Dynamic Discretization Discovery Algorithms for Time-Dependent Shortest Path Problems

Edward He, Natashia Boland, George Nemhauser, Martin Savelsbergh

H. Milton Stewart School of Industrial and Systems Engineering
Georgia Institute of Technology

Abstract

Finding a shortest path in a network is a fundamental optimization problem. We focus on settings in which the travel time on an arc in the network depends on the time at which traversal of the arc begins. In such settings, reaching the sink as early as possible is not the only objective of interest. Minimizing the duration of the path, i.e., the difference between the arrival time at the sink and the departure from the depot, and minimizing the travel time along the path from source to sink, are also of interest.

We introduce dynamic discretization discovery algorithms to efficiently solve such time-dependent shortest path problems with piecewise linear arc travel time functions. The algorithms operate on partially time-expanded networks in which arc costs represent lower bounds on the arc travel time over the subsequent time interval. A shortest path in this partially time-expanded network yields a lower bound on the value of an optimal path. Upper bounds are easily obtained as byproducts of the lower bound calculations. The algorithms iteratively refine the discretization by exploiting breakpoints of the arc travel time functions. In addition to time discretization refinement, the algorithms permit time intervals to be eliminated, improving lower and upper bounds, until, in a finite number of iterations, optimality is proved. Computational experiments show that only a small fraction of breakpoints must be explored, and that the fraction decreases as the length of the time horizon and the size of the network increases, making the algorithms highly efficient and scalable.

keywords: minimum duration paths, time-dependent travel times, fastest paths, time-expanded networks, time discretization, dynamic discretization discovery

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 growing interest in settings in which the travel time along an arc in the network is a function of the time the travel starts. Such *time-dependent travel times* are typically a result of congestion. As is commonly done, we assume that travel times on arcs satisfy the First-In First-Out (FIFO) property: it is impossible to arrive at the end of an arc earlier by starting travel along the arc later.

Several different objective functions arise when seeking a path in a network with time-dependent travel times. In the simplest version of the problem, the departure time at the source is given and a path that reaches the sink as early as possible is sought. We refer to this as the Minimum *Arrival Time* Path Problem (MATP). The first approach developed to solve this problem is based on the Bellman-Ford algorithm (Bellman 1958, Ford and Fulkerson 1962) and is described by Cooke and Halsey (1966). An overview of other methods for this variant is given by Dean (2004).

In this paper, we focus on two different variants: (1) 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, and (2) the problem of finding a path such that the total travel time from the source to the sink is as small as possible. We refer to these variants as the Minimum *Duration* Time-Dependent Path Problem (MDP) and the Minimum *Travel Time* Time-Dependent Path Problem (MTTP), respectively. Both of these problems allow waiting at nodes in the network, which makes them much more complicated than the MATP.

The MDP arises in many contexts. It has been studied, for example, in the context of traffic networks (Demiryurek et al. 2011), and it has even arisen in the analysis of social networks (Gunturi et al. 2012). The MTTP is especially relevant when seeking to reduce emissions in urban environments, since a vehicle's emissions along a given section of road are approximately proportional to its travel time on that section, for the range of speeds typically used in urban settings (see, for example, the emissions formula used by Jabali et al. (2012) in the time-dependent vehicle routing context).

Recently, Boland et al. (2017) introduced a dynamic discretization discovery algorithm for solving the continuous time minimum cost service network design problem, which uses integer programming formulations over partially time-expanded networks. The key to the approach is that it discovers, in an efficient way, times to include in the time-expanded network so that the associated integer program yields a solution that is optimal for the continuous time problem. It does so by solving a sequence of (small) integer programs, each a function of the subset of times used to define the partially time-expanded network. The partially time-expanded networks are carefully designed to result in integer programs that are tractable in practice, and that yield a dual (lower) bound on the optimal continuous-time value. Once the *right* (very small) subset of times is discovered, the resulting integer programming model yields the continuous-time optimal value.

In this paper, we present and analyze dynamic discretization discovery algorithms for the MDP and the MTTP when the travel times on arcs are given by piecewise linear functions. The algorithms solve a sequence of shortest path problems in partially time-expanded networks, each of which provides a successively better lower bound on the value of the continuous-time problem. The time-space nodes in the partially time-expanded networks are derived from breakpoints of the piecewise linear travel time functions. The shortest path at each iteration suggests new times to include in the time-expanded network, and its true objective value (its duration or travel time) can easily be calculated to yield an upper bound on the value of the continuous-time problem. The sequence concludes when the length of the shortest path in the current partially time-expanded network matches the best upper bound.

For the MDP, it was established only recently that an algorithm polynomial in the number of travel time function breakpoints exists (Foschini et al. 2014). We extend this result to the MTTP. However, our key contribution is the development of algorithms that, through dynamic discretization discovery, investigate only a small fraction of the travel time function breakpoints in the search for an optimal path and the proof of its optimality.

The remainder of the paper is organized as follows. In Section 2, we formally introduce the MDP and MTTP and briefly discuss the relevant literature. In Section 3, we describe our algorithm for the MDP and illustrate its operation on a small instance. We then present an algorithm for the MTTP in Section 4. In Section 5, we present the results of a comprehensive computational study.

2 Problem Description

We are given a directed network $D = (N, A)$ with $N = \{1, 2, \dots, n\}$ and $A \subseteq N \times N$, a time interval $[0, T]$, and piecewise linear travel times $c_{i,j}(t) > 0$ for $t \in [0, T]$ satisfying the FIFO property for arcs $(i, j) \in A$. Satisfying the FIFO property, in the case of piecewise linear travel time functions, is equivalent to having the slopes of the linear pieces be at least -1. Without loss of generality, we let node 1 be the source and node n be the sink.

A *timed path* $P = ((i_1^P, t_1^P), (i_2^P, t_2^P), \dots, (i_{m(P)}^P, t_{m(P)}^P))$ consists of a sequence of $m(P)$ node and time pairs, where its node sequence, $(i_1^P, i_2^P, \dots, i_{m(P)}^P)$, induces a path in D from i_1^P to $i_{m(P)}^P$, and where each node in the sequence, i_k^P, has an associated time, t_k^P, representing the time at which the traversal of arc (i_k^P, i_{k+1}^P) begins, for each $k = 1, \dots, m(P) - 1$, and the arrival time at node i_k^P, for $k = m(P)$. The sequence of times is thus required to satisfy $t_{k+1}^P \geq t_k^P + c_{i_k^P, i_{k+1}^P}(t_k^P)$ for $k = 1, \dots, m(P) - 1$. If $t_k^P > t_{k-1}^P + c_{i_{k-1}^P, i_k^P}(t_{k-1}^P)$ for any $k \geq 2$, (or if $t_1^P > 0$), then we say that there is waiting at node i_k^P. If $t_k^P = t_{k-1}^P + c_{i_{k-1}^P, i_k^P}(t_{k-1}^P)$ for all $k \geq 2$ then we say that P is *waiting-free*. If P starts at node $i_1^P = 1$ with $t_1^P \geq 0$ and ends at node $i_{m(P)}^P = n$

with $t^P_{m(P)} \leq T$ then we say that P is a *feasible* timed path. Let $\mathcal{P}$ denote the set of all feasible timed paths.

Three objectives are of interest:

1. minimizing the arrival time at the sink, $\min_{P \in \mathcal{P}} t^P_{m(P)}$,
2. minimizing the duration of the path from source to sink, $\min_{P \in \mathcal{P}} (t^P_{m(P)} - t^P_1)$, and
3. minimizing the travel time on the path from source to sink, $\min_{P \in \mathcal{P}} \sum_{k=1}^{m(P)-1} c_{i^P_k, i^P_{k+1}}(t^P_k)$.

In the rest of the paper, for notational convenience, we drop the P in superscripts.

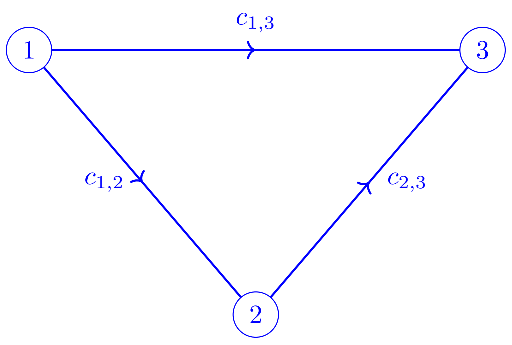
To illustrate these three objectives, consider the network and arc travel time functions shown in Figure 1, with $n = 3$ and given time horizon $[0, 4]$. The timed path minimizing arrival time at the sink is $((1, 0), (3, 3.5))$: the path would traverse arc $(1, 3)$, starting at $t = 0$ and arriving at $t = 3.5$, giving an objective value of 3.5. The timed path of minimum duration is $((1, 1.5), (2, 3), (3, 4))$: the path would traverse arc $(1, 2)$, departing at $t = 1.5$ and arriving at $t = 3$, and then traverse arc $(2, 3)$, departing at $t = 3$ and arriving at $t = 4$, giving an objective value of 2.5. As there was no waiting on this path, it also has a travel time of 2.5. However, this is not the minimum travel time. There are multiple optimal solutions for the problem of minimizing the travel time. For any $t_1 \in [0, 1]$, the timed path $((1, t_1), (2, 3), (3, 4))$ has minimum travel time: it would traverse arc $(1, 2)$, departing at t_1 and arriving at $t_1 + 1 \in [1, 2]$, then wait at node 2 until $t = 3$, and then traverse arc $(2, 3)$, departing at $t = 3$ and arriving at $t = 4$, giving an objective value of 2.

Note that because the travel time functions on the arcs have the FIFO property, when minimizing the arrival time at the sink or the duration of the path from source to sink, an optimal path will have $t_k + c_{i_k, i_{k+1}}(t_k) = t_{k+1}$ for $k = 1, \ldots, m(P) - 1$. In other words, there is no benefit in waiting at intermediate nodes and there must be an optimal solution that is waiting-free. Furthermore, when minimizing the arrival time at the sink, an optimal path will have $t^P_1 = 0$.

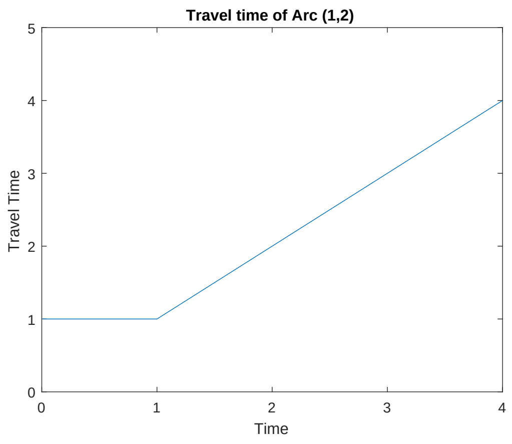
Finding a timed path that reaches the sink as early as possible, given a fixed start time at the source, can be done, in polynomial time, with straightforward extensions of algorithms for the standard shortest path problem (Cooke and Halsey 1966, Orda and Rom 1990, Dean 2004). Finding a timed path that departs the source as late as possible, given a fixed end time at the sink, can be solved similarly. This is done by pre-computing *reverse* travel time functions: given an arc (i, j) and an arrival time t, the reverse function for (i, j) evaluated at t gives the travel time τ so that starting traversal of (i, j) at time $t - \tau$ results in arrival at j at time t. In what follows, we refer to a problem having a given, fixed, start or end time, as a Time-Dependent Shortest Path Problem (TDSPP), and an optimal solution to the problem as a *time-dependent shortest path* (TDSP). As is the case in finding standard shortest paths, algorithms solving a TDSPP can just as easily provide either the forwards or the backwards shortest path *tree*. Solving a TDSPP is a key subroutine in algorithms for more complex settings.

The MDP has attracted much attention since the early work of Orda and Rom (1990). There are two classes of approach: discrete and continuous. In discrete approaches, such as that of Chabini (1998), time is discretized. Travel along an arc may only start at a time point in the discretization and the arrival time at the end of the arc is mapped, in some way, to a time point in the discretization. In other words, the travel time on an arc is forced to conform to the time discretization. Discrete approaches are thus 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 in Orda and Rom (1990), Nachtigall (1995), Ding et al. (2008), and the A* algorithm variant in Kanoulas et al. (2006), create and update arrival time functions at each node, and are exact. The *arrival time function* at a node takes as input the time of departure at the source and gives the earliest time of arrival at the node for any path departing from the source at the given time. In other words, the arrival time function at a node gives the value of the TDSPP with this node as the sink, for every possible start time at the source. The complexity analysis of these methods is given in terms of operations that store and manipulate arrival time functions. However, the analysis does not give complexity of these operations in terms of the problem parameters.

