An efficient implementation of the ALT algorithm for the Time-Dependent Shortest Path Problem

Genaro PEQUE Jr[1], Junji URATA[2]and Takamasa IRYO[3]

[1]Non-Member of JSCE, Dept. of Civil Eng., Kobe University
(1-1, Rokkodai-cho, Nada, Kobe, Hyogo, 657-8501, Japan)
E-mail: gpequejr@panda.kobe-u.ac.jp
[2]Member of JSCE, Dept. of Civil Eng., Kobe University
(1-1, Rokkodai-cho, Nada, Kobe, Hyogo, 657-8501, Japan)
E-mail: urata@person.kobe-u.ac.jp
[3]Member of JSCE, Professor, Dept. of Civil Eng., Kobe University
(1-1, Rokkodai-cho, Nada, Kobe, Hyogo, 657-8501, Japan)
E-mail: iryo@kobe-u.ac.jp

Although ALT algorithms have been successfully applied to time-independent transportation networks, applying it to time-dependent transportation networks is not straightforward. The difficulty arises when a driver has to recalculate the shortest path to his/her destination at the time of his/her departure by considering the effect of other vehicles on the travel time. While this doesn't pose a substantial negative computational effect when the number of drivers is small, this is crucial when dealing with large-scale time-dependent transportation networks with millions of vehicles and a very large time horizon. This is one of the most computationally demanding components when simulating large-scale time-dependent transportation networks and is called the time-dependent shortest path (TDSP) problem.

In this paper we are motivated in efficiently implementing and improving the time-dependent ALT algorithm using the C++ programming language that will be used in the repeated calculation of the shortest path of each driver to their destination in a large-scale time-dependent transportation network. Results show that search space reduction using a probabilistic variant of the landmark selection strategy $avoid$[16)], a proposed scoring mechanism for $maxcover$, an added landmark selection constraint based on a theorem on trees[14)] and careful implementation of the ALT algorithm in C++ plays a big role in the reduction of computation time when solving the TDSP problem.

Key Words: *ALT algorithm, Time-dependent shortest path problem, large-scale simulation, Time-dependent networks, c++ programming language*

1. Introduction

Transportation networks are normally represented using link-weighted graphs with non-negative link weights (e.g. the travel time on a road). Given this link-weighted graph, the task of finding the shortest path between two nodes (e.g. origin and destination nodes) is known as the (classical) shortest path problem and is usually solved using the well-known Dijkstra's algorithm[12)]. Many algorithms such as the bidirectional Dijkstra[22)], A* search[18)] and its extensions[15),16),19)] have been proposed which accelerated the shortest path calculations.

In this paper, we consider a generalization of the link-weighted graph where links have time-dependent weights that is a function of the departure time at the starting node. This is formally called a *time-dependent network*, and the shortest path problem in a time-dependent network is called the *time-dependent shortest path* (TDSP) problem. Work on the TDSP problem was first done by Cooke and Halsey[5)] where they proposed a Dynamic Programming algorithm[4)] (not polynomial-time) which can only treat integer time values. Dreyfus[13)] proposed a polynomial-time algorithm which is a straightforward generalization of Dijkstra's algorithm. However, it was shown to work correctly only if the time-dependent network satisfies the *first-in, first-out* (FIFO) property[17),20),26)]. Subsequently, the TDSP problem in FIFO networks have been solved using algorithms such as Dijkstra's algorithm[6)], its generalization known as the A* (A-star) search[28)] and its extensions[24),25),28)].

In the simulation of large-scale time-dependent transportation networks, the repeated calculation of the TDSP of each driver to their destination is one of its most computationally demanding components.

Hence, shortest path algorithms are not only being improved but also the preprocessing techniques used to store additional information about the network to accelerate the shortest path calculation. Delling and Wagner[11] identified the basic ingredients that all existing high-performance speed-up techniques for route planning in time-dependent networks rely on. These are Dijkstra's algorithm implemented as a label-correcting algorithm[6], A* search using preprocessed landmarks (ALT)[25] and its variants[24], Arc-Flags[7),11], and contraction[8]. From the list mentioned, we focus on the preprocessing of landmarks used in the A* search because of its easy adaption to time-dependent scenarios and its robustness to the input. Bauer et al.[3] have shown that most of the preprocessing based speed-up techniques have some degrees of freedom which are NP-hard to determine optimally, and are therefore heuristically determined in practice. This also applies to the ALT algorithm, for which it is shown that selecting a set of landmarks that minimizes the expected "search space" (i.e. the set of nodes that have to be "explored" before the solution is found) for a random single-source, single-target query is NP-hard. Several heuristics for this purpose have been proposed such as: $random$, $farthest$ and $avoid$[15], $advanced\ avoid$[9], as well as $maxcover$[16].

For our purpose, we propose a probabilistic variant of the landmark selection strategy $avoid$, a scoring mechanism for selecting a set of landmarks using $maxcover$ and a landmark selection constraint based on a theorem on trees[14]. Most of the improvements of $avoid$ is based on the assumption that some landmarks selected earlier on might be of limited usefulness once others are selected[9),16]. This is highly intuitive from a constructive heuristics and time-independent network perspective, however, as link weights change over time in time-dependent networks, specifically in highly congested parts of the network, the preprocessed landmark-based lower bound cost estimates become unreliable (i.e. the algorithm's search space would increase dramatically). Moreover, the set of landmarks determined using constructive heuristics are used to determine the next candidate landmark from a shortest path tree rooted at a node determined probabilistically. Based on this, it would then be intuitive to propose a probabilistic variant of $avoid$. This variant modifies the deterministic shortest path tree traversal to a probabilistic one using the child node's size to calculate its probability of being traversed. The justification is based on the heuristic nature of landmark selection and that any landmark selected by $avoid$ is optimal only to the origin-destination (OD) pairs in the shortest path tree graph it was selected from. Thus, even a deterministic shortest path tree traversal can lead to suboptimal landmark candidates for the overall network. To improve the landmarks selected by $avoid$, $maxcover$ was proposed[16]. The original $maxcover$ method selects a subset of landmarks from a larger candidate set of landmarks that "covers" as many links as possible. However, the goal in this paper is to select a subset of landmarks that covers as many links as possible including links that are not fully covered using some scoring mechanism to anticipate changes in link weights of time-dependent networks. This ensures that most of the nodes have good lower bound cost estimates that can account for potential changes in shortest paths across the time-dependent network. Lastly, a landmark selection constraint based on a theorem on trees[14] is introduced. Landmark candidates are restricted only to nodes that doesn't belong to a shortest path between landmarks in the landmark set. This is from the observation in[14] that in tree graphs, nodes on the shortest path between landmarks should not be added to the set of landmarks because these nodes will not improve the overall ALT search space.

