A Dynamic Discretization Discovery Algorithm for the Minimum Duration Time-Dependent Shortest Path Problem *

Edward He, Natashia Boland, George Nemhauser, and Martin Savelsbergh

H. Milton Steward School of Industrial & Systems Engineering, Georgia Institute of Technology, 755 Ferst Dr, Atlanta GA 30318

Abstract. We present an exact algorithm for the Minimum Duration Time-Dependent Shortest Path Problem with piecewise linear arc travel time functions. The algorithm iteratively refines a time-expanded network model, which allows for the computation of a lower and an upper bound, until - in a finite number of iterations - an optimal solution is obtained.

1 Introduction

Finding a shortest path between two locations in a network is a critical component of many algorithms for solving transportation problems. There is a growing interest in the setting where the travel time along an arc in the network is a function of the time the arc is entered. Time-dependent travel times are typically a result of congestion. We refer to these problems as Time-dependent Shortest Path Problems (TDSPPs). It is commonly assumed that travel times on arcs satisfy the First-In First-Out (FIFO) property, i.e., it is impossible to arrive at the end of the arc earlier by entering the arc later. Given a departure time at the source, the standard approach for finding a path that reaches the sink as early as possible is detailed in [1]. For an overview of other methods, see [2]. In this paper, we consider the problem of finding a path such that the difference between the departure time at the source and the arrival time at the sink is as small as possible. We call it the Minimum Duration Time-Dependent Shortest Path Problem (MD-TDSPP), sometimes referred to as the least travel time TDSPP or the minimum delay TDSPP. The MD-TDSPP arises in many contexts. It has been studied, for example, in the context of path planning in traffic networks [3], and it has even arisen in the analysis of social networks [4].

We present an efficient dynamic discretization discovery algorithm for the variant of MD-TDSPP in which travel times on the arcs are given by piecewise linear functions. It was established only recently that an algorithm polynomial in the number of travel time function breakpoints exists [5]. Our key contribution is the development of an algorithm that, in practice, investigates only a small

* This material is based upon work supported by the National Science Foundation under Grant No. 1662848.

fraction of the travel time function breakpoints in the search for an optimal path and the proof of its optimality. In Section 2, we formally introduce MD-TDSPP and briefly discuss the relevant literature. In Section 3, we describe our algorithm and illustrate it on a small instance. In Section 4, we present the results of a small computational study.

2 Problem Description

We are given a directed network $D = (N, A)$ with $N = \{1, 2, \ldots, n\}$ and $A \subseteq N \times N$, a time interval $[0, T]$, and piecewise linear travel times $c_{i,j}(t)$ for $t \in [0, T]$ satisfying the FIFO property for arcs $(i, j) \in A$. Without loss of generality, we let 1 be the source and n be the sink. Satisfying the FIFO property, in this case, is equivalent to having the slopes of the linear pieces being at least -1. (Note the FIFO property implies that waiting anywhere except at the source is sub-optimal, since it is always better to depart immediately.)

The MD-TDSPP is to find a starting time $0 \leq \tau \leq T$ and a time-dependent path $P(\tau) = (t_1, a_1, t_2, a_2, \ldots, a_{m-1}, t_m)$, which is a path $P = (a_1, a_2, \ldots, a_{m-1})$ from 1 to n in D and a set of associated departure times $(t_1 = \tau, t_2, \ldots, t_{m-1})$ and arrival time t_m at node n, where the departure times satisfy $t_k + c_{a_k}(t_k) = t_{k+1}$ for all $k = 1, \ldots, m-1$, meaning that the arrival time for one arc is the departure time of the next arc. Among all possible paths and starting times, $P(\tau)$ minimizes the duration, which is given by $t_m - t_1$. Furthermore, we require that $t_m \leq T$. We characterize time-dependent paths (TDPs) by their starting times as the problem of finding the minimum duration given a starting time is a TDSPP, which can be solved easily.