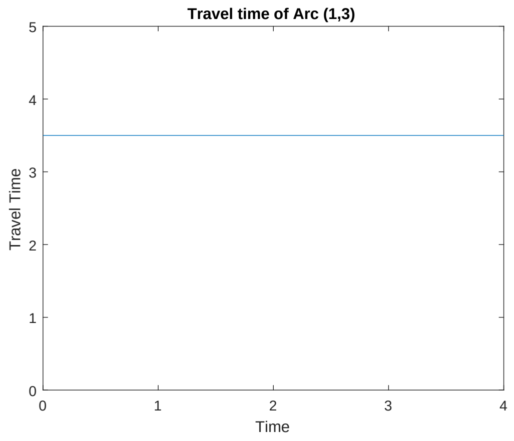
Even for continuous piecewise linear functions, it was only recently that an exact algorithm that is polynomial in the total number of breakpoints (in the piecewise linear functions) was proposed (Foschini



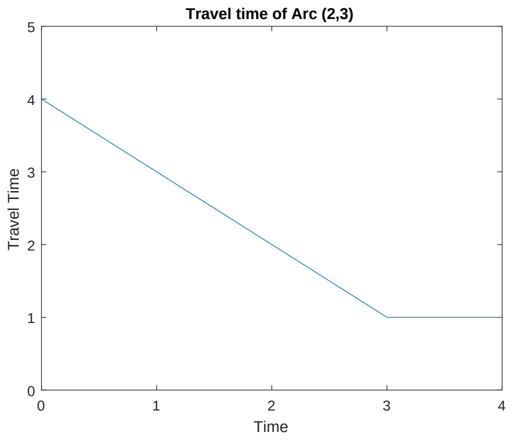
(a) Example Network 1



(b) Travel Time of Arc (1,2)



(c) Travel Time of Arc (1,3)



(d) Travel Time of Arc (2,3)

Figure 1: Example to illustrate the differences between the three objectives.

et al. 2014). Furthermore, as far as we are aware, this is the *only* such algorithm. The authors prove the intuitive result that there is an optimal path that either starts its traversal of some arc exactly at a breakpoint of the arc's travel time function, or starts at node 1 at time $t = 0$ or ends at node n at time $t = T$. Their algorithm investigates all arcs (i, j) and all breakpoints, t, of the function $c_{i,j}$, solves the TDSPP from i to n starting at time t and the TDSPP from 1 to i ending at time t. Concatenating the two resulting paths yields a feasible timed path, and one such path must be optimal. Foschini et al. (2014) observe that this algorithm has computational complexity $\mathcal{O}(K^{\text{arcs}} \times SP)$, where SP is the complexity of solving a time-dependent shortest path problem (with a fixed starting time) in the given network and $K^{\text{arcs}} = \sum_{(i,j)\in A} k_{i,j}$ is the total number of breakpoints, where $k_{i,j}$ is the number of breakpoints in the function $c_{i,j}(\cdot)$, for each $(i, j) \in A$. We note here that, in fact, if arcs with the same tail node have common breakpoints in their travel time functions, the complexity is $\mathcal{O}(K^{\text{nodes}} \times SP)$ for $K^{\text{nodes}} < K^{\text{arcs}}$ given by

$$K^{\text{nodes}} = \sum_{i\in N} \left| \bigcup_{j:\,(i,j)\in A} B_{i,j} \right|,$$

where $B_{i,j}$ is the set of breakpoints of $c_{i,j}(t)$ over $t \in [0, T]$. It is then natural to consider breakpoints as associated with nodes, where if the travel time function of arc (i, j) has a breakpoint at time t, we say that *node i has a breakpoint at time t*. We may also say that (i, t) *is a breakpoint*. In addition, we shall consider $(1, 0)$ and (n, T) to also be breakpoints.

As we show in Section 4, the ideas of Foschini et al. (2014) can be extended to the MTTP, and used to define an algorithm for the MTTP that is polynomial in the total number of breakpoints. This is in contrast to the recent work presented in a research report of Omer and Poss (2018), who show that in the presence of a constraint on the waiting time at nodes, the problem is NP-hard.

In this paper, we develop dynamic discretization discovery algorithms for MDP and MTTP that investigate only a very small fraction of the total number of breakpoints, while solving the problem to proven optimality.

3 A Dynamic Discretization Discovery Algorithm for the MDP

For ease of exposition, this paper focuses only on the case that the FIFO property holds *strictly*. Only minor modifications to the algorithm and proof of correctness are needed in the case the FIFO property is not strict.

We first provide some preliminary definitions, illustrated using the instance given by the network in Figure 2 with the travel time functions whose graphs are shown in Figure 4. 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 $0, 1, 2$ and 5. The arc travel time functions and their reverse functions are stated in Appendix 8.

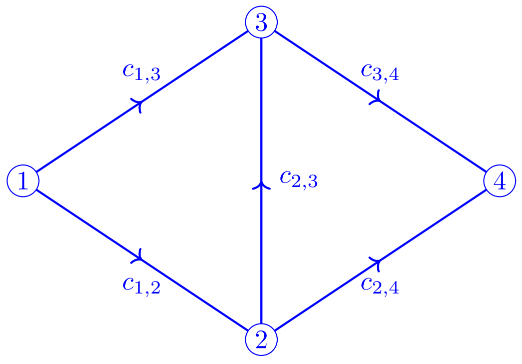
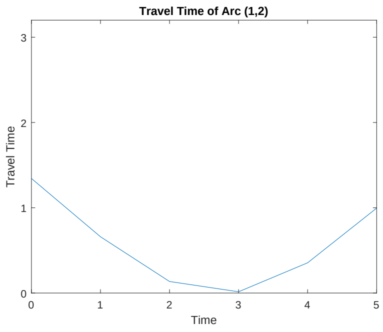


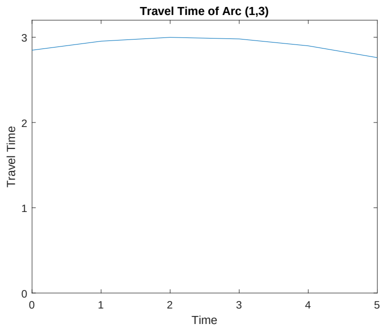
Figure 2: Network D

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

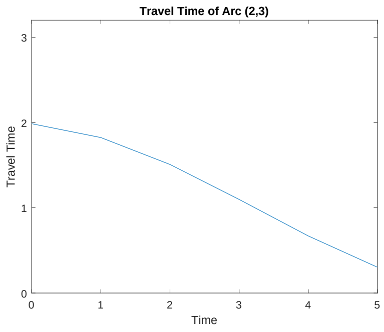
Table. 3: Arc Travel Times at Each Breakpoint (BP)



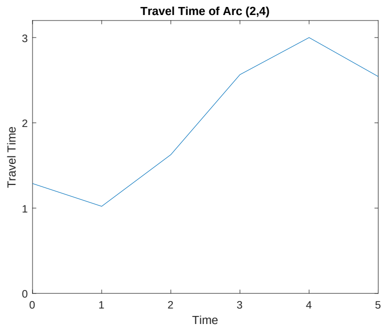
(a) Travel Time of Arc (1,2)



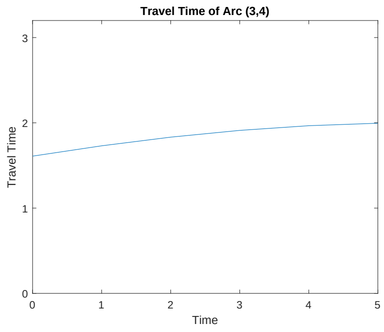
(b) Travel Time of Arc (1,3)



(c) Travel Time of Arc (2,3)



(d) Travel Time of Arc (2,4)



(e) Travel Time of Arc (3,4)

Figure 4: Arc Travel Times

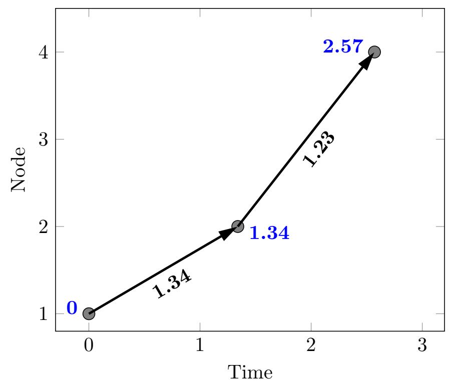
Given a time $t \in [0, T]$ that represents a specific arrival time at node n, a corresponding *Backwards Shortest Path Tree* (BSPT) is a time-expanded network (TEN), denoted by $\mathcal{B}^t$, which is an in-tree rooted at (n, t) defined by a set of timed nodes of the form $(i, t_i) \in N \times (-\infty, T]$ and timed arcs of the form $((i, t_i), (j, t_j))$ satisfying:

- for each $i \in N$ such that n is reachable from i, there is exactly one time, t_i, for which the timed node (i, t_i) is in $\mathcal{B}^t$,
- t_i is the latest departure time from i that allows arrival at n at time t,
- $(i, j) \in A$,
- $t_i + c_{i,j}(t_i) = t_j$, and
- there is a unique path from (i, t_i) to (n, t) in $\mathcal{B}^t$, induced by the solution to the TDSPP from source i starting at time t_i to sink n.

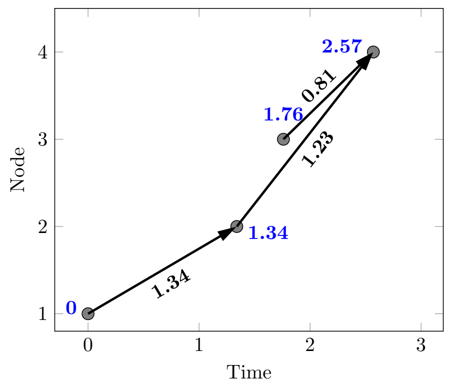
As mentioned in the previous section, the TDSPP can be solved, and hence a BSPT found, easily, using straightforward adaptations of standard shortest path algorithms.

The BSPT for node 4 at $t = 2.57$ in the example is shown in Figure 5(b). It is the TEN with nodes $\{(1, 0.00), (2, 1.34), (3, 1.76), (4, 2.57)\}$, (times are accurate to two decimal places), and arcs given by the unique timed copies of the arcs $(1, 2)$, $(2, 4)$ and $(3, 4)$ induced by the timed nodes.

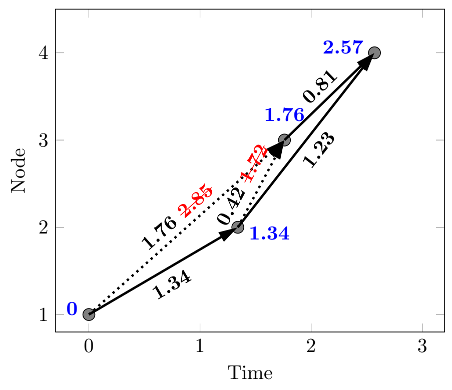
Note that for a given $t \in [0, T]$, in constructing $\mathcal{B}^t$, it may be that for some i it is impossible to reach n from i by time t for any positive departure time at i. Of course, if this is the case for $t = T$, the node i can simply be eliminated from the network in a preprocessing step. Otherwise, for simplicity of exposition in what follows, we ensure that a BSPT includes a timed node for every $i \in N$ by extending $c_{i,j}(t)$ to $t < 0$:



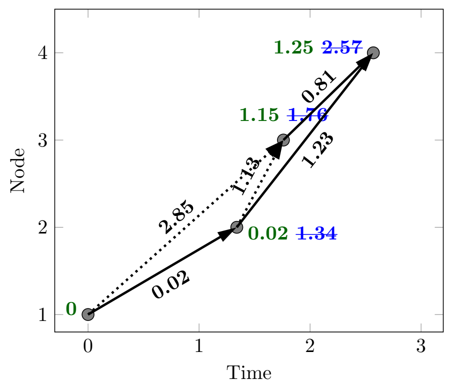
(a) TDSP from $(1, 0)$ (timed node times in blue, travel times in black)



(b) BSPT from $(4, 2.57)$ (timed node times in blue, travel times in black)



(c) ABSPT $\overline{B}^{2.57}$ (actual travel times in red, timed node times in blue, travel times implied by head and tail node times in black)



(d) ABSPT $\overline{B}^{2.57}$ with UTTs calculated with respect to $\mathcal{B}^5$ in black, timed node times in blue and shortest path node labels in green

Figure 5: Procedure to generate the ABSPT corresponding to $(1, 0)$.

we set $c_{i,j}(t) = c_{i,j}(0)$ for all $t < 0$ and all $(i,j) \in A$ and permit timed nodes of the form (i, t_i) with $t_i < 0$ to be included in $\mathcal{B}^t$.

