A* Algorithm for the time-dependent shortest path problem

Liang Zhao

Graduate School of Informatics
Kyoto University
Yoshidahonmachi, Sakyoku, Kyoto, Japan
liang@i.kyoto-u.ac.jp

Tatsuya Ohshima

Graduate School of Informatics
Kyoto University
Yoshidahonmachi, Sakyoku, Kyoto, Japan
ohshima@amp.i.kyoto-u.ac.jp

Hiroshi Nagamochi

Graduate School of Informatics
Kyoto University
Yoshidahonmachi, Sakyoku, Kyoto, Japan
nag@i.kyoto-u.ac.jp

Abstract: Given a directed graph, a nonnegative transit-time function $c_e(t)$ for each edge $e = (v, w)$ (where t is the time to leave v), a source node s, a destination node d and a departure time t_0, the time-dependent shortest path problem asks to find an s, t-path that leaves s at time t_0 and minimizes the arrival time at d. This formulation generalizes the classical shortest path problem in which c_e are constants.

For this problem, during the 40 years since the generalized Dijkstra algorithm was suggested by Dreyfus '69, there was no significant advancement despite of many studies. This paper presents a novel generalized A* algorithm and, as an application, also gives a generalization of the ALT algorithm for the classical problem due to Goldberg and Harrelson '05.

Keywords: shortest path, time-dependent shortest path, A* algorithm, ALT algorithm

1 Introduction

The shortest path problem is a classical problem that appears in every book on combinatorial optimization. It has countless applications and so far numerous algorithms have been proposed (see, e.g., [1]), including the well-known Dijkstra's algorithm. Recently, partly because new improvement becomes fairly difficult, researchers began to study variants of this problem, which include the *time-dependent* generalization.

Given a directed graph $G = (V, E)$, a nonnegative transit-time function $c_e(t)$ for each edge $e = (v, w) \in E$ (where t is the time to leave v), a source node $s \in V$, a destination node $d \in V$

*Research is partially supported by the Japan Ministry of Education, Culture, Sports, Science and Technology.

and a departure time t_0, the *time-dependent shortest path problem* asks to find an s,t-path that leaves s at time t_0 and minimizes the arrival time at d (see Figure 1 for an illustration). Notice undirected graphs can be treated by replacing each (undirected) edge with two reverse directed edges. Without loss of generality, we suppose d is reachable from s. For simplicity, we suppose the domain of definition for all $c_e(t)$ is $\mathbb{R}^+$ (i.e. the set of nonnegative reals), but our algorithms work for the discrete version too. We also assume the time complexity to calculate a $c_e(t)$ is bounded by some constant α. This formulation generalizes the classical shortest path problem (with constant $c_e(t)$ and t_0). It can further handle time-variable edge costs, thus has more applications than the classical one, which is also referred to as the *static* problem in contrast.

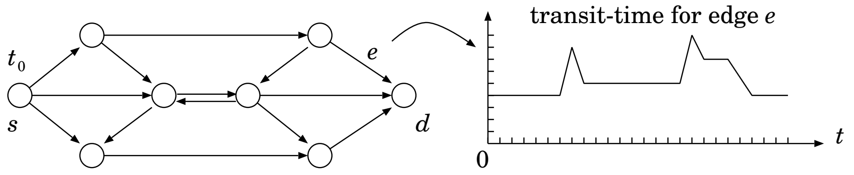

Figure 1: An illustration of the time-dependent shortest path problem. The difference from the classical (static) problem is that the edge length is generalized from a constant to a time-variable function (hence a departure time t_0 at the source s is also needed as an input).

Actually this problem is not new. Cook and Halsey [2] considered it and gave a Dynamic-Programming algorithm which is not polynomial-time at all. Dreyfus [4] then suggested a (polynomial-time) *straightforward* generalization of the Dijkstra algorithm (see also Section 2). However, he did not notice that it works correctly only for instances satisfying the FIFO (First-In First-Out) property, i.e., for any edge $e = (v, w)$ and $t_1 \leq t_2$, it holds that $t_1 + c_e(t_1) \leq t_2 + c_e(t_2)$ (in other words, the arrival-time function $t + c_e(t)$ is nondecreasing). With this property, we can ensure that there is no cycle of negative transit-time, hence a simple optimal solution exists. This was pointed out and discussed later by a number of studies [7, 10, 13].