Using the time-independent Chicago and Kanto networks for experimental analysis, results show that the proposed landmark selection strategies outperform the previous methods. However, when applied to time-dependent networks, landmark-based methods' effectiveness (including the proposed methods) degrade quickly up to the point where its advantage over Dijkstra and A* algorithms is almost negligible.

This paper is organized as follows. In section 2, we introduce the notations used in this paper. Section 3 introduces the ALT algorithm for time-independent and time-dependent networks, and its bidirectional variant. Additionally, the landmark selection strategies used by the ALT algorithm is presented together with some proposed improvements. In section 4, a brief summary of the graph and landmark data representation, data structures and speed-up techniques used in the C++ language is introduced. This is followed by the experimental analysis conducted on the Chicago and Kanto networks in section 5. Lastly, we present our conclusions based on our experiments in section 6.

2. Preliminaries

A graph, $G = (V, E)$, consists of a finite set of nodes, V, and a finite set of links, E. Links can either be composed of unordered or ordered pairs, $(u, v) \in E$, $u, v \in V$, of nodes. We will sometimes interchange $e \in E$ and $(u, v) \in E$ to represent a link. When a graph is composed of the former, it is called an undirected graph. If it is composed of the latter, it is called a directed graph. Throughout this paper, only directed graphs are studied. The node u is called the tail while the node v is called the head of the link. The number of nodes, $|V|$, in the directed graph is denoted by n, the number of links $|E|$ by m, and its links are weighted by a link cost function, $c : E \to \mathbb{F}$. The function space, $\mathbb{F}$, consist of positive periodic functions, $f : \Pi \to \mathbb{R}^+$, $\Pi = [0, p]$, $p \in \mathbb{N}$, such that $f(0) = f(p)$ and $f(x) + x \leq f(y) + y$ for any $x, y \in \Pi$, $x \leq y$ which respects the FIFO property wherein the computation of shortest paths is polynomially solvable[20]. The link cost function is used to evaluate the travel time for a link on a specific departure time p. The upper and lower bounds of f is noted by $\overline{f} = max_{x \in \Pi} f(x)$ and $\underline{f} = min_{x \in \Pi} f(x)$, respectively. A time-independent lower bound graph, $\underline{G}$, of the time-dependent graph, G, can be obtained by substituting the time-dependent link cost function, c, by $\underline{c}$ using $\underline{f}$. A reverse graph, $\overleftarrow{G} = (V, \overleftarrow{E})$, is the obtained from graph G by substituting $(u, v) \in E$ by (v, u).

A node sequence $P = (u_1, ..., u_k)$ in G is called a path if $(u_i, u_{i+1}) \in E$ for all $1 \leq i < k$. In time-independent scenarios, the cost of a path is given by $dist(u_1, u_k) = \sum_{i=1}^{k-1} c_{(u_i, u_{i+1})}$ and a path of minimum cost between two nodes, s and d, is called the $(s, d)-$shortest path with its cost denoted as $dist^*(s, d)$. A potential function is a function $\pi : V \to \mathbb{R}^+$. Given π, the reduced link cost is defined as $c_{e,\pi} = c_{(u,v)} - \pi(u) + \pi(v)$. Suppose c_e is replaced by $c_{e,\pi}$, then for any two nodes x and y, the link cost of any $(x, y)-$path (including the shortest) changes by the same amount, $\pi(y) - \pi(x)$, and thus, are equivalent. We say that π is $feasible$ if $c_{e,\pi} \geq 0$ for all $e \in E$. Feasibility of π is necessary for A* (also Dijkstra) to work correctly. A potential, π, is called $valid$ to a given network if the A* algorithm with the potential, π, outputs an $(s, d)-$shortest path for any pair of nodes (s, d).

In this paper, we consider time-dependent scenarios where the link cost function is given by the non-negative travel time function $c_{(v,u)}(t_u)$, where $t_u \in \Pi$ is the time to leave the node u. In general, $c_{(u,v)}(t_u) \neq c_{(v,u)}(t_v)$. An $(s, d)-$path with a specified departure time from s, t_s, is called an $(s, d, t_s)-$path and the TDSP problem asks to find an $(s, d, t_s)-$path that leaves s at time t_s and minimizes the arrival time at d. Additionally, we only consider networks with the FIFO property and non-negative link weights unless stated otherwise. If the FIFO property doesn't hold, the problem is NP-hard[27].

The following Theorem shows that the FIFO property is important in calculating the TDSP.

Theorem 1. (Halpern[17], Kaufman and Smith[20], Orda and Rom[26]). For a time-dependent FIFO network and an (s, d, t_s) query, there exists an $(s, d, t_s)-$shortest path and it is simple.

3. The A* algorithm with landmarks (ALT)

In this section, the ALT algorithm developed by Goldberg and Harrelson[15] for the time-independent shortest path problem and its extension to the time-dependent problem is introduced.

(1) ALT algorithm development

a) Time-independent ALT algorithm

The ALT algorithm is based on the <u>A</u>* search, <u>L</u>andmarks, and <u>T</u>riangle inequality (not based on Euclidean distances but on shortest path distances). The ALT algorithm relies on landmarks, a small subset of nodes, $l \in L \subset V$, which are used to calculate the potentials of each node, v, in the network during the shortest path search. In order to explain the ALT algorithm, the A* search algorithm will be introduced first. The introduction is taken from[25].

Consider the shortest path problem from a source node, s, to a target node, d, in a network and suppose that there is a potential function $\pi : V \to \mathbb{R}^+$ such that $\pi(v)$ provides an estimate of the length of a $(v, d)-$shortest path for a given target node, d. Given a function $g : V \to \mathbb{R}^+$ and a priority, h, defined by $h(v) = g(v) + \pi(v)$. Let $g^*(v)$ and $\pi^*(v)$ represent the length of the $(s, v)-$shortest path and $(v, d)-$shortest path, respectively. This means that $g^*(v) = dist^*(s, v)$ and $\pi^*(v) = dist^*(v, d)$ and so $h^*(v) = g^*(v) + \pi^*(v) = dist^*(s, d)$. Let Q be a set of nodes that are active (i.e. currently being processed) and R be a set of nodes that have been settled (i.e. nodes that have been processed). Then the A* algorithm is described as follows.

```
A* algorithm:
Input: A network $(G, c)$, a source node $s$, a target node $d$ and a potential function $\pi : V \to \mathbb{R}^+$.
Output: An $(s, d)-$ shortest path and $dist^*(s, d)$.
Algorithm:
  1. For all $v \in V - \{s\}$, let $g(v) = \infty$, $prev(v) = NULL$. Let $g(s) = 0$, $Q = \{s\}$ and $R = \emptyset$.
  2. While $d \notin R$ do:
      (a) Let $u = argmin\{h(v) | v \in Q\}$. Remove $u$ from $Q$ and insert it into $R$.
      (b) For all adjacent nodes $v$ of $u$ with $v \notin R$, if $g(v) > g(u) + c_{(u,v)}$, then set $g(v) = g(u) + c_{(u,v)}$,
          $h(v) = g(u) + c_{(u,v)} + \pi(v)$ and $prev(v) = u$ and insert $v$ into $Q$.
  3. Output the $(s, d)-$path by tracing $prev(v)$ and the length $g(d)$.
```

The A* algorithm is equivalent to Dijkstra's algorithm when $\pi(v) = 0$ for all $v \in V$.