The MD-TDSPP has attracted much attention since the early work of Orda and Rom [6]. There are two classes of approaches: discrete and continuous. In the discrete approaches, a time-expanded network (TEN) is formed and the problem can be solved using the same method as for TDSPP. The DOT algorithm presented in [1] solves the TDSPP with complexity $\mathcal{O}(SSP + nM + mM)$, where SSP is the cost of solving a static SP, n is the number of nodes, m is the number of arcs, and M is the size of the time discretization. Discrete approaches are inexact and rely heavily on the quality of the discretization. A denser discretization leads to a better approximation, but an increase in computation time. Continuous methods, such as the Dijkstra's algorithm variants [7, 8], and the A* algorithm variant [9], create and update arrival time functions at each node and are exact. The complexity analysis of these methods has relied on being able to store and manipulate such functions efficiently and is given in terms of these operations, which are hard to quantify. Even for continuous piecewise linear functions, it was only recently that an algorithm that is polynomial in the total number of breakpoints (in the piecewise linear functions) was proposed [5]. The authors show that there is an optimal path that contains an arc (i, j) where the departure time occurs exactly at a breakpoint. Their algorithm investigates all arcs (i, j) and all its breakpoints t, solves the TDSPP from i to n starting at time t and the TDSPP from 1 to i ending at time t. The latter is done by

pre-computing the inverse costs (given an arc (i, j) and an arrival time t, what is the latest time to depart i so that we arrive at time t) so that we can solve the TDSPP from 1 to i ending at time t. If we let K be the total number of breakpoints in the network, then the complexity is $\mathcal{O}(K \times SSP)$. Such an approach performs many extraneous calculations due to its brute force nature. Our algorithm very significantly reduces the number of breakpoints investigated.

3 Dynamic Discretization Discovery Algorithm

Our algorithm is inspired by [10] and dynamically updates the discretization of a TEN. Any TEN allows the computation of lower and upper bounds on the duration of an optimal path. The lower and upper bounds are used to determine whether a minimum duration path has been found and, if not, for which parts of the TEN the time discretization should be refined.

We illustrate our ideas using the network in Figure 1 with travel time functions as given in Table 1. The time interval is $[0, 5]$, breakpoints for each arc are at every integer point, with the exception of arc $(3, 4)$ which only has breakpoints at $0, 1, 2, 5$. These values have been chosen to increase the visibility of the algorithm progression and to reduce the number of iterations.