On the other hand, the general problem without the FIFO constraint is NP-hard if waiting at nodes is *not* allowed ([12, 14]). Orda and Rom [13] showed that, however, if waiting at nodes *is* allowed (which is natural in transportation systems), then any instance can be converted to an equivalent instance that satisfies the FIFO property (hence no waiting is needed), and that can be done in polynomial time (if $c_e(t)$ can be calculated in polynomial time). Thus in the following, we will only consider instances that satisfy the FIFO property.

Even with the FIFO constraint, unlike the static case, studies are not rich. During the past near 40 years after Dreyfus's proposal of the generalized Dijkstra algorithm, despite of many studies (e.g., [3, 5, 7, 9, 10, 13, 14]), there was no *significant* advancement in solving the problem more efficiently. In this paper, we give a novel algorithm that generalizes the A* algorithm [8] for the static problem. Unlike the generalized Dijkstra algorithm, this generalization is not trivial, see Section 2. We note that Kanoulas et al. [9] considered, for piecewise linear functions, a slightly-generalized A* algorithm, which we will discuss later in Section 2 too.

Furthermore, as an application of our algorithm, in Section 3 we will give a generalization of the ALT algorithm ([6]) that is based on the static A* algorithm and is faster than the Dijkstra algorithm using preprocessing. Thus we have found the first algorithm for the time-dependent problem that speeds up the calculation using preprocessing, which is observed to be several times faster than the generalized Dijkstra algorithm. Finally we conclude in Section 4.

2 A* algorithm for the time-dependent shortest path problem

For ease of understanding, let us start from the classical and well-known Dijkstra algorithm.

Suppose $c_e(t) \equiv c_e$ is a constant for each edge e and $t_0 = 0$ (the value of t_0 is not important in this static case), the Dijkstra algorithm tries to find a shortest s, d-path in a greedy manner. Let $p(v)$ denote the precedent node of a node v in the shortest s, v-path found so far. The Dijkstra algorithm maintains for each node v a status(v) $\in$ {"unlabeled", "labeled", "finished"} and a distance label $g(v)$. At the beginning, $g(s)$ is set to 0 and all status(v) are initialized to "unlabeled" except that s is "labeled". Then it repeatedly finds a "labeled" node v with the smallest $g(v)$ (such v is called the *active* node) until $v = d$; then it tries to *relax* all non-"finished" neighbours w of v, i.e., if status(w) = "unlabeled" then set it to "labeled" and let $g(w) = g(v) + c_{(v,w)}$, $p(w) = v$; otherwise status(w) = "labeled", then let $g(w) = g(v) + c_{(v,w)}$, $p(w) = v$ if $g(w) > g(v) + c_{(v,w)}$; after all these have done, set status(v) to "finished" and continue. See Table 1 for the pseudo-code.

Table 1: Pseudo-code of the Dijkstra algorithm for the (static) shortest path problem.

```
1  status($s$) := "labeled", $g(s) := 0$, status($v$) := "unlabeled" for all $v \neq s$
2  Let $v$ be a "labeled" node with the smallest $g(v)$ (the active node). IF $v = d$ GOTO 11
3  FOR all edges $(v, w)$ DO
4     IF status($w$) = "unlabeled" THEN
5        status($w$) := "labeled", $g(w) := g(v) + c_{(v,w)}$, $p(w) := v$
6     ELSE IF status($w$) = "labeled" AND $g(w) > g(v) + c_{(v,w)}$ THEN
7        $g(w) := g(v) + c_{(v,w)}$, $p(w) := v$
8     END IF
9  DONE
10 status($v$) := "finished". GOTO 2
11 OUTPUT $g(d)$ and the $s, d$-path found (i.e. the reverse of $d, p(d), p(p(d)), \ldots, s$).
```

The A* algorithm (Table 2) follows the same fashion except that it employs an *estimator* $h(v)$ for all v and chooses the active node by the smallest $g(v) + h(v)$. Notice that how to determine $h(v)$ is not part of the algorithm. It must be obtained by some other method, and the choice of h determines the correctness and the efficiency of the A* algorithm (a good lower-bound on the v, d-distance is preferred). Clearly the Dijkstra algorithm is a special case with $h \equiv 0$.

