dijkstra algorithm with a small amount of preprocessing cost. It is revealed that more landmarks and more samplings yield higher search space efficiencies, but neither too many landmarks nor too many samplings contributes the speed-ups. Our study showed that 9 landmarks and 2 samplings by farthest selection achieves the best performance for the instance with about 1,200,000 vertices. However, for some small instances, our algorithm becomes worse in the running times. It is considered, for a pair of a source and target which are close to each other, the generalized Dijkstra algorithm is fast enough. Unfortunately, our algorithm does not run twice faster even if the search space becomes half. Looking at the value of the time efficiency divided by the search space efficiency, it is well apparent. Reducing the overhead is one of the future work.

Future work

Though we tried to apply other preprocessing algorithms to the time-dependent shortest path problem, only the ALT algorithm can be extended. Our attempt for the overlay graph algorithm [9, 22, 23, 37, 38] defined a time-dependent overlay graph $G' = (V', E')$, where $V' \subseteq V$. For the overlay graph $G' = (V', E')$, an edge $(v, w) \in E' \subseteq V' \times V'$ exists if and only if there exists some time t such that no inner vertex of any (v, w, t)-shortest paths belonging to V'. However, this approach also requires shortest paths between all $(v, w) \in V' \times V'$ for all starting time t, it is impossible.

The reach-based approach by Gutman [19] is well compatible with the ALT algorithm in static networks. Extending the approach to time-dependent is expected to reduce the search space dramatically. Similarly to the any other algorithm for static networks, however, no good extensions has been proposed. The difficulty of extending is how to prepare time-dependent data at low space.

Focusing on the experimental results where the larger instance performs better, the following algorithm can be considered. Prepare landmarks as much as possible to load into memory, but use only a subset of them which can be selected by the information of the source and the target. Note that the algorithm cannot change the set of landmarks during the calculating procedure to keep the optimality.

References

[1] R. K. Ahuja, J. B. Orlin, S. Pallottino, and M. G. Scutella. Minimum time and minimum cost path problems in street networks with traffic lights. Technical Report TR-99-13, Universitá di Pisa Diartimento di Informatica, 1999.

[2] 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*. SIAM, 2007.

[3] Richard Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16:87–90, 1958.

[4] Jean François Bérubé, Jean Yves Potvin, and Jean G. Vaucher. Time-dependent shortest paths through a fixed sequence of nodes: application to a travel planning problem. *Computers & OR*, 33:1838–1856, 2006.

[5] Gerth Stølting Brodal and Riko Jacob. Time-dependent networks as models to achieve fast exact time-table queries. *Electr. Notes Theor. Comput. Sci.*, 92:3–15, 2004.

[6] Kenneth L. Cooke and Eric Halsey. The shortest route through a network with time-dependent internodal transit. *J. Math. Anal. Appl.*, 14:493–498, 1966.

[7] Brian C. Dean. Continuous-time dynamic shortest path algorithms. Master's thesis, Massachusetts Institute of Technology, 1999.

[8] Brian C. Dean. Shortest paths in FIFO time-dependent networks: theory and algorithms. Technical report, Massachusetts Institute of Technology, 2004.

[9] Daniel Delling, Martin Holzer, Kirill Muller, Frank Schulz, and Dorothea Wagner. High-performance multi-level graphs. The 9th DIMACS Implementation Challenge: Shortest Paths.

[10] Daniel Delling, Peter Sanders, Dominik Schultes, and Dorothea Wagner. Highway hierarchies star. In the 9th DIMACS Implementation Challenge, 2006.

[11] Daniel Delling and Dorothea Wagner. Landmark-based routing in dynamic graphs. In Camil Demetrescu, editor, *WEA*, volume 4525 of *LNCS*, pp. 52–65. Springer, 2007.

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

[13] Stuart E. Dreyfus. An appraisal of some shortest-path algorithms. *Operations Research*, 17(3):395–412, 1969.

[14] Michael L. Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34(3):596–615, 1987.

[15] D. Frigioni, A. Marchetti Spaccamela, and U. Nanni. Semidynamic algorithms for maintaining single-source shortest path trees. *Algorithmica*, 22(3):250–274, 1998.

[16] Andrew V. Goldberg and Chris Harrelson. Computing the shortest path: A^* search meets graph theory. In *SODA 2005*, pp. 156–165. SIAM, 2005.

[17] Andrew V. Goldberg, Haim Kaplan, and Renato F. Werneck. Reach for A^*: Efficient point-to-point shortest path algorithms. Technical Report MSR-TR-2005-132, Microsoft Research, Microsoft Corporation, 2005.

[18] Andrew V. Goldberg, Haim Kaplan, and Renato Fonseca F. Werneck. Better landmarks within reach. In Camil Demetrescu, editor, *WEA*, volume 4525 of *LNCS*, pp. 38–51. Springer, 2007.

[19] Ronald J. Gutman. Reach-based routing: A new approach to shortest path algorithms optimized for road networks. In Lars Arge, Giuseppe F. Italiano, and Robert Sedgewick, editors, *ALENEX/ANALC*, pp. 100–111. SIAM, 2004.

[20] Haifa J. Halpern. Shortest route with time dependent length of edges and limited delay possibilities in nodes. *Operations Research*, 21:117–124, 1977.