Our dynamic discretization discovery algorithm is based on the following key observations about BSPTs. First, it is clear from the FIFO property that if (i, s) is a node in BSPT $\mathcal{B}^t$ and (i, s') is a node in BSPT $\mathcal{B}^{t'}$ with $t' > t$, then $s' > s$. Second, any minimum duration timed path from 1 to n that arrives at node n at time t or later has a *representative* timed node sequence in $\mathcal{B}^t$, meaning that for each node-time pair (i, s) in the path, there is a timed node (i, s') in $\mathcal{B}^t$, which is not later: $s' \leq s$. Note that such a representative timed node sequence in $\mathcal{B}^t$ need not follow timed arcs in $\mathcal{B}^t$. This motivates the following definition.

An *Arc-completed Backwards Shortest Path Tree* (ABSPT) is the TEN formed from a given BSPT by adding timed arcs, $((i, t_i), (j, t_j))$, for every $(i,j) \in A$ having both (i, t_i) and (j, t_j) in the BSPT. By the definition of a BSPT, it must be that $t_i + c_{i,j}(t_i) \geq t_j$ for all such arcs in the ABSPT. In other words, $t_j - t_i \leq c_{i,j}(t_i)$ for all $((i, t_i), (j, t_j))$ in an ABSPT. The ABSPT for the example of $\mathcal{B}^t$ with $t = 2.57$ is shown in Figure 5(c), with the values of $c_{i,j}(t_i)$ shown crossed out in red, and replaced by the value $t_j - t_i$, on all new arcs added to form the ABSPT. We use the notation $\overline{\mathcal{B}}^t$ to denote the ABSPT formed from $\mathcal{B}^t$.

Our algorithm works by maintaining a *list* of ABSPTs, ordered by their time at the end node. The list is initialized with two ABSPTs: the ABSPT for the earliest time that the end node can be reached and the ABSPT with end time given by the end of the time horizon. These two times define the time interval of interest: all feasible timed paths must arrive at the end node within this interval. Additional ABSPTs are generated dynamically, subdividing this interval. In the example, the initial ABSPTs are $\overline{\mathcal{B}}^{2.57}$ and $\overline{\mathcal{B}}^5$ (since $T = 5$). The latter has timed nodes $(1, 2.90)$, $(2, 2.92)$, $(3, 4.05)$ and $(4, 5)$.

The algorithm relies on the following third, and final, key observation, which concerns two ABSPTs that appear *consecutively* in the list. Say $\overline{\mathcal{B}}^t$ and $\overline{\mathcal{B}}^{t^+}$, whose end times t and t^+, respectively, satisfy $t^+ > t$, are two such ABSPTs. For each arc $((i, s_i), (j, s_j))$ in the earlier ABSPT, $\overline{\mathcal{B}}^t$, we define its *underestimated travel time* (UTT) with respect to $\overline{\mathcal{B}}^{t^+}$ by

$$\underline{c}_{(i,s_i),(j,s_j)} = \min_{\tau} \{c_{ij}(\tau) \mid s_i \leq \tau \leq s_i^+\},$$

where (i, s_i^+) is the timed node for i in the later ABSPT. Figure 5(d) shows the UTTs for the example with $t = 2.57$ and $t^+ = 5$ in black along each arc. These are calculated over the time intervals at each node between the times for that node in $\overline{\mathcal{B}}^{2.5}$ and $\overline{\mathcal{B}}^5$. For example, the UTT for timed arc $((1, 0), (2, 1.34))$ is the minimum of $c_{1,2}(\tau)$ over τ in the interval $[0, 2.90]$, which is 0.02, achieved at the right-hand end of the interval. By contrast, the UTT for timed arc $((1, 0), (3, 1.76))$ is the minimum of $c_{1,3}(\tau)$ over τ in the interval $[0, 4.05]$, which is 2.85, achieved at the left-hand end of the interval.

The least UTT path in each ABSPT in the list provides a lower bound on the duration of any feasible timed path arriving in the subinterval between its arrival time at the end node, and that of the next ABSPT in the list. For the example, Figure 5(d) shows the node labels associated with the least UTT path in the ABSPT $\overline{\mathcal{B}}^{2.57}$ in blue against each node. Since the node label on the end node is 1.25, no feasible path arriving at the end node between times 2.57 and 5 can have duration less than 1.25.

The least such lower bound, over all ABSPTs in the list, is a lower bound on the MDP. In the example, since 2.57 is the earliest time that the end node can be reached and 5 is the end of the time horizon, 1.25 is, in fact a lower bound on the duration of *any* feasible timed path.

The lower bound resulting from the list of ABSPTs can be improved by refining the discretization: the insertion of a new ABSPT with end time in the interval following the end time of the ABSPT that gave the lower bound will increase (or at least not decrease) its UTTs.

An upper bound can always be obtained from an ABSPT: if $(1, t_1)$ and (n, t_n) are the two timed nodes in the ABSPT on the start and end node respectively, then $t_n - t_1$ must be an upper bound, by the definition of the BSPT it was generated from. The algorithm maintains the best such upper bound. In the example, the ABSPT $\overline{\mathcal{B}}^{2.57}$ has duration 2.57 and the ABSPT $\overline{\mathcal{B}}^5$ has duration 2.10. Clearly the latter gives the best of these upper bounds. So at this stage of the example, the two ABSPTs give a best lower bound of 1.25 and a best upper bound of 2.10.

The above elements alone – obtaining lower and upper bounds from the list of ABSPTs, and inserting a new ABSPT in the subinterval following the end time of the ABSPT that gave the lower bound – can give an algorithm that converges to the optimal solution, for example, by choosing the new ABSPT to have end time bisecting the subinterval. However, by exploiting the observation of Foschini et al. (2014) – that there is an optimal solution using some arc travel time function breakpoint – the end times for new ABSPTs can be created more carefully, and subintervals eliminated from further consideration, without the need for new ABSPTs. The result is a finitely terminating algorithm. We now describe how we make use of breakpoints in the arc travel time functions to arrive at such an algorithm.

First, we only create ABSPTs that contain at least one node-time pair (i, t) that is a breakpoint. Clearly the initial two ABSPTs satisfy this requirement, as $(1, 0)$ is in the first ABSPT and (n, T) is in the second, and both $(1, 0)$ and (n, T) are deemed to be breakpoints.

Now suppose two consecutive ABSPTs in the list have timed nodes denoted by (i, t_i) for the earlier ABSPT, and (i, t_i^+) for the later, and assume that the earlier ABSPT gave the current lower bound. First consider the case that for some arc $(i, j) \in A$, its arc travel time function has a breakpoint at τ_i with $t_i < \tau_i < t_i^+$. In this case, we say that the breakpoint (i, τ_i) lies *between* the two ABSPTs, and a new ABSPT containing the breakpoint can be constructed by first finding a TDSP to n, starting from i at time τ_i. Say this TDSP arrives at n at time τ_n. Then the BSPT from τ_n, $\mathcal{B}^{\tau_n}$, must include the timed node (i, τ_i). We call its arc completion, $\overline{\mathcal{B}}^{\tau_n}$, the ABSPT *corresponding to* (i, τ_i). By properties of ABSPTs discussed earlier, it must be that $t_n < \tau_n < t_n^+$, and the ABSPT corresponding to (i, τ_i) can be inserted in the list between the two ABSPTs with end times t_n and t_n^+.

Now consider the remaining case, that for every arc $(i, j) \in A$, its arc travel time function has no breakpoint between t_i and t_i^+. In this case, we exploit the fact that the arrival time function at n for departures from node 1 between t_1 and t_1^+ is concave if no minimum duration path that departs between t_1 and t_1^+ contains a breakpoint (Foschini et al. 2014). This situation occurs when there are no more breakpoints left to explore between two ABSPTs, i.e., in this remaining case. When this happens, concavity of the arrival time function at n implies that the minimum duration path departing between t_1 and t_1^+ departs at either t_1 or t_1^+, both of which have already been calculated as the upper bound from the respective ABSPTs. Hence the lower bound for the earlier ABSPT can be updated to one of these upper bounds. In this case, we say the status of the ABSPT is *resolved*.

A high-level overview of our algorithm can be found in Algorithm 1. For a given ABSPT in the ordered list maintained by the algorithm, $\overline{\mathcal{B}}^t$, say, having timed nodes denoted by (j, t_j) for each j, the procedure *computeUB*$(\overline{\mathcal{B}}^t)$ simply returns $t_n - t_1$. The procedure *computeLB*$(\overline{\mathcal{B}}^t)$ constructs the UTTs for each arc in the ABSPT with respect to the subsequent ABSPT in the list, and then finds the least UTT path from $(1, t_1)$ to (n, t_n) in the ABSPT, returning its value. The values computed by these procedures are recorded, and associated with the ABSPT, using the notation UB^t and LB^t, respectively. Note that the UTTs for the last ABSPT in the list, $\overline{\mathcal{B}}^T$, are taken to be the actual travel times if the arc is in the BSPT, and infinity otherwise, since no later departure time at any node can reach the end node within the time horizon. Thus the lower bound from this ABSPT is identical to its upper bound.

Algorithm 1 is actually a family of algorithms, since it does not specify which breakpoint to choose when there are multiple breakpoints between $\overline{\mathcal{B}}^t$ and $\overline{\mathcal{B}}^{t^+}$. We discuss alternative schemes for breakpoint selection, and their effects on computational performance, in Section 5.2.

We illustrate the algorithm on the example from Figure 2 and Table 3: see Figure 6. The algorithm's ABSPT list is initialized with $\overline{\mathcal{B}}^{2.57}$, which corresponds to $(1, 0)$, (the TDSP starting from $(1, 0)$ arrives at node 4 at time $t_0 = 2.57$), and $\overline{\mathcal{B}}^5$, as discussed earlier. Recall that the former gives the best lower bound on the duration, $LB = 1.25$, and the latter gives the best upper bound, $UB = 2.10$. In the first iteration, since arc $(1, 2)$ has a breakpoint at time $\tau = 1 \in [0, 2.90]$, the subinterval arising from timed nodes $(1, 0)$ in $\overline{\mathcal{B}}^{2.57}$ and $(1, 2.90)$ in $\overline{\mathcal{B}}^5$ gives the lower bound, a new breakpoint, $(j, \tau) = (1, 1)$ between the two ABSPTs is found. The TDSP starting from $(1, 1)$ is $((1, 1), (2, 1.66), (4, 3.0826))$, which reaches node 4 at time 3.08 (to two decimal places), and so $\overline{\mathcal{B}}^{3.08}$ is created and added to the list. The list now has three ABSPTs, as shown in Figure 6(b). Computing its upper and lower bounds yields $UB^{3.08} = 2.08$ and $LB^{3.08} = 1.45$

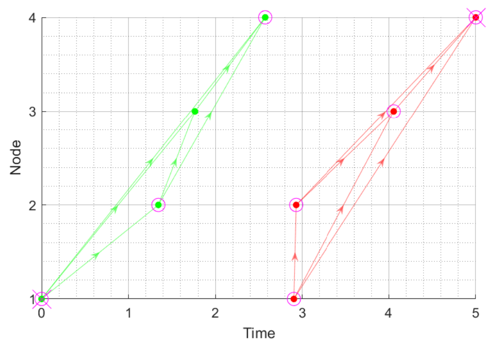
Algorithm 1: Dynamic Discretization Discovery (DDD) Algorithm for the MDP

```
input : digraph $D = (N, A)$, latest time, $T$, arc travel time function $c_{i,j}(t)$ for all $t \in [0, T]$, each
        $(i, j) \in A$
output: minimum duration path from node 1 to $n$ departing and arriving at times in $[0, T]$
Solve the TDSP starting from $(1, 0)$ to determine $t_0$, the ealiest time that $n$ can be reached ;
Initialize ordered list of ABSPTs: set $L \leftarrow (\overline{\mathcal{B}}^{t_0}, \overline{\mathcal{B}}^{T})$ ;
$UB \leftarrow \min\{computeUB(\overline{\mathcal{B}}^{t_0}), computeUB(\overline{\mathcal{B}}^{T})\}$ ;
$LB^{t_0} \leftarrow computeLB(\overline{\mathcal{B}}^{t_0})$ ;
Set $LB \leftarrow LB^{t_0}$, set $t \leftarrow t_0$ and set $t^+ \leftarrow T$ ;
while $(LB < UB)$ do
    if some breakpoint $(j, \tau)$ lies between $\overline{\mathcal{B}}^{t}$ and $\overline{\mathcal{B}}^{t^+}$ then
        Solve the TDSP starting from $(j, \tau)$ to determine, $s$, the earliest arrival at $n$ ;
        Create the new ABSPT $\overline{\mathcal{B}}^{s}$ and set $UB^s \leftarrow computeUB(\overline{\mathcal{B}}^{s})$ ;
        if $UB^s < UB$ then
            $UB \leftarrow UB^s$
        Insert $\overline{\mathcal{B}}^{s}$ in the list $L$ between $\overline{\mathcal{B}}^{t}$ and $\overline{\mathcal{B}}^{t^+}$ ;
        $LB^t \leftarrow computeLB(\overline{\mathcal{B}}^{t})$ ;
        $LB^s \leftarrow computeLB(\overline{\mathcal{B}}^{s})$ ;
    else
        The status of $\overline{\mathcal{B}}^{t}$ is resolved: set $LB^t \leftarrow UB^t$ ;
    Update the lower bound: set $t \leftarrow \arg\min_\tau LB^\tau$ and $LB \leftarrow LB^t$ ;
    Identify the next ABSPT in the list: $t^+ \leftarrow \min\{\tau : \overline{\mathcal{B}}^{\tau}$ is in $L$ and $\tau > t\}$ ;
```