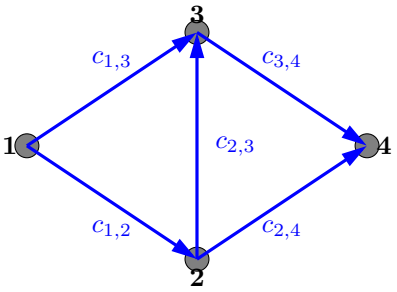


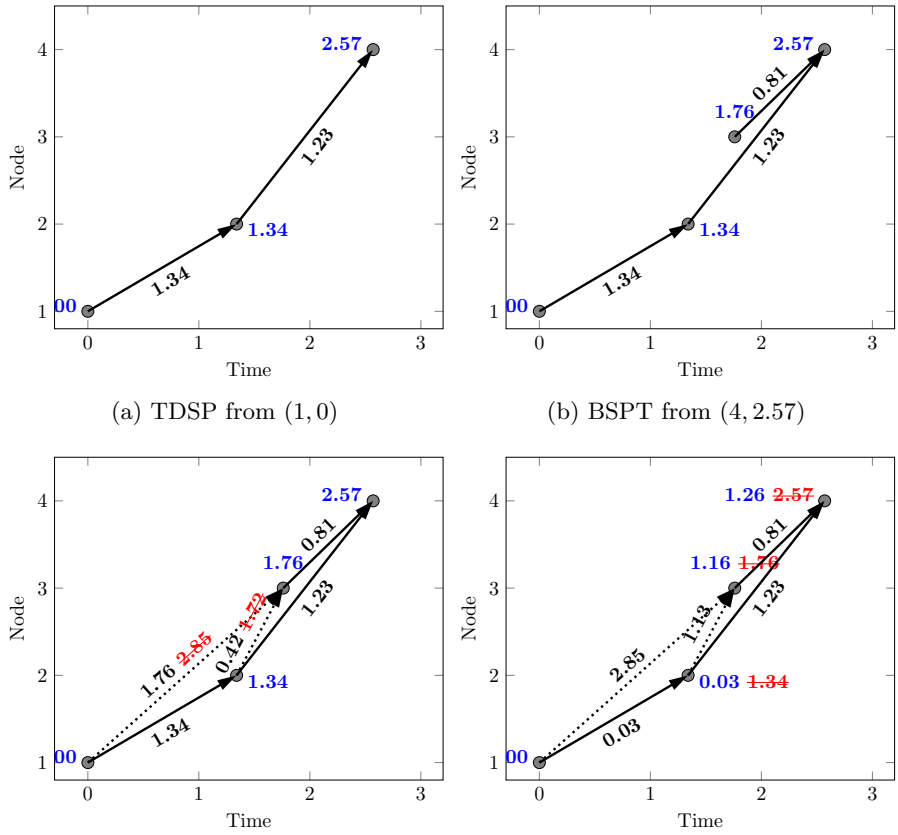
Fig. 1: Network D

BP Time	Arc Travel Times				
	(1,2)	(1,3)	(2,3)	(2,4)	(3,4)
0	1.34	2.85	1.99	1.29	0.61
1	0.66	2.95	1.82	1.02	0.73
2	0.14	3.00	1.51	1.63	0.83
3	0.01	2.98	1.10	2.57	—
4	0.35	2.90	0.67	3.00	—
5	1.00	2.76	0.30	2.54	1.00

Table 1: Arc Travel Times at Each Breakpoint (BP)

We maintain a set of *Arc-completed Backwards Shortest Path Trees* (ABSPTs), each denoted by $\mathcal{D}^{(k,t_k)}$, where $k \in N$ and $t_k \in [0, T]$, and is created by the following procedure. First, find a TDSP from (k, t_k) to n to obtain an arrival time t_n at n, see Figure 2a. Then, compute a time-dependent *backwards shortest path tree* (BSPT), giving node-time pairs (i, t_i) for each node in N, see Figure 2b. Finally, "arc-complete" the tree by adding an arc $((i, t_i), (j, t_j))$ for each arc $(i, j) \in A$ which is missing, see Figure 2c. Note that $\mathcal{D}^{(k,t_k)}$ can also be identified as $\mathcal{D}^{(i,t_i)}$, for any $(i, t_i) \in \mathcal{D}^{(k,t_k)}$, since the procedure starting at either (k, t_k) and (i, t_i) generates the same ABSPT. In particular, it is convenient to identify an ABSPT by its departure time at 1. By the FIFO property, the ABSPTs in a TEN have a natural chronological order (the ABSPTs can be sorted

in nondecreasing order of their departure time at 1). The benefit of working with ABSPTs is that any possible path can be represented in an ABSPT and an ABSPT can be used to compute a lower and upper bound on the duration of an optimal path.



(a) TDSP from $(1, 0)$

(b) BSPT from $(4, 2.57)$

(c) Arc-completed BSPT (actual travel times in red)

(d) Arc-completed BSPT with underestimated travel times (using $\mathcal{D}^{(4,5)}$) and node times (actual node times in red)

Fig. 2: Procedure to generate the ABSPT corresponding to $(1, 0)$ (departure times in blue; travel times in black).