Let $L = \{l_1, ..., l_k\}$ be a set of nodes. Based on

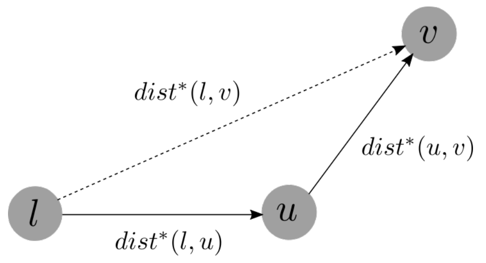

Fig.1 A triangle inequality formed by a landmark and two arbitrary nodes (u, v).

the optimality of the shortest path, for any 3-tuple (u, w, v), it holds that $dist^*(u,v) \leq dist^*(u,w) + dist^*(w,v)$. A potential $\pi(u,l)$ of u with respect to a landmark $l \in L$ is defined as $\pi(u,l) = dist^*(l,v) - dist^*(l,u)$ for a target node v. By definition, the triangle inequality $\pi(u,l) = dist^*(l,v) - dist^*(l,u) \leq dist^*(u,v)$ holds and is shown in **Fig. 1**.

To compute tighter bounds, the maximum potential produced by a landmark in the landmark set can be used, i.e. $\pi(u) = max\left\{0, max\left\{dist^*(l,v) - dist^*(l,u) | l \in L\right\}\right\}$. The following Lemma by Goldberg and Harrelson[15] states that π is feasible.

Lemma 1. (Goldberg and Harrelson[15]) For a given network, a target node v and a landmark l, the potential function π computed using landmarks is feasible.

b) Time-dependent ALT algorithm

For the ALT algorithm in a time-dependent network, the following Lemma shows that a road's free-flow travel time (assumed to be bounded, non-negative and time-independent) used as the link cost can be used to produce feasible potentials for any given landmark as long as the network satisfies the FIFO property and triangle inequality condition (**Fig. 2**).

Lemma 2. (Ohshima[25]). In a FIFO time-dependent network and a given landmark, $l \in L$, the time-dependent potential, $\tilde{\pi}(u,l,t_u)$, is feasible.

Let $\tilde{c}_{(u,v)} = min\left(c_{(u,v)}(t_u)\right)$ for all $t_u \in \Pi$ (e.g. road's free-flow travel time) and $\widetilde{dist^*}(u,v)$ be the length of the $(u,v)-$shortest path with respect to $\tilde{c}$, then $\widetilde{dist^*}(u,v) \leq \widetilde{dist^*}(u,w) + \widetilde{dist^*}(w,v)$, for any 3-tuple (u,w,v) holds. Given a landmark l, the time-dependent potential for an arbitrary (u,v) is denoted as $\tilde{\pi}(u,l,t_u) = \widetilde{dist^*}(l,v) - \widetilde{dist^*}(l,u)$. It follows that, $\tilde{\pi}(u,l,t_u) = \widetilde{dist^*}(l,v) - \widetilde{dist^*}(l,u) \leq \widetilde{dist^*}(u,v) \leq dist^*(u,v,t_u)$ given a $v \in V$, for any $u \in V$ and all $t_u \in \Pi$. Assume that the FIFO time-dependent condition, $t_{u,1} + \tilde{\pi}(u,l,t_{u,1}) \leq t_{u,2} + \tilde{\pi}(u,l,t_{u,2})$, is satisfied. The following fact is well-known.

Lemma 3. If $\tilde{\pi}_1$ and $\tilde{\pi}_2$ are feasible potential functions, then $max\left\{\tilde{\pi}_1, \tilde{\pi}_2\right\}$ is a feasible potential function.

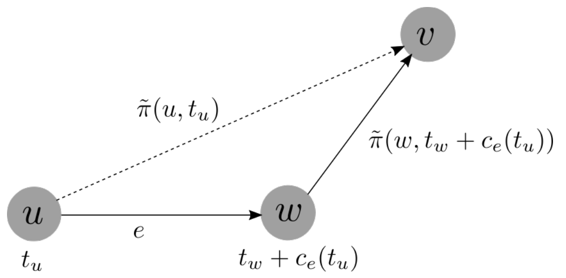

Fig.2 The triangle inequality condition for the potential function $\tilde{\pi}$.

Then, for a set of landmarks, L, each node u, a target v and any time t_u, the time-dependent potential,

$$\tilde{\pi}(u,t_u,l) = \\ max\left\{0, max\left\{\widetilde{dist^*}(l,v) - \widetilde{dist^*}(l,u) | l \in L\right\}\right\}, \quad (1)$$

is feasible. It is clear that a potential that returns zero is feasible. Additionally, as long as time-dependent link costs only increase and do not drop below the minimum link costs used to calculate the potentials, the potentials stay feasible. For any 3-tuple (u,w,v) and all time $t_u, t_w \in \Pi$, if the FIFO time-dependent and triangle inequality condition $\tilde{\pi}(u,l,t_u) \leq c_e(t_u) + \tilde{\pi}(w,l,t_w + c_e(t_u))$ are satisfied, as a consequence of the theorem on the generalized A* algorithm[28] the next Corollary states that the ALT algorithm is valid.

Corollary 1. If the potential function $\tilde{\pi}$ satisfies the FIFO property, the triangle inequality condition and node v is reachable from node u in a time-dependent network, then the ALT algorithm is valid.

For simplicity, the $l \in L$ and $t_u \in \Pi$ in $\tilde{\pi}(u,l,t_u)$ will be dropped since only the potentials in the time-dependent network $(\underline{G}, c_e(t))$ are treated for all e and t, and the potentials in an ALT algorithm are always taken as the shortest path distance from the node u to a landmark l. Hence, $\tilde{\pi}(u,l,t_u) = \tilde{\pi}(u)$.

c) Bidirectional ALT algorithm

Although the ALT algorithm provides faster results when compared to Dijkstra's algorithm, this is only a mild speed-up when compared to an ALT algorithm applied bidirectionally (i.e. alternating an (s,d,t) and (d,s,t) query on the graphs G and $\overleftarrow{G}$, respectively). It may seem simple to do this, however, for a solution to be valid the potential functions of the forward and backward searches (i.e. $\tilde{\pi}_f$ and $\tilde{\pi}_b$, respectively) must be *consistent*, meaning $c_{(u,v),\tilde{\pi}_f}$ must be equal to $c_{(v,u),\tilde{\pi}_b}$. Ikeda et al.[19] developed an average potential function defined as $p_f(v) = (\tilde{\pi}_f(v) - \tilde{\pi}_b(v))/2$ for the forward and $p_b(v) = (\tilde{\pi}_b(v) - \tilde{\pi}_f(v))/2$ for the backward search. By adding $\tilde{\pi}_b(d)/2$ to the forward and $\tilde{\pi}_f(s)/2$ to the backward search, p_f and p_b provide lower bounds to the target and source nodes, respectively. These potentials are both feasible and consistent but provide worse lower bounds than the original potentials.