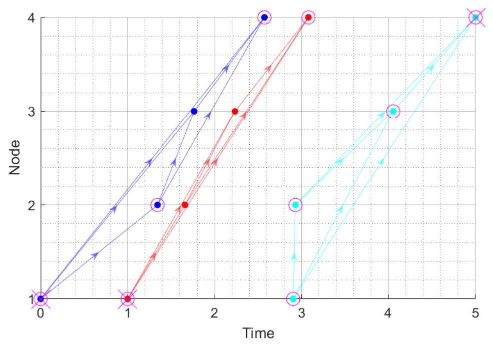
(to two decimal places). After updating the UTTs and recomputing the lower bound for $\overline{\mathcal{B}}^{2.57}$, we update $LB^{2.57} = 1.89$. Now $UB = 2.08$ and $LB = 1.45$. Since $LB < UB$, we continue with the algorithm. We proceed to add the ABSPTs corresponding to $(1, 2)$ and $(2, 2)$ ($\overline{\mathcal{B}}^{3.89}$ and $\overline{\mathcal{B}}^{3.63}$ respectively). The latter is the third ABSPT in the list at Iteration 4. In Iteration 4, $\overline{\mathcal{B}}^{3.08}$ gives the current lower bound, $LB = 1.71$, and is the second ABSPT in the list. There are no breakpoints between the second and third ABSPTs in the list, so we replace the lower bound of $\overline{\mathcal{B}}^{3.08}$ with its upper bound: $LB^{3.08} \leftarrow 2.08$. The new lower bound is given by the ABSPT corresponding to $(2, 2)$, $\overline{\mathcal{B}}^{3.63}$, so at Iteration 5, $LB = 1.89$. We proceed to replace the lower bound of $\overline{\mathcal{B}}^{3.63}$ and $\overline{\mathcal{B}}^{3.89}$ with their upper bound due to the lack of breakpoints between them and the next ABSPT in the list, at which stage $UB = 1.90$, which is equal to $LB = 1.90$ and hence the algorithm terminates. The optimal path is the one that corresponds to the upper bound, which originates from ABSPT $\overline{\mathcal{B}}^{3.89}$ and gives us the minimum duration path $((1, 2.00), (2, 2.14), (4, 3.89))$.

4 A Dynamic Discretization Discovery Algorithm for the MTTP

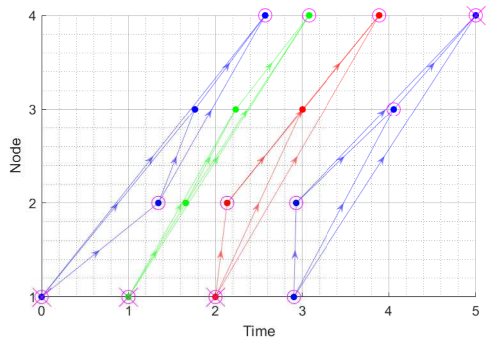
In this section, we extend the ideas of the dynamic discretization discovery algorithm for the MDP to a dynamic discretization discovery algorithm for the MTTP. However, we first observe that the list L of ABSPTs used in Algorithm 1 can be reinterpreted as a single TEN formed by the union of timed nodes and timed arcs in its ABSPTs. The single TEN should also include all timed arcs of the form $((1, t), (1, t^+))$ for *consecutive*, meaning chronologically adjacent, timed nodes $(1, t)$ and $(1, t^+)$ with $t < t^+$. Thus the TEN includes timed arc $((1, t), (1, t^+))$ only if no ABSPT in L includes timed node $(1, t'')$ with $t < t'' < t^+$. Similarly, it should include all timed arcs of the form $((n, t), (n, t^+))$ for consecutive (n, t) and (n, t^+) with $t < t^+$. These arcs model waiting at the source for the right time to start the path and waiting at the sink



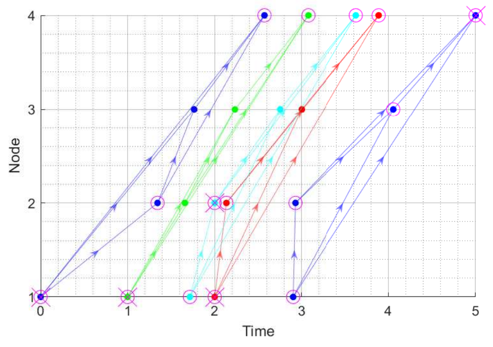
(a) Iteration 1.



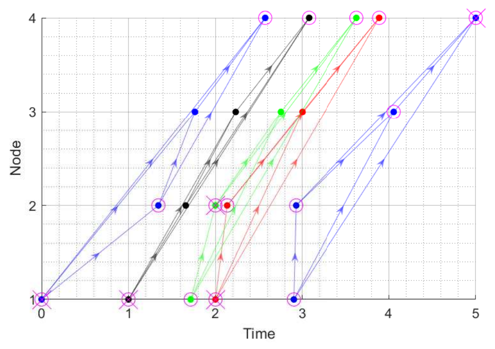
(b) Iteration 2.



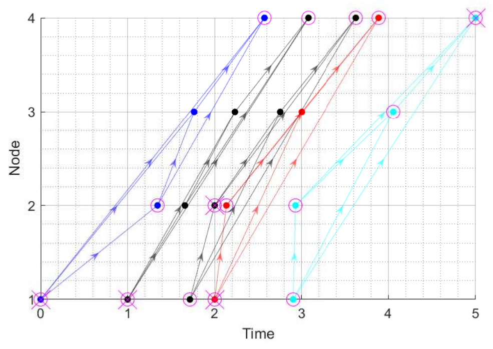
(c) Iteration 3.



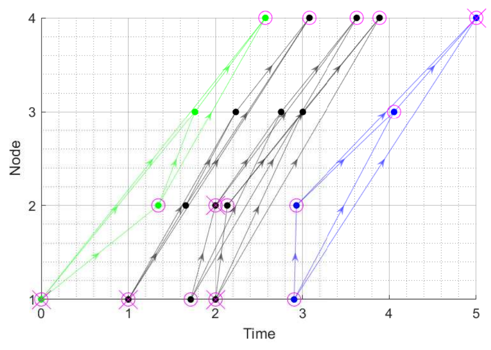
(d) Iteration 4.



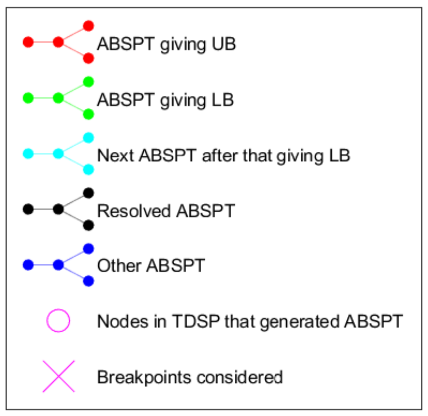
(e) Iteration 5.



(f) Iteration 6.



(g) Iteration 7.



(h) Legend

Figure 6: Time-expanded network (TEN) in each iteration for the example in Figure 2 (note that an ABSPT may satisfy multiple criteria in the legend).

after the end of the path. By assigning all such arcs a UTT of zero, the lower bound calculated at each iteration of the algorithm is precisely the value of the lower bound timed path from $(1, 0)$ to (n, T) in this single TEN. Algorithm 1 exploits the following observations: (i) finding a lower bound timed path in this single TEN can be decomposed into lower bound timed path calculations for each individual ABSPT, and (ii) these calculations will change for only one ABSPT when a new ABSPT is added.

Recall that the key difference between the MDP and the MTTP is that in the latter, it may be that an optimal path must wait at a node other than the source. This prevents decomposition into individual ABSPTs; our lower bound for the MTTP will be calculated using a single TEN that includes *waiting arcs* between timed nodes at the same node in N, which accommodate the option to wait at that node. In addition to loss of the decomposition property when solving the MTTP, the ability to use ABSPTs as the basis for the TEN is also lost: ABSPTs, alone, are not sufficient to characterize solutions to the MTTP. Instead, the TEN we construct is based on unions of *forwards* and backwards trees from a node *other than* the source or sink, starting and ending, respectively, at a given time. In what follows, we give the motivation and details for this construction.

In this section, we extend the ideas of the dynamic discretization discovery algorithm for the MDP to a dynamic discretization discovery algorithm for the MTTP. However, we first observe that the list L of ABSPTs used in Algorithm 1 can be reinterpreted as a single TEN formed by the union of timed nodes and timed arcs in its ABSPTs. The single TEN should also include all timed arcs of the form $((1, t), (1, t^+))$ for *consecutive*, meaning chronologically adjacent, timed nodes $(1, t)$ and $(1, t^+)$ with $t < t^+$. Thus the TEN includes timed arc $((1, t), (1, t^+))$ only if no ABSPT in L includes timed node $(1, t'')$ with $t < t'' < t^+$. Similarly, it should include all timed arcs of the form $((n, t), (n, t^+))$ for consecutive (n, t) and (n, t^+) with $t < t^+$. These arcs model waiting at the source for the right time to start the path and waiting at the sink

after the end of the path. By assigning all such arcs a UTT of zero, the lower bound calculated at each iteration of the algorithm is precisely the value of the lower bound timed path from $(1, 0)$ to (n, T) in this single TEN. Algorithm 1 exploits the following observations: (i) finding a lower bound timed path in this single TEN can be decomposed into lower bound timed path calculations for each individual ABSPT, and (ii) these calculations will change for only one ABSPT when a new ABSPT is added.

Recall that the key difference between the MDP and the MTTP is that in the latter, it may be that an optimal path must wait at a node other than the source. This prevents decomposition into individual ABSPTs; our lower bound for the MTTP will be calculated using a single TEN that includes *waiting arcs* between timed nodes at the same node in N, which accommodate the option to wait at that node. In addition to loss of the decomposition property when solving the MTTP, the ability to use ABSPTs as the basis for the TEN is also lost: ABSPTs, alone, are not sufficient to characterize solutions to the MTTP. Instead, the TEN we construct is based on unions of *forwards* and backwards trees from a node *other than* the source or sink, starting and ending, respectively, at a given time. In what follows, we give the motivation and details for this construction.

4.1 Foundational Theory for the MTTP

The foundations for our algorithm for the MTTP start with the simple observation that any solution to a MTTP instance will contain periods of travel and periods of waiting, as shown, for example, in Figure 7. Indeed, any timed path corresponds to a sequence of waiting-free timed paths.

Definition 1 *A* travel subpath *of a given timed path is created from a maximal sequence of consecutive timed nodes in the timed path, say $(i_1, t_1), \ldots, (i_k, t_k)$ is such a sequence, satisfying (i) $t_{\ell+1} = t_\ell + c_{i_\ell, i_{\ell+1}}(t_\ell)$ for all $\ell = 1, \ldots, k-2$ and (ii) either $t_k > t_{k-1} + c_{i_{k-1}, i_k}(t_{k-1})$, which indicates that the path waits at i_k, or $i_k = n$ and $t_k = t_{k-1} + c_{i_{k-1}, i_k}(t_{k-1})$. The travel subpath is then defined as the timed node sequence $((i_1, t_1), \ldots, (i_{k-1}, t_{k-1}), (i_k, \tau))$ where $\tau = t_{k-1} + c_{i_{k-1}, i_k}(t_{k-1})$.*

For example, if $((1, 0), (2, 6.9), (3, 7.9), (4, 8.1), (5, 8.5))$ is the sequence of nodes in a timed path, with $\tau_{12} = 1$, $\tau_{23} = 0.2$, $\tau_{34} = 0.2$ and $\tau_{34} = 0.4$, then the timed path has three travel subpaths, given by the timed node sequences $((1, 0), (2, 1))$, $((2, 6.9), (3, 7.1))$ and $((3, 7.9), (4, 8.1), (5, 8.5))$. A timed path like this is illustrated in Figure 7, which shows a TEN for the network $D = (\{1,2,3,4,5\}, \{(1,2),(2,3),(3,4),(4,5)\})$, with one timed path shown in red and green. Its three travel subpaths are shown in red.