Suppose we have at least two ABSPTs in a TEN. Let (i, t_i^1) be a node-time pair in one ABSPT and (i, t_i^2) be the node-time pair for i in the (chronologically) next ABSPT. Instead of actual travel times $c_{ij}(t_i^1)$ on $\mathcal{D}^{(i,t_i^1)}$, use the *underestimated travel times* (UTTs) given by $\underline{c}_{ij}(t_i^1) = \min_{t'} \{c_{ij}(t') \mid t_i^1 \leq t' \leq t_i^2\}$, see Figure 2d. It is easy to see that a shortest path from $(1, t_1^1)$ to (n, t_n^1) using the underestimated travel times gives a lower bound on a minimum duration path

departing in $[t_1^1, t_1^2)$, and that a shortest path from $(1, t_1^1)$ to (n, t_n^1) using the actual travel times gives an upper bound. The last ABSPT, which will always be $\mathcal{D}^{(n,T)}$ in our algorithm, is treated separately. Let $(1, t_1)$ be the node-time pair in $\mathcal{D}^{(n,T)}$ for 1. By the construction of $\mathcal{D}^{(n,T)}$, it is not possible to depart later than t_1 and arrive at n by time T, hence, we do not need to use underestimated travel times and instead keep the actual travel times for this particular ABSPT.

Given an ordered set L of ABSPTs, $L = (\mathcal{D}^{(1,t_1^1)}, \ldots, \mathcal{D}^{(1,t_1^p)})$, as well as associated lower and upper bounds, suppose that $\mathcal{D}^{(1,t_1^k)}$ contains the smallest lower bound. We choose to refine our time discretization by exploring the gap in the TEN between ABSPTs $\mathcal{D}^{(1,t_1^k)}$ and $\mathcal{D}^{(1,t_1^{k+1})}$. Adding an ABSPT corresponding to any node-time pair (i, t) such that $t_i^k < t < t_i^{k+1}$ and updating the underestimated travel times for $\mathcal{D}^{(1,t_1^k)}$ (since the next ABSPT is no longer $\mathcal{D}^{(1,t_1^{k+1})}$) may improve the lower bound, since the interval used to calculate the underestimated travel times has shortened.

The concepts presented so far can be used to devise an algorithm that converges to an optimal path, but not enough to ensure finite termination. Finite termination can be achieved by exploiting the fact that there exists an optimal path that contains a departure at a node i at time t for some arc (i, j) that has a breakpoint at time t (see [5]). Therefore, we only create ABSPTs that contain at least one node-time pair (i, t) corresponding to a breakpoint. Since there are a finite number of breakpoints, this ensures finite termination. To achieve efficiency, we exploit the fact that the arrival time function at n for departures between t_1^1 and t_1^2 is concave if no shortest path that departs between t_1^1 and t_1^2 contains a breakpoint (also shown in [5]). This situation occurs when there are no more breakpoints remaining in the gap between two ABSPTs $\mathcal{D}^{(1,t_1^k)}$ and $\mathcal{D}^{(1,t_1^{k+1})}$, and we know that the minimum duration path departing between t_1^k and t_1^{k+1} departs at either t_1^k or t_1^{k+1}, both of which have already been calculated as an upper bound, hence the lower bound for $\mathcal{D}^{(1,t_1^k)}$ can be updated to one of these upper bounds and thus no longer needs to be considered. This gives us an additional termination criterion: we can terminate when the smallest lower bound among the ABSPTs still being under consideration is larger than the best upper bound obtained so far. A high-level overview of our algorithm can be found in Algorithm 1.

The algorithm explores breakpoints. We choose to look for a breakpoint t in the travel time function of arc (i, j) such that i is minimized, then j is minimized, and t is the median among the breakpoints. The performance of the algorithm depends greatly on being able to efficiently compute the minimum arc travel time in a departure time interval. This is accomplished by (efficiently) pre-computing a look-up table that gives the next local minimum for any breakpoint.

Next, we illustrate the algorithm on the example; see Figure 3. The algorithm is initialized with $\mathcal{D}^{(1,0)}, \mathcal{D}^{(4,5)}$, which are generated by the paths $P = ((1, 2), (2, 4))$ and $P = ((1, 2), (2, 3), (3, 4))$, respectively.