Since potentials are always feasible if the free-flow travel time of the roads are used, both landmark selection and distance computation can still be performed on the lower bound graph, $\underline{G}$. The bidirectional ALT implemented in this paper was developed by Nannicini et al.[24] and was summarized by Delling and Wagner[11] as follows:

1. A bidirectional ALT is applied to the graph, G, where the forward search is performed on the time-dependent network, and the backward search is run on the lower bound graph, $\underline{G}$. All nodes settled by the backward search are added to a set M. Phase 1 terminates as soon as the two search scopes meet.
2. Suppose that $v \in V$ is a node settled by both searches, then the time-dependent cost $\mu = dist_v(s,d,t)$ of the path from s to d passing through v is an upper bound to $dist(s,d,t)$. Let β be the value of the minimum element in the priority queue of the backward search. Then, phase 2 terminates as soon as $\beta > \mu$.
3. In phase 3, only the forward search continues with the additional constraint that only nodes in the set M can be explored. The forward search terminates when d is settled.

(2) Proposed landmark selection strategies

In this subsection, the landmark selection strategies $avoid$[15] and $maxcover$[16] will be discussed. Additionally, a probabilistic variant of $avoid$, a $maxcover$ scoring mechanism and a landmark selection constraint based on a theorem on trees is proposed. New landmark selection strategies are then developed.

As noted by Goldberg and Werneck[16], "optimal" landmark selection can be defined in many ways depending on how landmark quality is measured. In their paper, $avoid$ and $maxcover$ are shown to be connected as $maxcover$ uses $avoid$ to create a candidate set of nodes bigger than the required number of landmarks.

In the $avoid$ method, it is assumed that a set of landmarks, L, has already been picked and that additional landmarks are required. A shortest path tree, T_r, rooted at node r, selected uniformly at random from the set of nodes, is computed. Then, for each node $v \in V$, the difference between $dist^*(r,v)$ and the lower bound for $dist^*(r,v)$ given by L is calculated. This is the node's $weight$ which is a measure of how bad the current cost estimates are. For each node $v \in V$, its size, $size(v)$, is computed. The size depends on T_v, the subtree of T_r rooted at v. If T_v contains a landmark, set $size(v) = 0$, otherwise, set $size(v)$ as the sum of the $weights$ of all nodes in T_v. Let w be the node of maximum size, traverse T_w starting from w and always follow the child with

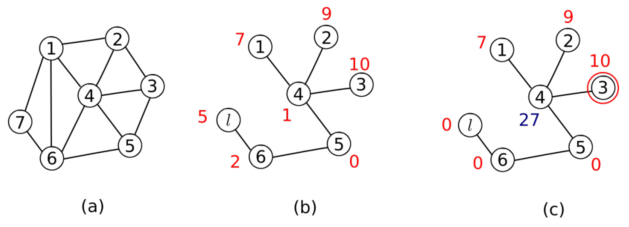

Fig.3 The $avoid$ method. (a) A sample network with 7 nodes is shown. In (b), Node 5 is randomly selected and a shortest path tree, T_5, is computed. The node weights are then computed even for the node 7 landmark. In (c), node sizes are computed. Since, node 7 is a landmark and the subtree, T_6, has a landmark, both sizes are set to 0. Starting from the node with the maximum weight (node 4), traverse the tree deterministically until a leaf is reached (node 3) and make this a landmark.

the largest size until a leaf node is reached. Make this leaf a new landmark (see **Fig. 3**). A variant of $avoid$, called $advanced\ avoid$, was proposed[9] to try to compensate for the main disadvantage of $avoid$ by exchanging the initial landmarks with other landmarks later generated by $avoid$.

Most of the improvements of $avoid$ are based on the assumption that some landmarks selected earlier on might be of limited usefulness once others are selected due to the constructive heuristics used to determine them. Thus, improvements such as $advanced\ avoid$ or $maxcover$ were used to try to discard landmarks selected initially or probabilistically, respectively, based on some criteria. However, landmark selection becomes restricted to the existing landmark set or its subset. Therefore, a probabilistic variant of $avoid$ is proposed where the deterministic shortest path tree traversal is replaced with a probabilistic one using the softmax function,

$$\sigma(v) = \frac{e^{\tau size(v)}}{\sum_{\tilde{v} \leftarrow u} e^{\tau size(\tilde{v})}}, \quad \tilde{v} \in Z, \tag{2}$$

where "$\leftarrow$" denotes node adjacency, $\tau \in [0,1]$, and $Z \subset V$ represents the set of child nodes, $\tilde{v}$, of node u in the directed shortest path tree graph. A $\tau = 1$ uses deterministic traversal while a $\tau = 0$ uses a uniformly random traversal of the nodes in Z. The softmax function highlights nodes with large sizes and suppresses nodes with sizes which are significantly below the maximum size. This is used based on the assumption that a landmark set's effectiveness is tested only with respect to the root node and the possible OD pairs in its shortest path tree and thus, landmarks selected in this manner are optimal only to the OD pairs in their respective shortest path trees and the landmark set used to calculate the node sizes. Moreover, these landmarks may only be reliable in the beginning, uncongested period, and end of a simulation for time-dependent networks as increase in link costs can make preprocessed landmark information unreliable.

For the $maxcover$ method, the reduced cost of a link with respect to a landmark,

$$c_{(u,v),l} = c_{(u,v)} - dist^*(l,u) + dist^*(l,v), \quad (3)$$
$$(u,v) \in E \ \ , \ l \in L,$$

is used to measure how good a solution (set of landmarks) is. A landmark covers a link if the reduced cost is zero, with a best case when the landmark covers every link on the path. This method also tries to compensate for the disadvantages of $avoid$ by computing a set of landmarks 4 times bigger than needed using $avoid$. Interpreting each landmark as the set of links it covers, a local search procedure based on swapping is then applied to a subset, k, of the $4k$ landmarks until a local optimum is reached (i.e. a solution where a set of landmarks covers most links). Anticipating changes in link weights in time-dependent networks, $maxcover$'s scoring mechanism should also include links whose reduced link costs are close to zero. This is because given an (s,d) query, the priorities $h(\tilde{v}) < h(\hat{v})$ of adjacent nodes $\tilde{v}$ and $\hat{v}$ of node u and a potential π, if $h(\tilde{v}) + x < h(\hat{v})$ holds for some $x \geq 0$ and all $\tilde{v}$ due to some potential $\hat{\pi} < \pi$, then the algorithm's search space remains the same. Thus, the goal is not to find a landmark set that provides the tightest landmark-based lower bounds for all nodes (i.e. nodes with zero reduced costs) but to find a landmark set that provides a good overall landmark-based lower bound (i.e. nodes with near zero and zero reduced costs). For each landmark, let each link be represented by,

$$b_{(u,v),l} = \begin{cases} 1, & \text{if } c_{(u,v),l} = 0 \\ \frac{1}{1+c_{(u,v),l}}, & \text{otherwise} \end{cases} . \quad (4)$$