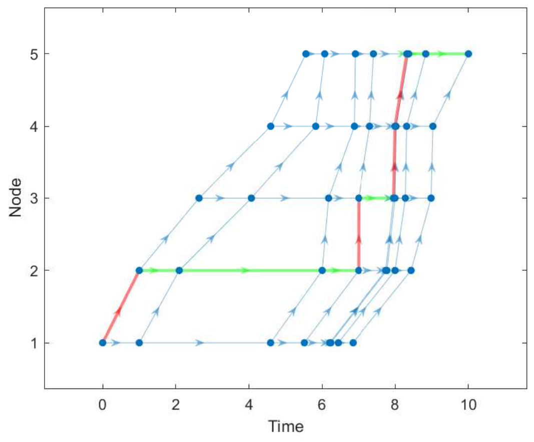


Figure 7: Illustration of a timed path with 3 periods of waiting in green and 3 travel subpaths in red, in the TEN for a network consisting only of the arcs $(1,2),(2,3),(3,4)$ and $(4,5)$.

A travel subpath of a timed path is maximally waiting-free: it cannot be extended at either end while remaining waiting-free. A timed path can be viewed as a sequence of travel subpaths, with waiting between them: whenever timed nodes (i, t_i) and (j, t_j) appear consecutively in a timed path and have $t_j > t_i + c_{i,j}(t_i)$, then the timed node $(j, t_i + c_{i,j}(t_i))$ is the end of one of its travel subpaths and the timed node (j, t_j) is the start of the next. Note that a travel subpath is a timed path in its own right, and is, by construction, waiting-free.

Obviously any sequence of timed paths with the property that $i = j$ and $t_j \geq t_i$ whenever (i, t_i) is the last timed node in one of them and (j, t_j) is the first timed node of the next can be converted to a single timed path simply by omitting the last timed node of every timed path in the sequence, except the last, and then concatenating them. Clearly, for any timed path P there is a unique sequence of waiting-free timed paths that can be used to generate P by this process, the elements of which are precisely its travel subpaths. We say that this sequence *corresponds to* P and *vice versa*.

We now present the key lemma needed to extend the observation of Foschini et al. (2014) to the minimum travel time case.

Lemma 1 *Given an optimal solution, P, to an MTTP instance on network D with arc travel time functions c and time horizon $[0, T]$, consider any travel subpath of it, S say. Say S starts at timed node (i, t_i) and ends at timed node (j, t_j). Then S is an optimal solution to the MDP instance on the same network, D, with the same arc travel time functions, c, but with source i, sink j and time horizon $[\tau^-, \tau^+]$, where $\tau^- \leq t_i$ is the end time of the travel subpath immediately preceding S in P, if any, and $\tau^+ \geq t_j$ is the start time of the travel subpath immediately following S in P, if any. Here $\tau^- = 0$ if $i = 1$ (there is no travel subpath preceding S) and $\tau^+ = T$ if $j = n$ (there is no travel subpath after S).*

Proof. Given P, S, i, j, t_i, t_j, τ^- and τ^+ as defined above for the MTTP instance, we will refer to the MDP instance with source i, sink j and time horizon $[\tau^-, \tau^+]$ as the $i \to j$ *MD instance*. Observe that S is a feasible solution to the $i \to j$ MD instance and its total travel time is given by $t_j - t_i$, which is precisely its duration, since S is a travel subpath and hence is waiting-free. Thus S has total travel time *at least* the optimal value of the $i \to j$ MD instance. Also observe that the duration of any timed path is at least its total travel time. Now suppose that S is *not* optimal for the $i \to j$ MD instance. Then it has strictly greater total travel time than the optimal value of the $i \to j$ MD instance, which is at least the total travel time of any of its optimal solutions. Hence, S can be replaced in the sequence of waiting-free timed paths that corresponds to P by any solution to the $i \to j$ MD instance. The resulting, new, sequence of waiting-free timed paths must yield a corresponding timed path that is feasible for the MTTP and has strictly lower total travel time than that of P, contradicting the optimality of P. □

The following is the extension of the observation of Foschini et al. (2014) to the MTTP.

Proposition 1 *For any MTTP instance, there exists an optimal timed path such that each of its travel subpaths includes at least one timed node at an arc travel time function breakpoint.*

Proof. Suppose that P is an optimal timed path for a MTTP instance having a travel subpath, S, that does *not* include a timed node at an arc travel time function breakpoint. Then by Lemma 1, S is an optimal solution to the MDP instance on the same network with the same arc travel time functions, but with source i, sink j and time horizon $[\tau^-, \tau^+]$, where i, j, τ^- and τ^+ are defined as in the statement of the lemma. By the results of Foschini et al. (2014), there must be another optimal solution to this MDP instance, S' say, which includes at least one timed node at a breakpoint, or which starts at i at time τ^- or ends at j at time τ^+. Since S and S' are both waiting-free, their total travel time is precisely their durations, which are identical, as both are optimal for the same MDP instance. By replacing S with S' in the sequence of waiting-free timed paths corresponding to P, a corresponding new timed path, P', is obtained. Clearly P' is feasible for the MTTP, has the same total travel time as that of P, and has one fewer travel subpaths that does not include a timed node at a breakpoint. (In the case that S' starts at time τ^-, and (i, τ^-) is not at an arc travel time function breakpoint, the travel subpath preceding S is no longer maximal: it now extends to S', resulting in one fewer travel subpaths in P' than in P. The case that S' ends at time τ^+, and (j, τ^+) is not at a breakpoint is similar: S' is not a travel subpath of P' but will extend to the travel subpath that followed S.)□

The above proposition shows that, given an MTTP instance, there must exist a (finite) TEN and an arc length for each of its arcs so that any shortest path in the TEN corresponds to an optimal solution of the MTTP instance. We define this TEN and its arc lengths below. But first, we extend the definition of a backwards shortest path tree to be one rooted at a timed node (k, t) for *any* node $k \in N$. Such a BSPT is formed by solving a TDSPP to find, for each node, the latest departure time at that node so that a path from the node to k arrives at time t. We denote this BSPT by $\mathcal{B}^{k,t}$. We refer to any such BSPT a *k-BSPT*. Similarly, a *forwards shortest path tree* (FSPT) is formed by solving a TDSPP to find, for each node, the earliest arrival time at that node on a path from the node to k departing at time t, for any node k. We denote this FSPT by $\mathcal{F}^{k,t}$ and refer to any such FSPT a *k-FSPT*.

Definition 2 *Given a MTTP instance with arc travel time functions c, construct* its time-expanded network with associated arc lengths*, abbreviated as* its TENL*, as follows.*

1. *For each breakpoint (i, t) in the given instance, solve two TDSPP problems, one to calculate the i-FSPT $\mathcal{F}^{i,t}$ and the other to calculate the i-BSPT $\mathcal{B}^{i,t}$. Form a single TEN from the union of all the timed nodes and timed arcs in $\mathcal{F}^{i,t}$ and $\mathcal{B}^{i,t}$ over all breakpoints, (i, t), in the instance.*

2. *For each node in the given instance, add waiting arcs between chronologically adjacent timed copies of it in the TEN.*

3. *Assign any arc in the resulting TEN, $((i, t), (j, t'))$ say, a* length *of zero if $j = i$ and a length of $c_{i,j}(t)$ (which must be equal to $t' - t$, by construction), otherwise.*

Recall that $(1, 0)$ and (n, T) are deemed to be breakpoints in a MTTP instance with node set $\{1, \ldots, n\}$ and time horizon $[0, T]$, so both are nodes in its TENL.

Corollary 1 *Given a MTTP instance with node set $\{1, \ldots, n\}$ and time horizon $[0, T]$, the sequence of timed nodes in any shortest path from $(1, 0)$ to (n, T) in its TENL corresponds to a timed path that solves the MTTP instance.*

Proof. Let P be an optimal solution to the given MTTP instance with the property that each of its travel subpaths includes a breakpoint in the instance. Such a path must exist, by Proposition 1. Suppose P starts with $(1, t_1)$ and ends with (n, t_n). It is not difficult to see that the sequence of travel subpaths corresponding to P induces a path in the TEN from $(1, t_1)$ to (n, t_n) of length equal to the total travel time of P: the latter part of each travel subpath, from the breakpoint it contains to its end, is part of the FSPT for that breakpoint, and the earlier part of each travel subpath, to the breakpoint it contains from its start, is part of the BSPT for that breakpoint. Thus each travel subpath is a path in the TEN, and the sum of the TEN arc lengths over all of the travel subpaths of P is precisely its total travel time. Furthermore, the travel subpaths can obviously be connected via waiting arcs in the TEN, and connected to (n, T) from the end of the last travel subpath and from $(1, 0)$ to the start of the first travel subpath, to create a single path in the TEN from $(1, 0)$ to (n, T), while adding nothing to the length. We refer to this path as P^{TEN}. Also obvious is that any timed path in the TEN from $(1, 0)$ to (n, T) has length identical to the sum of lengths of the arcs in same path with all waiting arcs removed, and that the connected components of these form a sequence of waiting-free timed paths whose corresponding timed path departs node 1 no earlier than 0, arrives at n no later than T, and has total travel time equal to the sum of their lengths. Thus every path in the TEN from $(1, 0)$ to (n, T) has length equal to the total travel time of some feasible timed path for the MTTP instance. The result follows. $\square$

Corollary 1 provides a method for solving the MTTP that is analogous to the method of Foschini et al. (2014) for solving the MDP. The difference is that now a shortest path problem over the TEN, in its entirety, must be solved; it cannot be decomposed into distinct shortest path problems for each breakpoint. Since its TEN can be expected to have $\mathcal{O}(K^{\text{nodes}}n)$ nodes and $\mathcal{O}(K^{\text{nodes}}n)$ arcs for an MTTP instance with n nodes and K^{nodes} breakpoints, its size is clearly sensitive to the number of breakpoints in the instance. Specifically, we calculate the following complexity.

Proposition 2 *The complexity of the MTTP on digraph (N, A) having $n = |N|$ nodes and K^{nodes} travel time function breakpoints at nodes is*

$$\mathcal{O}(K^{nodes} \times SSP(n, |A|) + ASPP(K^{nodes}n, K^{nodes}n)),$$

where $SPP(\alpha, \beta)$ denotes the complexity of solving a static shortest path problem on a digraph with α nodes and β arcs, and $ASPP(\alpha, \beta)$ denotes the same thing but for an acyclic digraph.

Proof. For each node, there can be up to two timed copies created by the FSPT and BSPT for each breakpoint, giving up to $2K^{\text{nodes}}$ timed copies of each node and hence at most $2K^{\text{nodes}}$ waiting arcs at each node. Thus there would be a total of at most $2K^{\text{nodes}}n$ nodes in the TEN and at most $2K^{\text{nodes}}n$ waiting arcs. Each breakpoint creates a FSPT, which can have at most n arcs in it, and a BSPT, which can similarly have at most n arcs. So the FSPTs and BSPTs contribute in total no more than $2n$ arcs per breakpoint, giving a total of $2nK^{\text{nodes}}$ arcs. Adding to the waiting arcs, this gives a total of no more than $4nK^{\text{nodes}}$ arcs. Calculating the FSPT and BSPT for each breakpoint requires $\mathcal{O}(2 \times K^{\text{nodes}} \times SSP(n, |A|))$ operations, and, since the TENL is acyclic, (as all arcs point forwards in time), finding the shortest path in the TENL requires $\mathcal{O}(ASPP(2K^{\text{nodes}}n, 4K^{\text{nodes}}n))$ operations. The result follows. $\square$

4.2 A DDD Algorithm for the MTTP

As for the MDP, we develop a dynamic discretization discovery algorithm to reduce the number of breakpoints explored whilst still being able to certify optimality.

Based on Proposition 1, ABSPTs are not the natural structures to use in an algorithm for the MTTP. An optimal solution consists of a sequence of travel subpaths connected by waiting, where each travel subpath can safely (by Proposition 1) be assumed to include a breakpoint and the waiting time between any two consecutive travel subpaths can safely be assumed positive (not zero). (By a “safe” assumption, we mean one that is satisfied by at least one optimal solution.) Under the former assumption, it must also be the case that each travel subpath consists of a TDSP to/from the breakpoint it includes. In the remainder of this paper, we make this assumption, using the following definition.

Definition 3 *A* travel subpath for breakpoint (i,t)*, referred to as an* (i,t)*-travel subpath, is a waiting-free timed path concatenating a TDSP ending at* (i,t) *with a TDSP starting from* (i,t)*. The term* travel subpath *is used to refer to such a path for some unspecified breakpoint, while* i-travel subpath *is used for a travel subpath for a breakpoint at node* i *at an unspecified time.*

Thus there exists some optimal solution to an MTTP that has each of its travel subpaths lying in a TEN formed by the union of a forward and backward shortest path tree for a specific breakpoint. Hence, instead of an ABSPT, the more natural structure to use for solving the MTTP is the union of an i-FSPT and an i-BSPT for the same breakpoint at node i. We call such a union a *mangrove*, and define it formally as follows. Note that we will use the union operator, $\cup$, to form a TEN as a union of two TENs, in the obvious way.