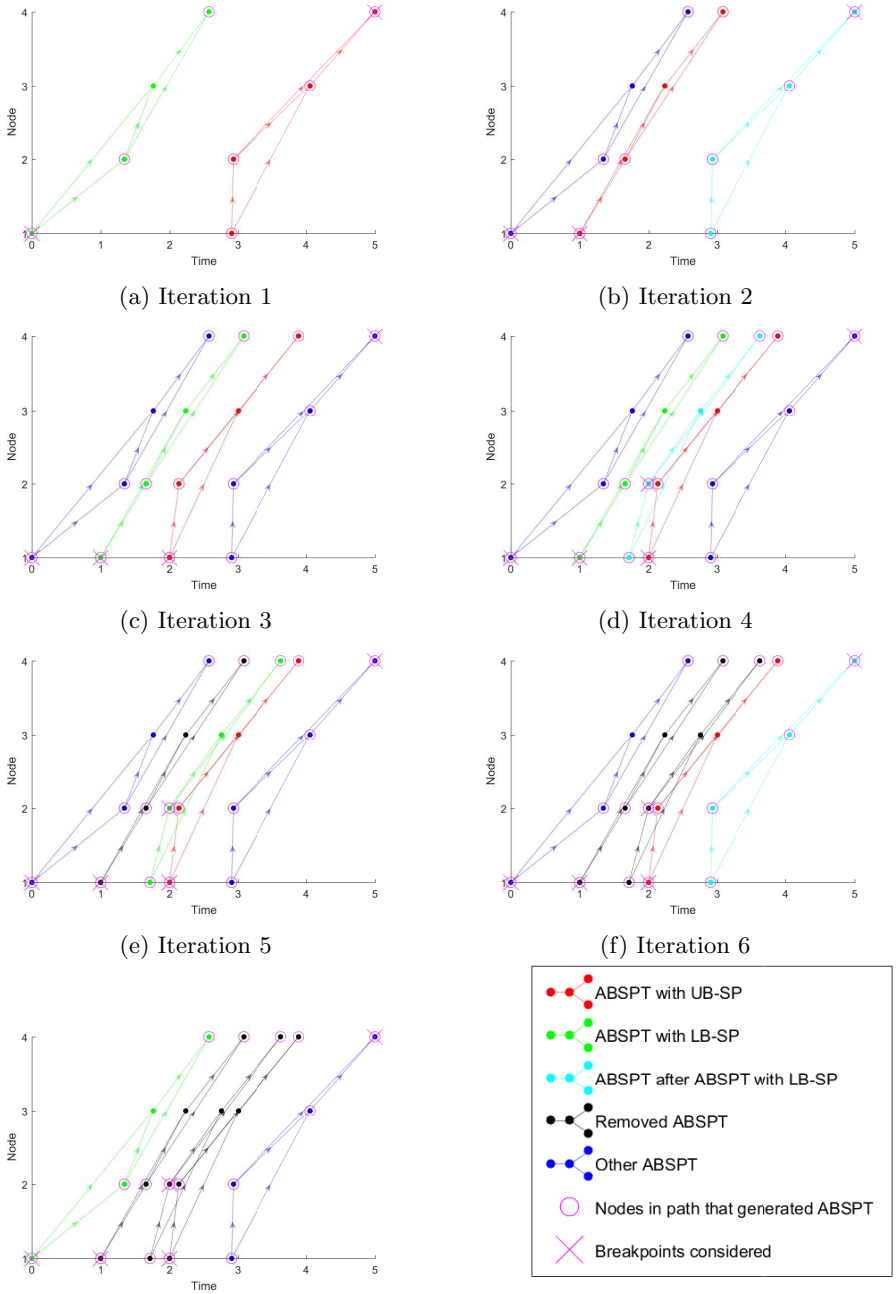
In Iteration 1, since $\mathcal{D}^{(1,0)}$ gives the lower bound, look at the section of the TEN succeeding $\mathcal{D}^{(1,0)}$, and observe that arc $(1, 2)$ has a breakpoint at $t = 1$, so we add $\mathcal{D}^{(1,1)}$. It turns out that $\mathcal{D}^{(1,1)}$ has UB= 2.0804 and LB= 1.4470. Since