Then, the following score should be maximized,

$$score_{L\subset C} = \sum_{(u,v)\in E} max\{b_{(u,v),l} | l \in L\}, \quad (5)$$
$$k = |L|.$$

Lastly, an additional landmark selection constraint is added that restricts nodes on shortest paths between landmarks from being added in the landmark set. Fuchs[14)] observed that on tree graphs, nodes on the shortest path between landmarks will not improve the overall ALT search space. This is shown by the following Theorem.

Theorem 3. (Fuchs[14)]). Let the graph, $G = (V,E)$, be a tree and $L = \{l_1, ..., l_k\}$ with $k \geq 2$ a set of landmarks. Without loss of generality, let v be a node on the shortest path between l_a and l_b, where $a \neq b$ and $a, b \leq k$. Then, the set $\hat{L} = \{l_1, ..., l_k, v\}$ will not improve the overall ALT search space.

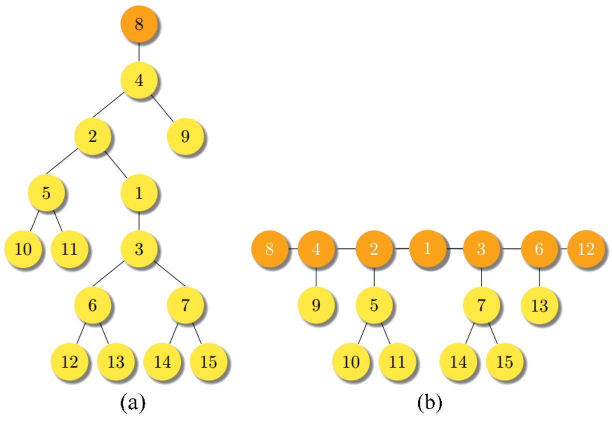

Fig.4 (A figure taken from[14)]). In both images the same complete binary tree with 15 nodes is shown. On the left (a) the tree is drawn with one landmark, node 8, as root and on the right (b) the path from the landmark 8 to the landmark 12 substitutes a root nodes.

a) Probabilistic *avoid* and probabilistic *maxcover*

New landmark selection strategies called *probabilistic avoid* and *probabilistic maxcover* are developed to select k landmarks as follows.

Probabilistic *avoid*:
Algorithm:
1. Starting with x landmarks, select a node, $r \in V$, uniformly at random. If the node is in the landmark set L, repeat this step. Otherwise, create a shortest path tree, T_r, rooted at node r.
2. Calculate the weight of each node defined as the difference between the shortest path distance $dist^*(r,v)$ and the lower bound of $dist^*(r,v)$ with respect to L.
3. Then calculate the size, $size(v)$, of each node in the shortest path tree T_v (a subtree of T_r). The size is calculated as follows, if T_v contains a landmark, set $size(v) = 0$, otherwise, set $size(v)$ as the sum of the $weights$ of all nodes in T_v.
4. Select the node with the maximum size, $w \in V$. Traverse the subtree T_w starting from w probabilistically using equation (2) until a leaf is reached.
5. If the leaf is not a landmark or a node in the shortest path between any two landmarks, make this leaf a new landmark. Otherwise, repeat step 1.
6. Repeat these steps until k landmarks are obtained.

Probabilistic *maxcover*:
Algorithm:
1. Starting with x landmarks, generate $4k - x$ landmarks using *probabilistic avoid* and add each to the landmark candidate set, C.
2. For each landmark $l \in C$, calculate the reduced cost, $c_{(u,v),l}$, of each link, (u,v), with respect to land-

mark l and represent each link by $b_{(u,v),l}$ from equation (4).

3. Starting with a set L of k landmarks selected uniformly at random, calculate this set's score using equation (5).

4. Select a landmark, $\tilde{l} \in L$, randomly and replace it with a landmark, $\hat{l} \in C - L$, then calculate its score using equation (5). If the current score is higher than the previous score, retain the landmark $\hat{l}$. Otherwise, replace landmark $\hat{l}$ with the earlier replaced landmark $\tilde{l}$. Repeat this process for k iterations.

5. After the iteration terminates, select the landmark set, L, with the highest score.

4. C++ implementation

Our implementation stores the graph and landmark data on a hard disk with no assumed capacity constraint. The graphs (G and $\overleftarrow{G}$) are represented as 32-bit vectors which consist of the tail node, its adjacent node and the link cost (e.g. 12 bytes or 92 bits per row). When read into the memory (e.g. Random Access Memory), the graphs are stored on an associative container which contains a key-value pair with unique keys. This allows for a search, insertion and removal of elements in the container to have average constant-time complexity. The key is the tail node u while the value is a 2-tuple which consist of the head node and link cost, (u, c_e). The landmark data is stored as a 32-bit, $n \times (k+1)$ vector consisted of node IDs and k landmarks (e.g. 24 bytes or 192 bits per row for $k = 5$). A fixed, 32-bit vector container is used to store the landmark data when read into the memory which means that there is also no assumed capacity constraint on the memory.

To accelerate computation of expensive function calls (e.g. repeated calls of the Dijkstra's algorithm to compute the shortest path tree in $avoid$), memoization is used. It is an optimization technique used primarily to speed up computer programs by storing the results of expensive function calls and returning the cached result when the same inputs occur again. This reduced the time to create shortest path trees from a few days in the Kanto network to only a few minutes.

5. Experimental analysis

In this section, the proposed landmark selection algorithms, *probabilistic avoid* (ALT prob_avoid) and *probabilistic maxcover* (ALT prob_maxcover), were tested using the ALT algorithm on a time-independent Chicago network, and a time-independent and time-dependent Kanto network. Results of the tests were compared with the Dijkstra, A*, and ALT algorithms using $random$ (ALT random), $avoid$ (ALT avoid) and $maxcover$ (ALT maxcover) landmark selection strategies on the same networks using the same settings. Each ALT algorithm was run 5 times and the best solution was used. The Chicago network has 933 nodes and 2,950 links while the Kanto network has 195,180 nodes and 439,979 links. Shortest path search were conducted on 500 OD pairs selected uniformly at random. The same OD pairs were used for each run of the ALT algorithm with different landmark sets chosen based on the different landmark selection strategies. In the Chicago network, 6 landmarks were used for each ALT shortest path search while 16 landmarks were used in the Kanto network. These tests were conducted on an Ubuntu 16.04 LTS, 64-bit computer with 62.8 GB of RAM and an Intel Core i7-5960X CPU @ 3.00 GHz $\times$ 16.