Definition 4 *A* mangrove *for breakpoint* (i,t)*, denoted by* $\mathcal{M}^{i,t}$*, is the TEN created by taking the union of the i-FSPT* $\mathcal{F}^{i,t}$ *and the i-BSPT* $\mathcal{B}^{i,t}$*, written as* $\mathcal{M}^{i,t} = \mathcal{F}^{i,t} \cup \mathcal{B}^{i,t}$*. We refer to any such mangrove as an* i*-mangrove and use the prefix* (i,t)*- if we wish to specify the mangrove for breakpoint* (i,t)*.*

Similarly to the BSPTs used in our MDP algorithm, the FIFO property ensures that for any node i, the set of all i-mangroves is totally ordered with respect to time for timed nodes in their i-FSPTs and for timed nodes in their i-BSPTs. Specifically, if $\mathcal{M}^{i,t} = \mathcal{F}^{i,t} \cup \mathcal{B}^{i,t}$ and $\mathcal{M}^{i,t'} = \mathcal{F}^{i,t'} \cup \mathcal{B}^{i,t'}$ for $t' > t$, then for any node j with (j,s) in $\mathcal{F}^{i,t}$ and (j,s') in $\mathcal{F}^{i,t'}$ it must be that $s' > s$, and, similarly, for any node j with (j,s) in $\mathcal{B}^{i,t}$ and (j,s') in $\mathcal{B}^{i,t'}$ it must be that $s' > s$.

As a consequence, any i-travel subpath that arrives at node i at time t or later has a representative timed node sequence in $\mathcal{M}^{i,t}$. However, to ensure its arcs are also represented, arc-completion is, again, needed. For given breakpoint (i,t), the *Arc-completed Forwards Shortest Path Tree* (AFSPT), denoted by $\overline{\mathcal{F}}^{i,t}$, is given by $\mathcal{F}^{i,t}$ together with the addition of arcs $((j,t'),(k,t''))$ that are not already in $\mathcal{F}^{i,t}$, for every $(j,k) \in A$ having both (j,t') and (k,t'') in the FSPT. Similarly, we extend the ABSPT definition: the ABSPT for breakpoint (i,t), denoted by $\overline{\mathcal{B}}^{i,t}$, is given by $\mathcal{B}^{i,t}$ together with the addition of arcs $((j,t'),(k,t''))$ that are not already in $\mathcal{B}^{i,t}$, for every $(j,k) \in A$ having both (j,t') and (k,t'') in the BSPT. The *Arc-completed Mangrove* for breakpoint (i,t) is denoted by $\overline{\mathcal{M}}^{i,t}$ and is the TEN formed by taking the union of $\overline{\mathcal{F}}^{i,t}$ and $\overline{\mathcal{B}}^{i,t}$. Note that to form an arc-completed mangrove, $\overline{\mathcal{M}}^{i,t}$, its FSPT and its BSPT are arc-completed separately, since we need only that the arc-completed mangrove represent travel subpaths containing node i at time t or later.

Our algorithm maintains a set of TENs, each either the mangrove or the arc-completed mangrove for some breakpoint. Each subset of these TENs that is induced by breakpoints at the same node are ordered by their breakpoint time. The algorithm thus maintains, for each node i, an ordered list, denoted by L^i, of breakpoints, (i,t). Whenever (i,t) is added to L^i, the (i,t)-mangrove, $\mathcal{M}^{i,t}$, and its arc-completion, $\overline{\mathcal{M}}^{i,t}$, are computed. For each consecutive pair of breakpoints in L^i, say (i,t) and (i,t^+) are such a pair, with $t^+ > t$, we proceed as follows. In the case that there is no breakpoint at i between t and t^+, we say that the mangrove for (i,t) is *resolved*, or simply that (i,t) is resolved. In this case, by the same properties used for the MDP, it must be that if an i-travel subpath using a breakpoint in $[t,t^+)$ appears in the optimal solution, then it must lie in the (i,t)-mangrove. Thus the algorithm keeps only the (i,t)-mangrove, discarding any arcs in $\overline{\mathcal{M}}^{i,t} \setminus \mathcal{M}^{i,t}$, and sets the UTT to the true travel time, so

$$\underline{c}_{(j,s),(k,s')} := c_{j,k}(s)$$

for each timed arc $((j,s),(k,s'))$ in $\mathcal{M}^{i,t}$. Otherwise, in the case that (i,t) is *not* resolved, the algorithm sets the UTT on each arc $((j,s),(k,s'))$ in $\overline{\mathcal{M}}^{i,t}$ as follows: the UTT on each arc $((j,s),(k,s'))$ in $\overline{\mathcal{F}}^{i,t}$ (respectively $\overline{\mathcal{B}}^{i,t}$) is set to be

$$\underline{c}_{(j,s),(k,s')} := \min_{\tau}\{c_{j,k}(\tau) : s \le \tau \le s^+\}$$

where s^+ is the unique time such that (j,s^+) is a node in $\overline{\mathcal{F}}^{i,t^+}$ (respectively, $\overline{\mathcal{B}}^{i,t^+}$). Now any travel subpath for a breakpoint at i at a time in the interval $[t,t^+]$ is represented by a path in $\overline{\mathcal{M}}^{i,t}$ having total UTT no greater than the travel time of the travel subpath.

By ensuring that, for all i, L^i includes a sufficiently early breakpoint, any travel subpath is thus represented by a path in a mangrove or an arc-completed mangrove whose UTT is a lower bound on the travel subpath's travel time. For simplicity of exposition, our algorithm initializes L^i with $(i, 0)$ for each i, but, in practice, it would be more efficient to use (i, t) for t the time of the first breakpoint at node i after s, where (i, s) is in $\mathcal{F}^{1,0}$. We similarly include (i, T) in the initial list, L^i, for each i, and deem the mangrove for (i, T) to be resolved, so the UTT for its arcs are simply their true travel time.

To obtain a lower bound on the travel time of the MTTP solution, which (without loss of generality) corresponds to a *sequence* of travel subpaths, the i-mangroves for different i must be connected, by arcs that represent waiting at a node between travel subpaths.

Our algorithm thus maintains a TEN, denoted by $\mathcal{D}_{LB}$, with associated UTTs on each timed arc, defined as a function of the mangroves and arc-completed mangroves whose breakpoints are in the set $\bigcup_{i\in N} L^i$, as follows.

Timed Nodes and Travel Arcs. All timed nodes and all arcs in resolved mangroves and arc-completed mangroves are included in $\mathcal{D}_{LB}$, together with their UTTs, as defined above.

Waiting Arcs. In addition, waiting arcs are included to connect the (arc-completed) mangrove[1] representing one travel subpath to another (arc-completed) mangrove that may represent the next. We consider how to represent any possible sequence consisting of an i-travel subpath ending at node j, then waiting at node j, followed by a k-travel subpath starting at node j, for $k \neq i$, while ensuring that the UTT of its representation is no greater than its true travel time. Recall that we need only consider the case of positive (nonzero) waiting time at j.

Specifically, for each timed node (j, s), with $j \neq n$, in the FSPT for some mangrove, say it is in $\mathcal{M}^{i,t}$, we consider representing a sequence starting with an i-travel subpath ending at j at time s or later, and hence visiting node i at time t or later. If $t = T$, then no (i, t)-travel subpath can appear in a feasible solution. So we assume $t < T$, and let s^+ be such that (j, s^+) is in the FSPT for the i-mangrove with the next breakpoint in L^i after (i, t). This must exist, since L^i is initialized to include (i, T). We consider two cases, where the second case has two subcases. These are illustrated in Figure 8 and Figure 9. In these figures, a square indicate mangrove "root", a circle indicates a regular node, and travel subpaths in the network are in blue.

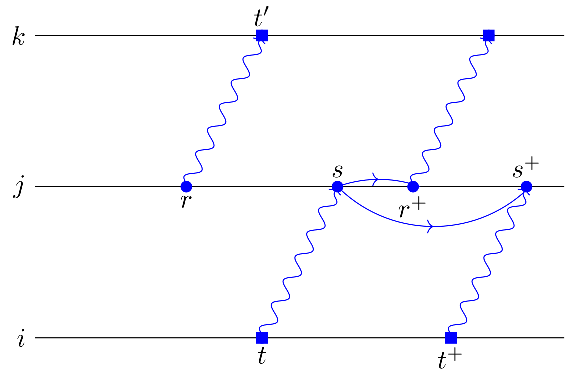


Figure 8: Forwards waiting arcs: Cases 1 and 2(a).

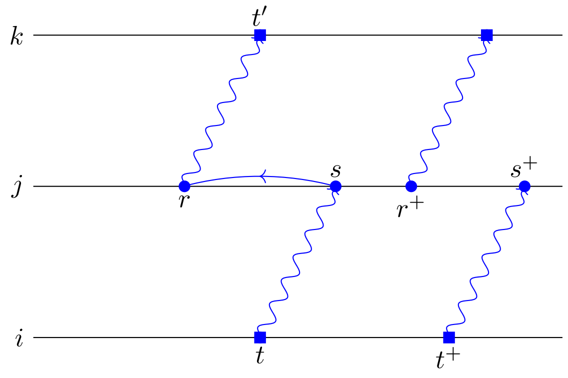


Figure 9: Backwards waiting arc: Case 2(b).

Case 1. To represent waiting at j until time s^+ or later, for each node i, the timed arcs $((j, s), (j, s^+))$ are included in $\mathcal{D}_{LB}$, and assigned a UTT of zero. Waiting arcs joining timed nodes in the BSPTs for consecutive i-mangroves in L^i are not needed, since we may, without loss of generality, assume that waiting occurs at the end of a travel subpath rather than at the start.

[1]Meaning a resolved mangrove or an arc-completed mangrove. Recall that mangroves and their arc-completions have the same timed node set.

Case 2. For each node $k \neq i$, let r be the latest time that is earlier than s at which a timed node (j, r) appears in the BSPT for a k-mangrove, $\mathcal{M}^{k,t'}$, say. (So there is no $t'' > t'$ with $\mathcal{M}^{k,t''}$ in L^k, having (j, r') in $\mathcal{B}^{k,t''}$ with $r < r' < s$.) Such a t' and r must exist, since (j, s) in FSPT $\mathcal{F}^{i,t}$ implies that either $s > 0$ or $i = j$ and $s = t = 0$, while any timed node in BSPT $\mathcal{B}^{k,0}$ has negative time at any node other than k, and we initialize L^k to include $(k, 0)$. In addition, if $t' < T$, take r^+ so that (j, r^+) is in the BSPT for the mangrove with next breakpoint in L^k after (k, t'). Note that such $r^+ \geq s$. We consider two subcases.

(a) The mangrove for (k, t') is resolved. In this case, no waiting arc joining (j, s) to the mangrove for (k, t') is needed, since this mangrove represents only (k, t')-travel subpaths and the timed node (k, t') cannot be reached after positive waiting time at (j, s). If $t' = T$ then there is no next k-mangrove after that for (k, t'). Otherwise, (j, s) is connected to the next k-mangrove: the arc $((j, s), (j, r^+))$ is included in $\mathcal{D}_{LB}$, with a UTT of zero. This allows $\mathcal{D}_{LB}$ to represent the sequence of an i-travel subpath ending at j at time $\tau \in [s, s^+)$ followed by a k-travel subpath starting at j at some time $\tau' > \tau$, using the next breakpoint at k after t'. Since $r^+ \geq s$ we call $((j, s), (j, r^+))$ a *forwards waiting arc.*

(b) The mangrove for (k, t') is not resolved. Then the arc $((j, s), (j, r))$ is included in $\mathcal{D}_{LB}$, with a UTT of zero. Then including the arc $((j, s), (j, r))$ allows $\mathcal{D}_{LB}$ to represent the sequence of an i-travel subpath ending at j at time $\tau \in [s, s^+)$ followed by a k-travel subpath starting at j at some time $\tau' \in [\tau, r^+)$. Since $r \leq s$ we call $((j, s), (j, r))$ a *backwards waiting arc.*

The way in which these waiting arcs allow the resulting TEN to represent possible optimal paths is illustrated in Figure 10, where possible optimal paths are drawn in green.