Algorithm 1: Dynamic Discretization Discovery (DDD) Algorithm.
input : $G = (N, A)$, $c_{i,j}(t)$, T
output: minimum duration shortest path
$L \leftarrow (\mathcal{D}^{(1,0)}, \mathcal{D}^{(n,T)})$;
$UB \leftarrow \min\{computeUB(\mathcal{D}^{(1,0)}), computeUB(\mathcal{D}^{(n,T)})\}$;
$LB \leftarrow computeLB(\mathcal{D}^{(1,0)})$;
$\mathcal{D}^{(1,t_1^k)} \leftarrow \mathcal{D}^{(1,0)}$;
while $(LB < UB)$ **do**
  if *there is a breakpoint* (j, τ) *between* $\mathcal{D}^{(1,t_1^k)}$ *and* $\mathcal{D}^{(1,t_1^{k+1})}$ **then**
    if $computeUB(\mathcal{D}^{(j,\tau)}) < UB$ **then**
      $UB \leftarrow UB(\mathcal{D}^{(j,\tau)})$
    end
    $recomputeLB(\mathcal{D}^{(1,t_1^k)})$;
    $computeLB(\mathcal{D}^{(j,\tau)})$; $insert(L, \mathcal{D}^{(j,\tau)})$;
  else
    $LB(\mathcal{D}^{(1,t_1^k)}) = UB(\mathcal{D}^{(1,t_1^k)})$;
  end
  $LB \leftarrow updateLB(L)$;
  $\mathcal{D}^{(1,t_1^k)} \leftarrow getBestLB(L)$;
end

$LB < UB$, we continue with the algorithm. We proceed to add $\mathcal{D}^{(1,2)}$ and $\mathcal{D}^{(2,2)}$. In Iteration 4, $\mathcal{D}^{(1,1)}$ contains the LB, however, since there are no breakpoints in the succeeding section, we replace the LB of $\mathcal{D}^{(1,1)}$ with its UB. The new LB is contained in $\mathcal{D}^{(2,2)}$. We proceed to replace the LB of $\mathcal{D}^{(2,2)}$ and $\mathcal{D}^{(1,2)}$ with their UB due to the lack of breakpoints in the succeeding sections, at which stage UB= 1.8886 is less than LB= 1.8888 and hence the algorithm terminates. The optimal path is the one that corresponds to UB, which was found when $\mathcal{D}^{(1,2)}$ was created and is $((1,2),(2,4))$ starting at time 2.

4 Computational Study

To analyze the performance of the DDD algorithm, we apply it to several randomly generated instances. In particular, we solve 10 instances with $n = 20$ and $T = 200$ and 10 instances with $n = 30$ and $T = 200$. The instances are generated as follows. The arc set A consists of all pairs of nodes (i, j) where $i < j$. The travel time on arc (i, j) is the piecewise linear interpolant of the function $f_{i,j}(t) = (j-i)+sin(b_{i,j} \times t)$ at each integer point from 0 to T, so that there are $T + 1$ breakpoints, where $b_{i,j}$ is a random number between 0 and 1 generated using a pre-specified random seed and the Matlab 'twister' RNG. The function f is designed so that $c_{i,j}$ satisfies the FIFO property (since the slope is always greater than -1). In addition, due to the additional constant $(j - i)$, the optimal path is likely to use many arcs. Note that we choose $T \geq n$ to avoid the



(a) Iteration 1 (b) Iteration 2

(c) Iteration 3 (d) Iteration 4

(e) Iteration 5 (f) Iteration 6

(g) Iteration 7 (h) Legend

Fig. 3: Time-expanded network in each iteration for the example in Figure 1 (note that a ABSPT may satisfy multiple criteria in the legend).