Two machine-independent measures of quality were used to get a better understanding of the algorithms. We measure an algorithm's efficiency defined as,

$$\text{efficiency} = \frac{|(s,d) - \text{shortest path}|}{|R|}, \qquad (6)$$

where $|(s,d) - \text{path}|$ is the number of nodes in the $(s,d)-$shortest path and $|R|$ is the number of nodes settled by the algorithm. The average and the 99th percentile efficiencies were taken as we are interested in optimizing the worst case efficiency of the algorithm[16)]. The 99th percentile is used as it gives a more stable measure of the relative performance between two different landmark selection methods.

Fig. 5 shows the efficiency of the Dijkstra, A* search and ALT algorithms on the time-independent Chicago network (higher is better). It is noticeable from the average results that a 9.2% and 11.3% increase in average efficiency was achieved by the *probabilistic avoid* and *probabilistic maxcover* against $avoid$ and $maxcover$, respectively. For the 99th percentile, the figure confirms the results of other researchers and also supports the results of the average efficiency showing the *probabilistic avoid* (a 1.6% increase) and *probabilistic maxcover* (a 3.7% increase) perform slightly better than $avoid$ and $maxcover$, respectively.

Table 1 shows the landmark generation and query times of the shortest path search algorithms in the time-independent Chicago network. These results were taken using one randomly picked OD pair for the search query (in this case, an OD pair with a shortest path which consist of 7 nodes). In terms of the speed in landmark generation, $random$ is the fastest, $avoid$ and *probabilistic avoid* took almost the same amount of time, and both $maxcover$ and *probabilistic maxcover* also took almost the same amount of time. The largest percentage of time taken by $maxcover$ and *probabilistic maxcover* was on the repeated calls to

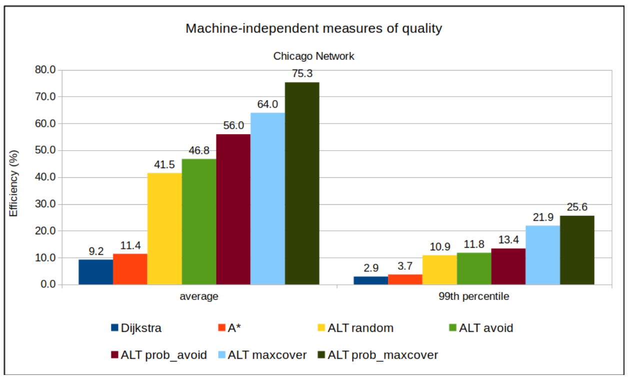

Fig.5 Shortest path search efficiency in the Chicago time-independent network.

Table1 Landmark generation of 6 landmarks and query times for a 7-node shortest path in the Chicago time-independent network.

Algorithm	Generation time (ms)	Query time (ms)
Dijkstra	—	84.30
A*	—	75.70
ALT random	23	0.707
ALT avoid	44	0.543
ALT prob_avoid	45	0.501
ALT maxcover	297	0.443
ALT prob_maxcover	301	0.442

avoid and *probabilistic avoid*, respectively. These results confirm our assumptions regarding the improvements for *avoid* and *maxcover*.

The same tests were conducted on the Kanto network shown in **Fig. 6**. In **Fig. 7**, although a similar pattern is shown, only modest increases in efficiencies were obtained. A 0.7% and 0.3% increase in average efficiency between ALT avoid and ALT prob_avoid, and ALT maxcover and ALT prob_maxcover, respectively. Additionally, a 4.1% difference in efficiency between the Dijkstra and A* algorithm in this network as compared to a 2.2% difference in the Chicago network can be seen. The difference is caused by the increased number of nodes in between the OD pairs in this network (Dijkstra's algorithm's search space increases exponentially) as compared to the Chicago network creating a larger disparity between the two. There is also a noticeable 8.5% increase in efficiency between the A* and ALT random algorithm which is due to the increase in landmarks available for the ALT random algorithm (16 landmarks in this network as compared to only 6 landmarks in the Chicago network). Moreover, a 5.4% increase in efficiency between the ALT prob_avoid and ALT maxcover can be seen. This difference is due to the fact that *maxcover* chooses 16 landmarks from 64 landmark candidates ($4k = 4\times16$) using a local optimization technique for maximum link coverage which confirms the results of Goldberg and Harrelson[15)]. Lastly, the 99th percentile

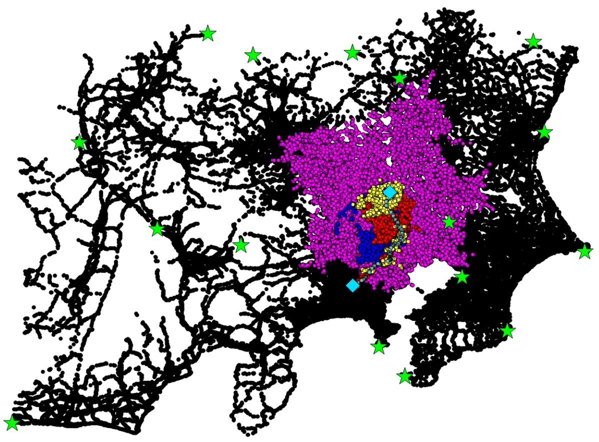

Fig.6 The Kanto network (black nodes) shown with the Dijkstra's (pink nodes), A*'s (blue nodes), ALT *maxcover*'s (red nodes) and ALT *probabilistic maxcover*'s (yellow nodes) search spaces. The OD's (blue diamond) shortest path are the grey nodes. The landmarks (selected using *maxcover*) are shown as green stars.

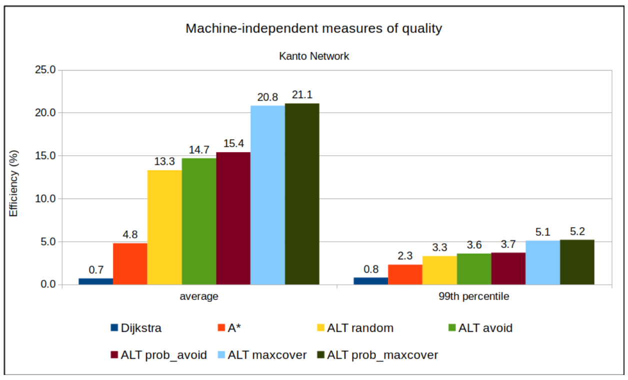

Fig.7 Shortest path search efficiency in the Kanto time-independent network.

Table2 Landmark generation of 16 landmarks and query times for a 185-node shortest path in the Kanto time-independent network.