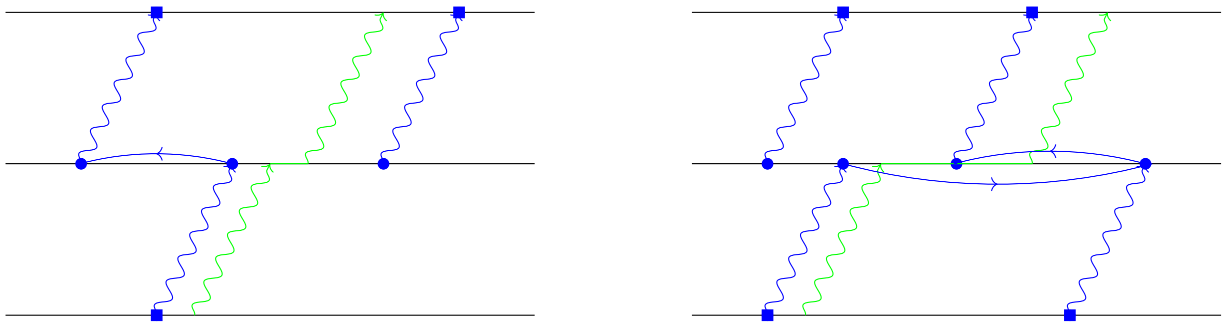


Figure 10: Lower bound paths using waiting arcs allow possible optimal paths to be represented.

Dummy Nodes and Arcs. Add a dummy start node and a dummy end node to $\mathcal{D}_{LB}$, and include in it arcs to connect the dummy start node to every timed node of the form $(1, t)$ with $t \geq 0$ and to connect every timed node of the form (n, t) with $t \leq T$ to the dummy end node, all arcs having zero UTT.

We claim that the least UTT path in $\mathcal{D}_{LB}$, constructed as above, from its dummy start to its dummy end node, has UTT a lower bound on the value of the MTTP. To see this, consider an optimal solution containing the sequence of an (i, α)-travel subpath, P_1^*, ending at (j, α') followed by a (k, β')-travel subpath, P_2^*, starting at (j, β), where $\beta - \alpha' > 0$ is the waiting time between the travel subpaths at node j. Let (i, s) be the latest breakpoint in L^i with $s \leq \alpha$ and let (k, t') be the latest breakpoint in L^k with $t' \leq \beta'$. Then P_1^* is represented by a path in $\overline{\mathcal{M}}^{i,s}$ with UTT at most the travel time of P_1^* and P_2^* is represented by a path in $\overline{\mathcal{M}}^{k,t'}$ with UTT at most the travel time of P_2^*. Let (j, s') be the unique timed node in $\mathcal{F}^{i,s}$ for node j and let (j, t) be the unique timed node in $\mathcal{B}^{k,t'}$ for node j. We claim that there is a path in $\mathcal{D}_{LB}$ from (j, s') to (j, t) with UTT of zero, by the construction above. Observe that $s' \leq \alpha'$ and $t \leq \beta$ by the ordering of mangroves for the same node. However $t - s'$ may have any sign. If $t < s'$ then the (k, t')-mangrove cannot be resolved, (since otherwise it must be that $t' = \beta'$ and so $t = \beta > \alpha' \geq s'$), so the backward arc $((j, t), (j, s'))$ is included in $\mathcal{D}_{LB}$, as per Case 2(b). Otherwise, if $t \geq s'$, there are two subcases. In the case that the (k, t')-mangrove is resolved, the forward arc $((j, t), (j, s'))$ is included in $\mathcal{D}_{LB}$, as per Case 2(a). Otherwise, let (i, r) be the first breakpoint in L^i after (i, s) so that the unique timed node (j, r') in $\mathcal{F}^{i,r}$ has $r' \geq t$. (Such an r must exist since $j \neq n$ and (i, T) is in L^i.) Then by Case 1, there is a sequence of zero-UTT arcs connecting (j, s') to (j, r') and by Case 2(b) there is backwards arc $((j, r'), (j, s'))$ having zero UTT in $\mathcal{D}_{LB}$. Thus in all cases there is a zero-UTT path in $\mathcal{D}_{LB}$ from (j, s') to (j, t), and so the sequence P_1^* followed by P_2^* is represented in $\mathcal{D}_{LB}$ by a path having UTT no more than the combined travel time of P_1^* and P_2^*. We have thus proved the following result.

Proposition 3 *Using the procedure described above, let $\mathcal{D}_{\mathcal{LB}}$ and a UTT for each arc in $\mathcal{D}_{\mathcal{LB}}$ be constructed from lists L^i having the property that $(i, 0), (i, T) \in L^i$ for each $i \in N$. Then the least UTT path in $\mathcal{D}_{\mathcal{LB}}$ from its dummy start node to its dummy end node has UTT a lower bound on the value of the MTTP.*

In the statement of the algorithm, procedure $createLBTEN(\{L^i\}_{i \in N}, Resolved)$ constructs $\mathcal{D}_{LB}$ with its associated UTTs, given in the vector $\underline{c}$, as described above, where $Resolved$ indicates the set of resolved breakpoints, and the procedure $computeSP(\mathcal{D}_{LB}, \underline{c})$ calculates the least UTT path, returning the path and its total UTT.

The lists $\{L^i\}_{i \in N}$ are also used to furnish an upper bound, by the construction of a TEN, denoted by $\mathcal{D}_{UB}$, as follows.

1. All arcs in (i, t)-mangroves for all (i, t) in L^i and all $i \in N$ are included in $\mathcal{D}_{UB}$, with their original travel times.

2. For each (j, s) in the FPST for some i-mangrove, and for each $k \neq i$, let r be the earliest time later than s at which node (j, r) appears in the BSPT for a k-mangrove, if any. If r exists, add the waiting arc $((j, s), (j, r))$ to $\mathcal{D}_{UB}$, with travel time set to zero.

3. Add a dummy start node and a dummy end node to $\mathcal{D}_{UB}$, and include in it arcs to connect the dummy start node to every timed node of the form $(1, t)$ with $t \geq 0$ and to connect every timed node of the form (n, t) with $t \leq T$ to the dummy end node, all arcs having travel time set to zero.

Clearly any path in $\mathcal{D}_{UB}$ from its dummy start to its dummy end node corresponds to a feasible solution to the MTTP, and its total travel time gives an upper bound on the value of the MTTP. Minimizing the total travel time of such a path gives the best upper bound available from $\mathcal{D}_{UB}$. In the statement of the algorithm, procedure $createUBTEN(\{L^i\}_{i \in N})$ constructs $\mathcal{D}_{UB}$, and its associated travel times, stored in the vector $\overline{c}$, as described above, and the procedure $computeSP(\mathcal{D}_{UB}, \overline{c})$ calculates the current best feasible path for the MTTP, returning both the path and its total travel time.

The algorithm, stated formally as Algorithm 2, proceeds by solving a shortest path problem on both the upper bound and lower bound TEN to obtain upper and lower bounds on the value of the MTTP, respectively. If the two bounds are not equal, we consider the least UTT path in the lower bound TEN, and the arc-completed mangroves it uses. It must be that at least one of the arc-completed (i, t)-mangroves it uses has its breakpoint (i, t) not resolved, since otherwise the lower and upper bounds would be the same. The algorithm then chooses any set of one or more such breakpoints. For each breakpoint, (i, t) say, in the chosen set, some (one or more) breakpoints between (i, t) and the next breakpoint in L^i are selected, and added to L^i. If, now, any breakpoint in L^i is the first breakpoint at node i after some breakpoint (i, s) in L^i, then (i, s) is marked as resolved. The lists $\{L_i\}_{i \in N}$ are again used to construct upper and lower bound TENs, and the bounds computed. This continues until the upper and lower bounds are equal.

Clearly this algorithm is really a class of algorithms, and its performance in practice will depend on which set of unresolved breakpoints is chosen at each iteration, and, for each, which set of breakpoints at the same node are added to the list for that node. In Section 5.2 we describe and compare specific choices for these elements of the algorithm. Here, we simply name the function that decides a new set of breakpoints to add to the lists $FindBP(P, \{L^i\}_{i \in N})$, where P is the least UTT path found in $\mathcal{D}_{LB}$.

Algorithm 2: Dynamic Discretization Discovery (DDD) Algorithm for the MTTP

```
input  : digraph $D = (N, A)$, latest time at end node $T$, arc travel time function $c_{i,j}(t)$ for all
         $t \in [0, T]$, for each $(i, j) \in A$
output: minimum travel time path from node 1 to $n$ departing from 1 and arriving at $n$ at times in
         $[0, T]$
$Resolved := \emptyset$;
for $i = 1$ to $n$ do
    Compute the $(i, 0)$-mangrove and the arc-completed $(i, 0)$-mangrove;
    Compute the $(i, T)$-mangrove and the arc-completed $(i, T)$-mangrove;
    Set $L^i := ((i, 0), (i, T))$;
    $Resolved := Resolved \cup \{(i, T)\}$;
Initialize $LB = -\infty$ and $UB = \infty$;
while $(LB < UB)$ do
    Construct lower and upper bound TENs, with their UTTs and travel times, respectively:
     $(\mathcal{D}_{LB}, \underline{c}) \leftarrow createLBTEN(\{L^i\}_{i \in N}, Resolved)$, $(\mathcal{D}_{UB}, \overline{c}) \leftarrow createUBTEN(\{L^i\}_{i \in N})$;
    Compute current lower and upper bounds along with their corresponding path:
     $(P_{LB}, LB) \leftarrow computeSP(\mathcal{D}_{LB}, \underline{c})$, $(P_{UB}, UB) \leftarrow computeSP(\mathcal{D}_{UB}, \overline{c})$;
    $BP \leftarrow findBP(P_{LB}, \{L^i\}_{i \in N})$ ;
    for $(j, t) \in BP$ do
        Compute the $(j, t)$-mangrove and the arc-completed $(j, t)$-mangrove;
        Insert $(j, t)$ in the list $L^j$;
        if $(j, t^-)$ is in $L^j$ immediately before $(j, t)$ and no breakpoint at $j$ is between them then
            $Resolved := Resolved \cup \{(j, t^-)\}$
        if $(j, t^+)$ is in $L^j$ immediately after $(j, t)$ and no breakpoint at $j$ is between them then
            $Resolved := Resolved \cup \{(j, t)\}$
return $(UB, P_{UB})$
```

5 Computational Study

To analyze the performance of the algorithms presented in the previous section, we apply them to randomly generated instances. The algorithms are implemented in Matlab 2017b and run on LENOVO S1 Yoga with Intel Core i7-4500U at 1.80GHz, 8GB RAM.

5.1 Instance generation

To investigate the effect of network density on the performance of the algorithms, we consider four different types of networks $D = (N, A)$ with node set $N = \{1, 2, \ldots, n\}$:

Type 1: $A = \{(i, j) \in N \times N \mid i < j\}$;

Type 2: $A = \{(i, i + 1) \mid i = 1, \ldots, n - 1\} \cup \{(i, j) \in N \times N \mid i < j + 1 \text{ and } U_{i,j} < 0.5\}$ where for each (i, j), the value $U_{i,j}$ is chosen (independently) from the uniform random distribution on $[0, 1]$, which we write as $U_{i,j} \sim U[0, 1]$;

Type 3: $A = \{(i, j) \in N \times N \mid i < j < i + 4\}$; and

Type 4: $A = \{(i, i + 1) \mid i = 1, \ldots, n - 1\} \cup \{(i, j) \in N \times N \mid i < j + 1 \text{ and } U_{i,j} < 1/|j - i|\}$ with $U_{i,j} \sim U[0, 1]$ for each (i, j).

In Type 1 networks, there is an arc between all (unordered) pairs of nodes; the density of the network is 0.5. In Type 2 networks, half of the arcs present in Type 1 networks are selected, but to ensure there exists at least one feasible path, all arcs between consecutive nodes are included; the density of the network is around 0.25. In Type 3 networks, each node has arcs to its three numerically next nodes; the density of the network is around $3/n$. Finally, Type 4 networks have arcs that depend on the distance between the nodes, the further apart (numerically) a pair of nodes, the less likely there is an arc between them, but to ensure there exists at least one feasible path, all arcs between consecutive nodes are included; the density of the network is about $\log n!/(n^2)$.

To investigate the effect of the travel time functions on the performance of the algorithms, we consider three types of travel time function. An illustrative example of each of the three types is shown in Figure 11. The travel time functions are constructed, for each arc (i, j), to be the piecewise linear interpolant (at each integer point) of a function, f, described below.

Type 1: f is a degree 4 polynomial, uniquely determined by its values at the 5 points, given in componentwise vector form, as

$$f([0, T/4, T/2, 3T/4, T]) = \begin{cases} [1.6, 1, 1.05, 1, 1.6](B_{i,j}|j-i|/10), & \text{if } U_{i,j} < 1/3, \\ [2, 1, 1.5, 1, 2](B_{i,j}|j-i|/10), & \text{if } 1/3 \le U_{i,j} < 2/3, \\ [2.5, 1, 1.75, 1, 2.5](B_{i,j}|j-i|/10), & \text{otherwise,} \end{cases}$$

with $B_{i,j}, U_{i,j} \sim U[0, 1]$.