possibility of having no feasible TDSP. To analyze the impact of the number of breakpoints on the performance of the DDD algorithm, we stretch the horizon T and the function $f_{i,j}(t)$; we multiply T by a factor $S = 2.5$ and $= 5$. In Table 2, we report the minimum, the average, and the maximum number of breakpoints investigated by our algorithm over the instances, the total number of breakpoints in the instance (which is the number of breakpoints that the enumeration algorithm investigates [5]), and the fraction of the total number of breakpoints investigated by the DDD algorithm. Furthermore, we report the solve times of the DDD algorithm and the enumeration algorithm and the ratio of these solves times. Note that the total number of breakpoints is $|N-1| \times (T+1)$ not $|A| \times (T+1)$, as in [5], since for arcs with a common tail node and breakpoint, we only need to investigate the breakpoint once.

Table 2: Computational results.

n	S	BP			Total	%BP	Avg. Time	Avg. Time	Ratio
		Min.	Avg.	Max.	#BP		DDD	Enum.	
20	1	112	169.7	224	3800	4.47	103.4	55.8	1.86
	2.5	127	175.2	229	9500	1.84	118.3	218.9	0.54
	5	139	185.0	242	19000	0.97	136.2	690.0	0.20
30	1	182	229.4	268	5800	3.96	290.6	170.4	1.71
	2.5	165	243.9	303	14500	1.68	376.7	774.9	0.49
	5	179	261.8	328	29000	0.90	439.5	2493.0	0.18

The results show clearly that the DDD algorithm investigates only a small fraction of the total number of breakpoints. In addition, the fraction decreases when both n and T increase. For finer discretizations of time, the DDD algorithm significantly outperforms the enumeration algorithm in terms of solve times as well even though it has not been optimized for efficiency.

5 Concluding Remarks

We have shown that dynamic discretization discovery concepts can dramatically reduce the number of breakpoints explored when solving MD-TDSPP instances. Preliminary computational results show that our method scales well in both the number of nodes and number of breakpoints. Next, we will explore extending these ideas to other types of transportation problems, e.g., the Time-Dependent Traveling Salesman Problem.

Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant No. 1662848.

References

1. Chabini, I.: Discrete dynamic shortest path problems in transportation applications: Complexity and algorithms with optimal run time. Transportation Research Record: Journal of the Transportation Research Board (1645) (1998) 170–175
2. Dean, B.C.: Shortest paths in fifo time-dependent networks: Theory and algorithms. Rapport technique, Massachusetts Institute of Technology (2004)
3. Demiryurek, U., Banaei-Kashani, F., Shahabi, C., Ranganathan, A.: Online computation of fastest path in time-dependent spatial networks. In: International Symposium on Spatial and Temporal Databases, Springer (2011) 92–111
4. Carley, M.: Information lifetime aware analysis for dynamic social networks venkata mv gunturi, kenneth joseph, shashi shekhar, kathleen. Technical report, University of Minnesota (2012)
5. Foschini, L., Hershberger, J., Suri, S.: On the complexity of time-dependent shortest paths. Algorithmica **68**(4) (2014) 1075–1097
6. Orda, A., Rom, R.: Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length. Journal of the ACM (JACM) **37**(3) (1990) 607–625
7. Nachtigall, K.: Time depending shortest-path problems with applications to railway networks. European Journal of Operational Research **83**(1) (1995) 154–166
8. Ding, B., Yu, J.X., Qin, L.: Finding time-dependent shortest paths over large graphs. In: Proceedings of the 11th international conference on Extending database technology: Advances in database technology, ACM (2008) 205–216
9. Kanoulas, E., Du, Y., Xia, T., Zhang, D.: Finding fastest paths on a road network with speed patterns. In: Data Engineering, 2006. ICDE'06. Proceedings of the 22nd International Conference on, IEEE (2006) 10–10
10. Boland, N., Hewitt, M., Marshall, L., Savelsbergh, M.: The continuous-time service network design problem. Operations Research **65**(5) (2017) 1303–1321