Algorithm	Generation time (s)	Query time (s)
Dijkstra	—	0.2471
A*	—	0.1660
ALT random	15.8	0.1553
ALT avoid	1673.4	0.1488
ALT prob_avoid	1597.4	0.0958
ALT maxcover	10140.4	0.0624
ALT prob_maxcover	9884.4	0.0515

efficiency result shows a similar pattern from the average efficiency result (an increase of 0.1% for both) which confirms the algorithms' effectiveness.

Table 2 shows the landmark generation and query times of the shortest path search algorithms in the time-independent Kanto network. These results were taken using one randomly picked OD pair for the search query (in this case, an OD pair with a short-

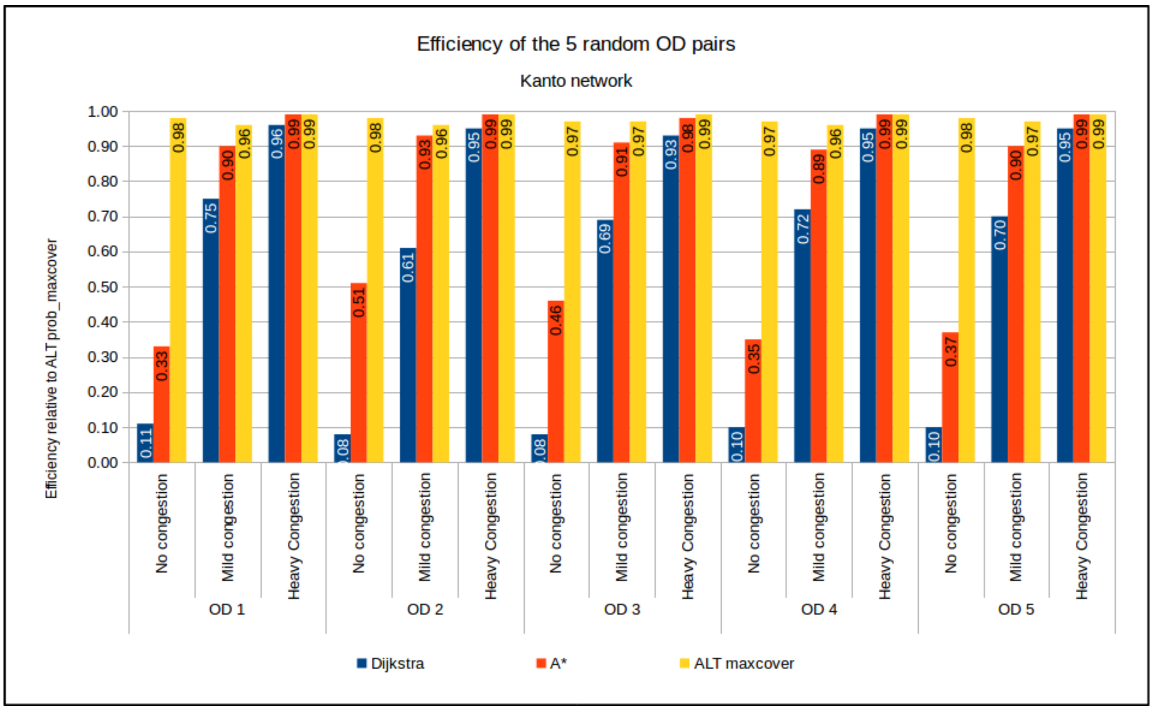

Fig.8 The effect of the time-dependent link costs on landmark efficiencies relative to ALT prob_maxcover in the Kanto network.

est path which consist of 185 nodes). In terms of the speed in landmark generation, *random* is the fastest while *avoid* and *probabilistic avoid* has a slight difference caused by the algorithm design (since *avoid* sometimes find the same landmark, a node r is again selected randomly to be the root for the shortest path tree). The same situation occurred in the case of *maxcover* and *probabilistic maxcover*. Similarly, the largest percentage of time taken by *maxcover* and *probabilistic maxcover* was on the repeated calls to *avoid* and *probabilistic avoid*, respectively. Additionally, the query times of each algorithm confirms the assumption that the lesser the search space of the algorithm, the faster the shortest path search calculation. These results also confirm our assumptions regarding the improvements for *avoid* and *maxcover*.

Tests of the algorithms' efficiencies on the time-dependent Kanto network were conducted on 5 randomly chosen OD pairs. The efficiency values shown in **Fig. 8** are all relative to ALT prob_maxcover for clarity (i.e. all ALT prob_maxcover values are all set to 1.00). Three tests representing *no congestion*, *mild congestion* (e.g. 5 minute delay) and *heavy congestion* (e.g. 30 minute delay) which is a usual scenario in a traffic simulation were used. Additionally, only the Dijkstra, A*, ALT maxcover and ALT prob_maxcover algorithms were run for each OD pair in each case.

In the *no congestion* sections of **Fig. 8**, the algorithms reproduce the results taken in the previous time-independent scenario of the Kanto network. Although a small variation in the results may be observed, these are consistent with the findings in the time-independent case. In the *mild congestion* sections, it can be noticed that ALT prob_maxcover performs better at anticipating the change in link costs than *maxcover*. However, these landmark-based lower bounds' effectiveness degrades quickly up to the point where the difference between the Dijkstra and A* algorithm during the *heavy congestion* sections are almost negligible. This result is expected because the landmark-based lower bound estimates were based on the lower bound link costs. Since the link costs have increased, these lower bounds become highly inaccurate. Additionally, The increase in query times for each case were proportional to the increase in search space in the algorithms for 16 landmarks.

Note that these savings in efficiency may be small but it can become useful in a large-scale simulation when multiplied with the number of drivers that will calculate their shortest paths to their destinations.

6. Conclusion

We presented an efficient implementation of the ALT algorithm for the time-dependent shortest path problem improving upon the landmark selection strategies in[15] by anticipating changes in the link costs in the network using the C++ programming language.

For the time-independent networks, results show that the proposed landmark selection strategies performed better than previous methods achieving at most an 11.3% and 3.7% increase in average efficiency and 99th percentile in the Chicago network, respectively. In the Kanto network, a similar result pattern, albeit, only a very modest improvement (at most a 0.7% and 0.1% for the average and 99th percentile efficiency, respectively) compared to the Chicago network was obtained.

When the algorithms were applied to the time-dependent Kanto network, the proposed algorithms still performed better than the previous methods. However, the landmark-based methods' effectiveness quickly degraded even in mild congestion scenarios. Additionally, it further degraded when heavy congestion occurred up to the point where its advantage over Dijkstra and A* algorithms were almost negligible. Moreover, as discussed in[11)], pure ALT suffers from two major drawbacks. Space consumption is rather high and even more important ALT cannot compete with hierarchical approaches concerning query performance in transportation networks.