Table 2: Pseudo-code of the A* algorithm for the static problem. Notice that the Dijkstra algorithm is a special case of $h \equiv 0$. For general h, however, the correctness is not guaranteed.

```
⋮                    (same as Table 1)
2  Let $v$ be a "labeled" node with the smallest $g(v) + h(v)$. IF $v = d$ GOTO 11
⋮                    (same as Table 1)
```

Now we are ready to describe our generalized A* algorithm. It generalizes $h(v)$ by the time-dependent version $h(v, t)$, where t is the time at node v. Thus in Table 3, we use $h(v, g(v))$ to replace $h(v)$. Notice the rule for choosing the active node (Line 2) has been changed in addition.

Table 3: Pseudo-code of our A* algorithm for the time-dependent shortest path problem.

```
1   status($s$) := "labeled", $g(s) := t_0$, status($v$) := "unlabeled" for all $v \neq s$
2   Let $v$ be a "labeled" node with the smallest $g(v) + h(v, g(v))$. In the case that there
      are multiple candidates, choose one with the smallest $g(v)$. IF $v = d$ GOTO 11
3   FOR all edges $(v, w)$ DO
4     IF status($w$) is "unlabeled" THEN
5       status($w$) := "labeled", $g(w) := g(v) + c_{(v,w)}(g(v))$, $p(w) := v$
6     ELSE IF status($w$) is "labeled" AND $g(w) > g(v) + c_{(v,w)}(g(v))$ THEN
7       $g(w) := g(v) + c_{(v,w)}(g(v))$, $p(w) := v$
8     END IF
9   DONE
10  status($v$) := "finished". GOTO 2
11  OUTPUT $g(d)$ and the $s, d$-path found (i.e. the reverse of $d, p(d), p(p(d)), \ldots, s$).
```

In general, the above algorithm may fail to find an optimal solution. The next theorem gives two sufficient (but reasonable) conditions for the correctness.

Theorem 1 *Given an instance (G, c, s, d, t_0) of the time-dependent shortest path problem such that satisfies the FIFO property and d is reachable from s, the generalized A* algorithm in Table 3 finds an optimal solution if h satisfies the next conditions.*

- **(FIFO Condition)** *For all nodes v and $t_1 \leq t_2$, $t_1 + h(v, t_1) \leq t_2 + h(v, t_2)$.*
- **(Triangle Condition)** *For all edges $e = (v, w)$ and t, $h(v, t) \leq c_e(t) + h(w, t + c_e(t))$.*

Before going to the proof, we remark that the Triangle Condition (illustrated in Figure 2) is a natural generalization from the classical A* algorithm, whereas the FIFO Condition is only available in the time-dependent case. The generalized Dijkstra algorithm is nothing but the simplest case with $h \equiv 0$, and the generalization of Kanoulas et al. [9], on the other hand, simply uses a constant function $h(v, t) = h(v)$, thus it is also a simple special-case of our algorithm.

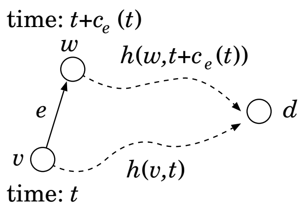

Figure 2: An illustration of the Triangle Condition for function h. Roughly speaking, it says the supposed transit-time $h(v, t)$ from v to d is no more than $c_e(t) + h(w, t + c_e(t)$, i.e. the supposed transit-time of the v, d-path $v \to w \rightsquigarrow d$. Notice $h(w, t + c_e(t))$ is the supposed transit-time from w to d by leaving w at time $t + c_e(t)$.