[21] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions Systems Science and Cybernetics*, 4(2):100–107, 1968.

[22] Martin Holzer, Grigorios Prasinos, Frank Schulz, Dorothea Wagner, and Christos D. Zaroliagis. Engineering planar separator algorithms. In Gerth Stølting Brodal and Stefano Leonardi, editors, *ESA 2005*, volume 3669 of *LNCS*, pp. 628–639. Springer, 2005.

[23] Martin Holzer, Frank Schulz, and Dorothea Wagner. Engineering multi-level overlay graphs for shortest-path queries. In *ALENEX 2006*. SIAM, 2006.

[24] Martin Holzer, Frank Schulz, Dorothea Wagner, and Thomas Willhalm. Combining speed-up techniques for shortest-path computations. *ACM Journal of Experimental Algorithms*, 10, 2005.

[25] D.E. Kaufman and R.L. Smith. Fastest paths in time-dependent networks for intelligent vehicle-highway systems application. *Journal of Intelligent Transportation Systems*, 1(1):1–11, 1993.

[26] Philip Klein, Satish Rao, Monika Rauch, and Sairam Subramanian. Faster shortest-path algorithms for planar graphs. In *STOC*, pp. 27–37. ACM Press, 1994.

[27] Sebastian Knopp, Peter Sanders, Dominik Schultes, Frank Schulz, and Dorothea Wagner. Computing many-to-many shortest paths using highway hierarchies. In *ALENEX 2007*. SIAM, 2007.

[28] Ekkehard Köhler, Rolf H. Möhring, and Heiko Schilling. Acceleration of shortest path and constrained shortest path computation. In Sotiris E. Nikoletseas, editor, *WEA 2005*, volume 3503 of *LNCS*, pp. 126–138. Springer, 2005.

[29] Bernhard Korte and Jens Vygen. Combinatorial Optimization: Theory and Algorithms. Springer, 2002.

[30] Michael Luby and Prabhakar Ragde. A bidirectional shortest-path algorithm with good average-case behavior. *Algorithmica*, 4(4):551–567, 1989.

[31] Rolf H. Möhring, Heiko Schilling, Birk Schütz, Dorothea Wagner, and Thomas Willhalm. Partitioning graphs to speedup Dijkstra's algorithm. *ACM Journal of Experimental Algorithms*, 11, 2006.

[32] Ariel Orda and Raphael Rom. Distributed shortest-path protocols for time-dependent networks. In *ICCC*, pp. 439–445, 1988.

[33] Ariel Orda and Raphael Rom. Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length. *J. ACM*, 37(3):607–625, 1990.

[34] Peter Sanders and Dominik Schultes. Highway hierarchies hasten exact shortest path queries. In Gerth Stølting Brodal and Stefano Leonardi, editors, *ESA 2005*, volume 3669 of *LNCS*, pp. 568–579. Springer, 2005.

[35] Peter Sanders and Dominik Schultes. Engineering highway hierarchies. In Yossi Azar and Thomas Erlebach, editors, *ESA 2006*, volume 4168 of *LNCS*, pp. 804–816. Springer, 2006.

[36] Peter Sanders and Dominik Schultes. Engineering fast route planning algorithms. In Camil Demetrescu, editor, *WEA*, volume 4525 of *LNCS*, pp. 23–36. Springer, 2007.

[37] Frank Schults. Timetable Information and Shortest Paths. PhD thesis, University Karlsruhe, 2005.

[38] Frank Schulz, Dorothea Wagner, and Christos D. Zaroliagis. Using multi-level graphs for timetable information in railway systems. In David M. Mount and Clifford Stein, editors, *ALENEX 2002*, volume 2409 of *LNCS*, pp. 43–59. Springer, 2002.

[39] Hanif D. Sherali, Antoine G. Hobeika, and Sasikul Kangwalklai. Time-dependent, label-constrained shortest path problems with applications. *Transportation Science*, 37(3):278–293, 2003.

[40] Hanif D. Sherali, Kaan Ozbay, and Shivaram Subramanian. The time-dependent shortest pair of disjoint paths problem: Complexity, models, and algorithms. *Networks*, 31(4):259–272, 1998.

[41] The 9th DIMACS Implementation Challenge: Shortest Paths. http://www.dis.uniroma1.it/~challenge9/.

[42] Dorothea Wagner and Thomas Willhalm. Geometric speed-up techniques for finding shortest paths in large sparse graphs. In Giuseppe Di Battista and Uri Zwick, editors, *ESA 2003*, volume 2832 of *LNCS*, pp. 776–787. Springer, 2003.

[43] Dorothea Wagner and Thomas Willhalm. Speed-up techniques for shortest-path computations. In Wolfgang Thomas and Pascal Weil, editors, *STACS 2007*, volume 4393 of *LNCS*, pp. 23–36. Springer, 2007.

[44] Dorothea Wagner, Thomas Willhalm, and Christos D. Zaroliagis. Dynamic shortest paths containers. *Electronic Notes Theoretical Computer Science*, 92:65–84, 2004.

[45] Dorothea Wagner, Thomas Willhalm, and Christos D. Zaroliagis. Geometric containers for efficient shortest-path computation. *ACM Journal of Experimental Algorithms*, 10, 2005.