It would be better to use other speed-up techniques combined with the ALT algorithm such as the Core-ALT[10)] or L-SHARC[7)], or a totally different speed-up technique like Contraction Hierarchies[1)] when used in time-dependent networks. However, the speed-up techniques mentioned above are highly applicable only if preprocessing time is not an issue with Core-ALT as an exception. So in[11)], they showed how Core-ALT remedies both drawbacks mentioned above without violating the advantages of pure ALT, i.e., easy adaption to dynamic scenarios and robustness to the input. Therefore, a future direction is to combine the proposed methods with the Core-ALT speed-up technique.

ACKNOWLEDGMENT: This research was conducted as part of Sub-group B, "Development of a System for Integrated Simulation of Earthquake and Tsunami Hazard and Disaster", of Priority Issue (3), "Construction of integrated prediction system of complex disasters by earthquake and tsunami", of the Post K Project, "Application development and R & D on social and scientific issues to be addressed intensively in Post K (Ministry of Education, Culture, Sports, Science and Technology)". Additionally, we would like to acknowledge the help of the Regional Futures Research Center for providing us useful information regarding shortest path search algorithms.

REFERENCES

1) Batz, V., Delling, D., Sanders, P. and Vetter, C. (2009). *Time-Dependent Contraction Hierarchies*. In Proceedings of the 11th Workshop on Algorithm Engineering and Experiments (ALENEX'09), pages 97105. SIAM, April.
2) Bauer, R., Delling, D., Sanders, P., Schieferdecker, D., Schultes, D., and Wagner, D. (2008). *Combining Hierarchical and Goal-Directed Speed-Up Techniques for Dijkstra ' s Algorithm.* In C. C. McGeoch, editor, Proceedings of the 7th Workshop on Experimental Algorithms (WEA ' 08), volume 5038 of Lecture Notes in Computer Science, pages 303318. Springer, June.
3) Bauer, R., Columbus, T., Katz, B., Krug, M. and Wagner, D. (2010). Preprocessing Speed-Up Techniques is Hard. *In Proceedings of the 7th Conference on Algorithms and Complexity (CIAC ' 10)*, Lecture Notes in Computer Science, Springer.
4) Bellman, R. (1958). On a routing problem. *Quarterly of Applied Mathematics*, 16:87-90.
5) Cooke, K. L. and Halsey, E. (1966). The shortest route through a network with time-dependent internodal transit. *J. Math. Anal. Appl.*, 14:493-498.
6) Dean, B. C. (1999). Continuous-Time Dynamic Shortest Path Algorithms. *Master ' s thesis*, Massachusetts Institute of Technology.
7) Delling, D. (2008). Time-Dependent SHARC-Routing. *Algorithmica, July 2009, Special Issue: European Symposium on Algorithms.*
8) Delling, D. (2009). Engineering and Augmenting Route Planning Algorithms. *PhD thesis*, Universitt Karlsruhe (TH), Fakultt fr Informatik.
9) Delling, D., Sanders, P., Schultes, D., and Wagner, D. (2006). Highway hierarchies star. *In 9th DIMACS Implementation Challenge*.
10) Delling, D. and Nannicini, G. (2008). *Bidirectional Core-Based Routing in Dynamic Time-Dependent Road Networks.* In S.-H. Hong, H. Nagamochi, and T. Fukunaga, editors, Proceedings of the 19th International Symposium on Algorithms and Computation (ISAAC ' 08), volume 5369 of Lecture Notes in Computer Science, pages 813824. Springer, December.
11) Delling, D. and Wagner, D. (2009). Time-dependent route planning. *Robust and online large-scale optimization*, 207-230.
12) Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269-271.
13) Dreyfus, S. E. (1969). An appraisal of some shortest-path algorithms. *Operations Research*, 17(3):395-412.
14) Fuchs, F. (2010). On Preprocessing the ALT Algorithm. *Master's thesis*, University of the State of Baden-Wuerttemberg and National Laboratory of the Helmholtz Association, Institute for Theoretical Informatics.
15) Goldberg, A. V. and Harrelson, C. (2005). Computing the shortest path: A search meets graph theory. *In SODA 2005*, pp. 156-165. SIAM.
16) Goldberg, A. V. and Werneck, R.F. (2005). Computing point-to-point shortest paths from external memory. *In Proceedings of the 7th Workshop on Algorithm Engineering and Experiments (ALENEX'05)*, pages 26-40. SIAM.
17) Halpern, H. J. (1977). Shortest route with time dependent length of edges and limited delay possibilities in nodes. *Operations Research*, 21:117-124.
18) Hart, P. E., Nilsson, N. J. and Raphael, B. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Sci-*

ence and Cybernetics SSC4, 4 (2): 100-107.
19) Ikeda, T., Hsu, M., Imai, H., Nishimura, S., Shimoura, H., Hashimoto, T., Tenmoku, K., and Mitoh, K. (1994). A Fast Algorithm for Finding Better Routes by AI Search Techniques. *In Proc. Vehicle Navigation and Information Systems Conference*, IEEE.
20) Kaufman, D. E. and Smith, R. L. (1993). Fastest paths in time-dependent networks for intelligent vehicle-highway systems application. *Journal of Intelligent Transportation Systems*, 1(1):1-11.
21) Lee, D., Choi, J., Noh, S. H., Min, S. L., Cho, Y., and Kim, C. S. (1999). On the Existence of a Spectrum of Policies that Subsumes the Least Recently Used (LRU) and Least Frequently Used (LFU) Policies. *In Proceedings of the 1999 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 134-143.
22) Luby, M. and Ragde, P. (1989). A bidirectional shortest-path algorithm with good average-case behavior. *Algorithmica*, 4(4):551-567.
23) Megiddo, N. and Modha, D. S. (2003). ARC: A Self-Tuning, Low Overhead Replacement Cache. *In Proceedings of the 2003 File and Storage Technologies (FAST)*, pp. 115-130.
24) Nannicini, G., Delling, D., Liberti, L., Schultes, D. (2008). Bidirectional A* Search for Time-Dependent Fast Paths. *In: McGeoch, C.C. (ed.) WEA 2008*, LNCS, vol. 5038, pp. 334-346. Springer, Heidelberg.
25) Ohshima, T.， Nagamochi, H， and Zhao, L. (2008). A Landmark Algorithm for the Time-Dependent Shortest Path Problem. *Master's thesis*, Kyoto University, Graduate School of Informatics, Department of Applied Mathematics and Physics.
26) Orda, A. and Rom, R. (1990). Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length. *Journal of the ACM*, 37 (3), pp. 607-625.
27) Sherali, H. D., Ozbay, K. and Subramanian, S. (1998). Time-dependent shortest pair of disjoint paths problem: Complexity, models, and algorithms. *Networks*, 31(4):259-272.
28) Zhao, L., Ohshima, T., and Nagamochi, H. (2008). A* algorithm for the time-dependent shortest path problem. *The 11th Japan-Korea Joint Workshop on Algorithms and Computation (WAAC08)*, July 19-20, Fukuoka, Japan, pp. 36-43.

(Received April 28, 2017)