It is easy to see the next lemma by the Triangle Condition (by induction on k).

Lemma 1 *Let $P = v_1, v_2, \ldots, v_k$ be a path and t be a departure time at v_1. Define $\sigma_1 = 0$ and $\sigma_i = \sum_{j=1}^{i-1} c_{(v_j,v_{j+1})}(t + \sigma_j)$ be the transit-time from v_1 to v_i, $i = 2, \ldots, k$. Then it holds that*

$$h(v_1, t) \;\leq\; \sigma_k + h(v_k, t + \sigma_k).$$

(*The Triangle Condition is the case of $k = 2$.*) ■

Proof for Theorem 1. We show by induction that, every active node v must get the optimal distance label, i.e., the earliest arrival time at v for leaving s at time t_0. Obviously this will prove the theorem (notice we have supposed d is reachable from s).

Let v be an active node and consider when v is active. If $v = s$, we are done. Otherwise let P be a simple optimal s, v-path (it exists!) and w be the *first* node on P such that status(w) $\neq$ "finished". Clearly w must exist and $w \neq s$ (it can be v), see Figure 3 for an illustration.

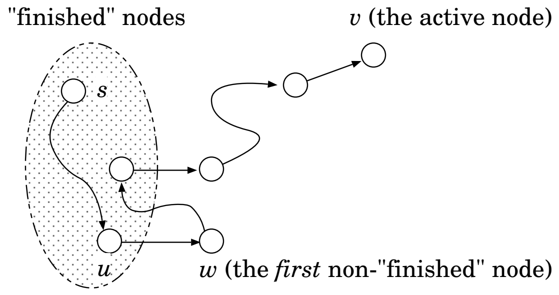

Figure 3: Illustration for the proof of Theorem 1 where an optimal s, v-path is being considered.

Let g^* denote the optimal distance (i.e. the earliest arrival time). It is obvious that $g(w) = g^*(w)$ because w was relaxed when the precedent node u of w was active and at that time $g(u) = g^*(u)$ (by the induction hypothesis). Let $\sigma = g^*(v) - g^*(w)$ be the shortest transit-time from w to v at departure time $g^*(w)$ (notice $\sigma \geq 0$). By applying Lemma 1 to the w, v-path on P with $t = g^*(w)$, we have

$$h(w, g^*(w)) \;\leq\; \sigma + h(v, g^*(w) + \sigma) \;=\; g^*(v) - g^*(w) + h(g^*(v)).$$

That is equivalent to

$$g^*(w) + h(w, g^*(w)) \;\leq\; g^*(v) + h(w, g^*(v)).$$

Then, since v is the active node (thus has the smallest $g(v) + h(v, g(v))$), we have

$$g(v) + h(v, g(v)) \;\leq\; g(w) + h(w, g(w)) \;=\; g^*(w) + h(w, g^*(w)) \;\leq\; g^*(v) + h(w, g^*(v)). \quad (1)$$

On the other hand, by the FIFO Condition and $g^*(v) \leq g(v)$ (the optimality of g^*), we have

$$g^*(v) + h(v, g^*(v)) \;\leq\; g(v) + h(v, g(v)). \quad (2)$$

Therefore we get the next fact by combining (1) and (2).

$$g(v) + h(v, g(v)) \;\leq\; g(w) + h(w, g(w)) \;=\; g^*(w) + h(w, g^*(w)) \;\leq\; g(v) + h(v, g(v)).$$

This means the equalities hold, hence $g(v)+h(v,g(v)) = g(w)+h(w,g(w))$. Then by our choice of the active node, $g(v) \leq g(w)$ must hold. Thus $g(v) \leq g^*(w) \leq g^*(v)$, hence $g(v) = g^*(v)$. ■

Remark 1. We remark that, analogously to the static version, an h with $h(d,t) \equiv 0$ implies $h(v,t)$ is a lower bound on the shortest transit-time from v to d with departure time $g(v)$ (by Lemma 1). Moreover, it is not difficult to show that with an h satisfying $h(d,t) \equiv 0$ and $h \geq 0$, the search space (the set of active nodes) of the generalized A* algorithm is no larger than that of the generalized Dijkstra algorithm. Using this observation, we will give an algorithm in the next section that is (practically) faster than the generalized Dijkstra algorithm.

Remark 2. We note the two Conditions and the way to choose the active nodes are reasonable in the meaning that without one of them, we can find examples for which the A* algorithm may fail to find an optimal solution. This is not difficult but we omit it due to the page limit.

3 An application: the generalized ALT algorithm

It is easy to see that the time complexity of the generalized Dijkstra algorithm is $O(n \log n + m\alpha)$ by using a Fibonacci heap (we note it was $O((m+n\log n)\alpha)$ in [5]), where m, n, α are the number of edges, the number of nodes, and the time complexity to calculate $c_e(t)$, respectively. While we cannot improve this theoretical bound, let us give a practically faster algorithm that is based on our A* algorithm and generalizes the (static) landmark-based ALT algorithm [6].

The ALT algorithm is such an algorithm that is supposed to answer (unknown) shortest-path queries for a known graph. This means we can preprocess the graph beforehand and use it to answer a query faster than a normal calculation by, e.g., the Dijkstra algorithm. Of course there is a trivial method of saving solutions for all possible queries and answer a query in $O(1)$ time, but the n^2 order (for the static case) is big (if not impossible) for large graphs, e.g., usually a road network is sparse (i.e., $m \leq kn$ for some small k) and has several millions of nodes. So researchers are seeking efficient algorithms that uses $O(n)$ storage, see [15] for a review. While this is an extremely hot topic for the static problem these several years, for the time-dependent case, as far as we know, there was no proposal before our work.

Now let us describe the detail of our generalized ALT algorithm. Let $\tau^*(v,w,t)$ denote the shortest transit-time from a node v to another node w with departure time t (hence we want to find an s,d-path of transit-time $\tau^*(s,d,t_0)$). Suppose we have a node z and values $\tau^*(z,v,t)$ for all nodes v and all t (z is called a *landmark*). Also suppose we can calculate a $\hat{t}$ (if exists) that

$$\hat{t} \;=\; \max\{t' : t' + \tau^*(z,v,t') \leq t\}.$$

In other words, $\hat{t}$ is the *latest* departure time in order to get v before t (from z). Define h by

$$h_z(v,t) \;=\; \begin{cases} \max\{\tau^*(z,d,\hat{t}) - \tau^*(z,v,\hat{t}),\ 0\} & \text{if } \hat{t} \text{ exists,} \\ 0 & \text{otherwise (i.e., } \hat{t} \text{ does not exist).} \end{cases} \quad (3)$$

It is clear that $h_z(d,t) \equiv 0$ and $h_z \geq 0$. Actually this definition is a generalization from the static case, i.e., h_z is an estimation (a lower bound) on the v,d transit-time, which is no shorter than the right side of (3) (by the triangle inequality due to the optimality of τ^*). Moreover, we can show that h_z satisfies the FIFO Condition and the Triangle Condition at the same time, too. The proof is not trivial nor difficult, but due to the page limit, we omit it in this paper. We note it is important to choose $\hat{t}$ to be the *maximum*.

We still have to show how to calculate $\hat{t}$, which usually is difficult if there is no explicit expression for $\tau^*(z, v, t)$. Moreover, in general it is difficult to hold all values of $\tau^*(z, v, t)$. Fortunately, however, we can show that *sampling of time* works, i.e., we can calculate and hold values $\tau^*(z, v, t_i)$ only for some $t_1 < t_2 < \ldots < t_k$ and define $\hat{t}$, if it exists, by

$$\hat{t} \;=\; \max\{t_i : t_i + \tau^*(z, v, t_i) \le t\}.$$

Again, we can show the function h_z defined by (3) with the above $\hat{t}$ satisfies the FIFO Condition, the Triangle Condition, and $h_z(d, t) \equiv 0$, $h_z \ge 0$. Moreover, we can employ more than one landmarks to get a better estimation (notice the maximum of all h_zs works). Applying this generalized ALT algorithm to a number of US road networks (obtained from the web site of the 9th DIMACS implementation challenge `http://www.dis.uniroma1.it/~challenge9/`, where $320,000 \le n \le 1,210,000$ and $m \le 3n$) with periodic piecewise-linear transit-time functions (with 9 samples a day), we have noticed that it ran at an average of about 4 times faster than the generalized Dijkstra Algorithm with 16 landmarks and 2 time samplings. For details, we refer the reader to the thesis [11].

As an example, Figure 4 shows a comparison of the search space between the generalized Dijkstra algorithm and our ALT algorithm for an instance with $321,270$ nodes and $800,172$ edges. The number of landmarks is 16 and the number of time samplings is 2. The search space of ALT algorithm is 1/18.2 smaller and the running time is 7.4 times faster.

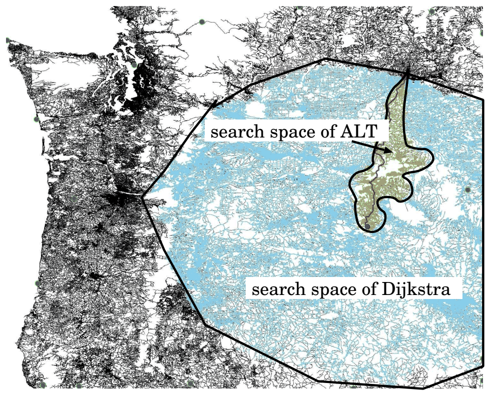

Figure 4: A comparison example of the search space between the generalized Dijkstra algorithm and the generalized ALT algorithm for the time-dependent shortest path problem.

4 Conclusion

In this paper, we have given a generalized framework of A* algorithm for the time-dependent shortest path problem. By constructing some appropriate estimator h, it is possible to get an algorithm that is faster than a normal generalized Dijkstra algorithm. As an example, we have generalized the landmark-based ALT algorithm, which we believe is the first algorithm that uses preprocessing to speed up the calculation of time-dependent shortest paths. Our experimental result shows it is several times faster than a normal generalized Dijkstra algorithm (which requires no preprocessing) for large road networks.

References

[1] R.K. Ahuja, T.L. Magnanti, J.B. Orlin, *Network flows: theory, algorithms, and applications*, Prentice-Hall (1993)

[2] K.L. Cook, E. Halsey, The shortest route through a network with time-dependent internodal transit, *J. Math. Anal. Appl.* (1966) **14**, 493–498

[3] B.C. Dean, Continuous-time dynamic shortest path algorithms, Master's thesis, MIT (1999)

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

[5] B. Ding, J.X. Xu, L. Qin, Finding time-dependent shortest paths over large graphs, *Proc. EDBT '08*, ACM Intl. Conf. Proc. (2008) **261**, 205–216

[6] A.V. Goldberg, C. Harrelson, Computing the shortest path: A* search meets graph theory, *Proc. SODA 2005* (2005), 156–165

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

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

[9] E. Kanoulas, Y. Du, T. Xia, D. Zhang, Finding fastest paths on a road network with speed patterns, *Proc. ICDE '06* (2006) 10–19

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

[11] T. Ohshima, A landmark algorithm for the time-dependent shortest path problem, Master's thesis, Graduate School of Informatics, Kyoto University (2008)

[12] A. Orda, R. Rom, Traveling without waiting in time-dependent networks is NP-hard, Manuscript, Dept. Electrical Engineering, Technion – Israel Institute of Technology, Haifa, Israel (1989)

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

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

[15] D. Wagner, T. Willhalm, Speed-up techniques for shortest-path computations, *Proc. STACS 2007* (2007) LNCS **4393**, 23–36