Type 2: f is a degree 6 polynomial, uniquely determined by its values at the 7 points, given in componentwise vector form, as

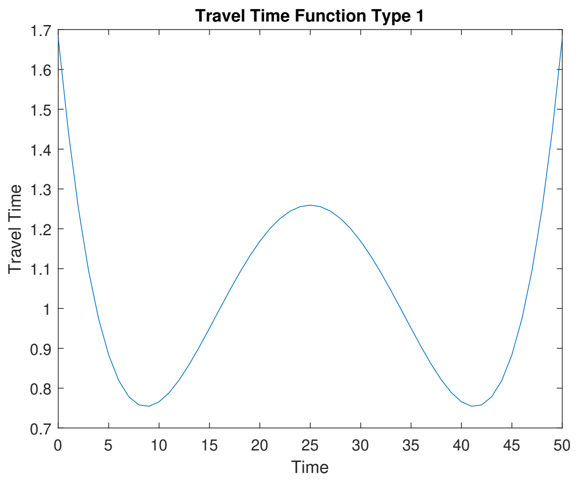
$$f([0, T/6, T/3, T/2, 2T/3, 5T/6, T]) = \begin{cases} [1, 1.6, 1, 1.05, 1, 1.6, 1](B_{i,j}|j-i|/10), & \text{if } U_{i,j} < 1/3, \\ [1, 2, 1, 1.5, 1, 2, 1](B_{i,j}|j-i|/10), & \text{if } 1/3 \le U_{i,j} < 2/3, \\ [1, 2.5, 1, 1.75, 1, 2.5, 1](B_{i,j}|j-i|/10), & \text{otherwise,} \end{cases}$$

with $B_{i,j}, U_{i,j} \sim U[0, 1]$.

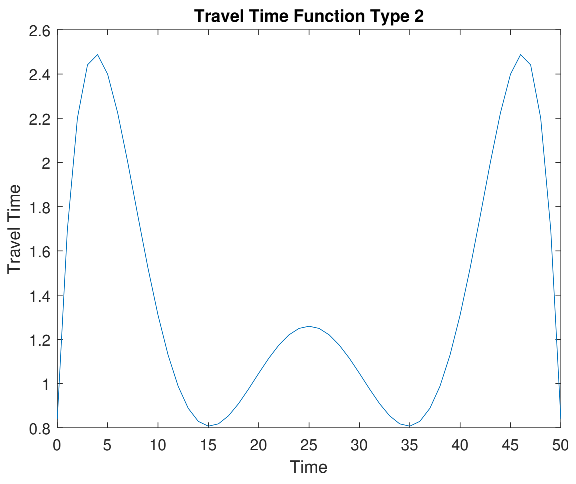
Type 3: $f(t) = (j - i) + \sin(B_{i,j} \times t)$, with $B_{i,j} \sim U[0, 1]$.

The first type of travel time function is inspired by Figliozzi (2012), who constructs travel *speed* functions by dividing the time horizon into five intervals of equal length and assigning each of the five intervals a constant travel speed. He takes a base time-independent travel speed, and then assigns travel speeds equal to 1.6, 1.0, 1.05, 1.0, and 1.6 times the base travel speed to the first, second, third, fourth, and fifth interval, respectively. In the same spirit, we have travel *time* functions that are equal to 1.6, 1.0, 1.05, 1.0, and 1.6 times the base travel time (which is random) at times 0, $^1\!/_4\,T$, $^1\!/_2\,T$, $^3\!/_4\,T$, and T, respectively. However, to create a continuous travel time function, we take the relevant polynomial interpolant through these points. It is then converted into a piecewise linear function by taking the piecewise linear interpolant of the polynomial sampled at integer points. The second type of travel time function is an extension of the first with two additional troughs for added difficulty. We have experimented with other travel time functions proposed in Figliozzi (2012), but found that with these travel time functions, the algorithms find both the minimum duration and minimum travel time solutions in a few iterations. In particular, we found that with these travel time functions the optimal solutions typically depart at the earliest or at the latest possible time. Thus these travel time functions are not very helpful for developing insights into the performance of the algorithms. The third type of travel time function is the most challenging for our algorithms as the travel time functions are not in phase for the arcs and have many local minima and maxima.

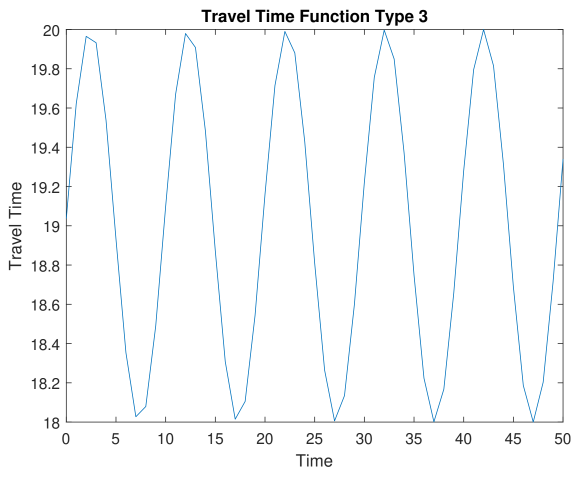
The performance of both DDD algorithms critically depends on the calculation of UTTs: being able to efficiently compute the minimum arc travel time in a start time interval. This can be accomplished by efficiently pre-computing a look-up table.



(a) Travel Time Type 1



(b) Travel Time Type 2



(c) Travel Time Type 3

Figure 11: Plots of different travel time function types

5.2 Algorithm configuration

In this section, we investigate the impact of the choice of the next breakpoint to explore on the performance of the algorithms. For the MDP, suppose the current lower bound is given by LB^t and t^+ is the arrival time at n of the next ABSPT in the list after $\overline{\mathcal{B}}^t$. Denote timed nodes in $\overline{\mathcal{B}}^t$ by (i, t_i) and timed nodes in $\overline{\mathcal{B}}^{t^+}$ by (i, t_i^+). We select (i, j) to be an outgoing arc from a timed node in the ABSPT $\overline{\mathcal{B}}^t$ that has a breakpoint in its travel time function within the interval (t_i, t_i^+), and that is first encountered when checking the nodes from 1 to n, in index order. We consider the following three alternative schemes for selecting one of the breakpoints of $c_{i,j}(t)$ lying within (t_i, t_i^+):

1. pick a random breakpoint (Rand),
2. pick the median breakpoint (with the breakpoints chronologically listed) (Med), and
3. pick the minimizer of $c_{i,j}(t)$ over t a breakpoint with $t \in (t_i, t_i^+)$ (Min).

For the MTTP, we furthermore compare adding a single breakpoint (S), a breakpoint for one of the travel subpaths, in each iteration, and adding multiple breakpoints (M), a breakpoint for each of the travel subpath, in each iteration. To select which breakpoint(s) to add, we identify the sequence of arc-completed mangroves used by the UTT path, P_{LB}. (Each travel subpath of P_{LB} occurs in an arc-completed mangrove.) For such an arc-completed mangrove, let (i, t) be the breakpoint used to generate it. Then, if (i, t^+) is the next element in L^i after (i, t), and there is a breakpoint (not yet explored) at i between t and t^+, we may select one such breakpoint to explore; such breakpoints are candidates for selection. Which one is selected is dictated by the scheme used: Rand, Med or Min. In the case of the (S) strategy, a breakpoint at i is selected for only the first arc-completed i-mangrove used by the UTT path, encountered in traversing it from node 1 to node n, for which a candidate breakpoint exists. In the case of the (M) strategy, a breakpoint is added for every arc-completed mangrove used by the UTT path for which a candidate breakpoint exists.

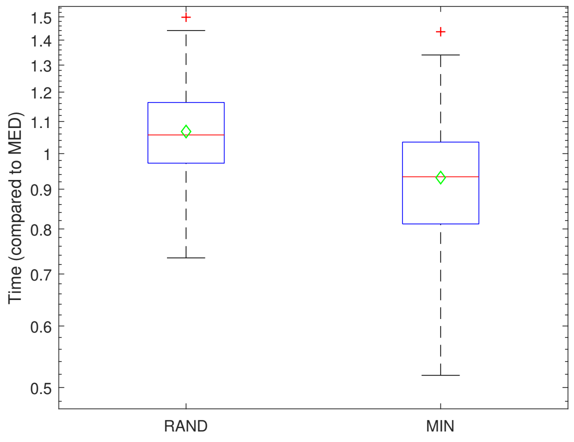
First, we consider the impact of the breakpoint selection scheme when solving MDP instances. The results can be found in Figure 12 where we show box-plots of the ratios solve time, iteration count, and number of breakpoints explored for schemes Rand and Min compared to scheme Med. The mean of each box-plot is represented by the green diamond and the median by the red horizontal line. Note that the y-axis is given in a logarithmic scale. We observe that the best choice among the three breakpoint selection scheme is Min, which does best on average in all performance metrics (solve time, number of iterations, and number of breakpoints explored).

Next, we consider the impact of the breakpoint selection scheme and the impact of the number of breakpoints explored each iteration when solving MTTP instances. The results can be found in Figure 13 where we show box-plots of the ratios solve time, iteration count, and number of breakpoints explored for schemes (Rand, S), (Rand, M), (Med, M), (Min, S), and (Min, M) compared to scheme (Med, S). Only instances with the first two travel times functions are used as the computation times with the third travel time function become quite large (more on this later). We observe that the best choice is to explore multiple breakpoints in each iteration and to select the breakpoints to explore using scheme Min, which does best on average in all performance metrics (solve time, number of iterations, and number of breakpoints explored). Furthermore, the benefits of exploring a breakpoint for each of the travel subpaths in an iteration are significant. It is interesting to see that (Rand, S) outperforms both (Med, S) and (Med, M) in terms of average solve time. The latter, especially, is somewhat surprising.

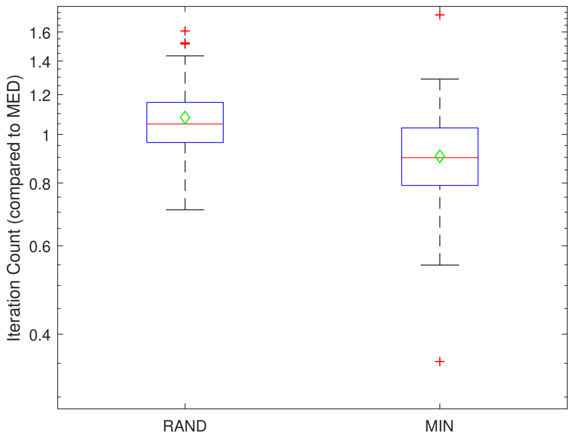
5.3 Benefits of dynamic discretization discovery

In this section, we compare the performance of our dynamic discretization discovery (DDD) algorithms to the performance of the breakpoint enumeration algorithms, where we use breakpoint selection scheme Min and, in the case of MTTP, we explore multiple breakpoints in each iteration.

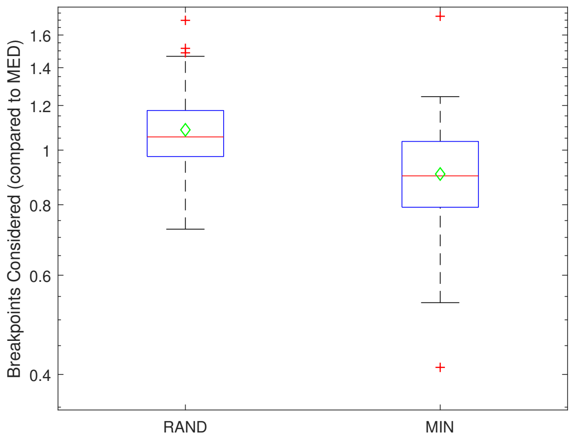
The results for MDP instances can be found in Table 1, where we present averages over 10 randomly generated instances for networks of the different types and of different sizes (n = 20, 30, and 50, and



(a) Effect on solve time

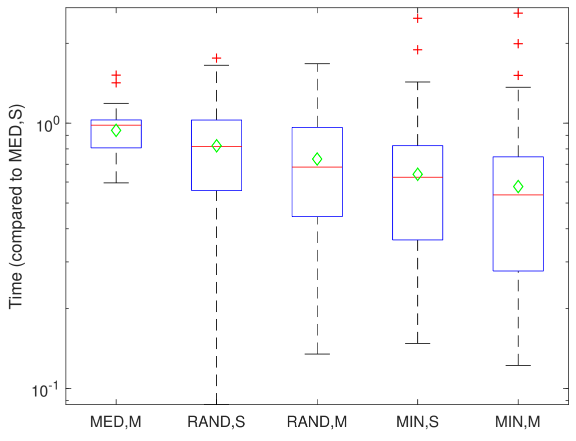


(b) Effect on iteration count

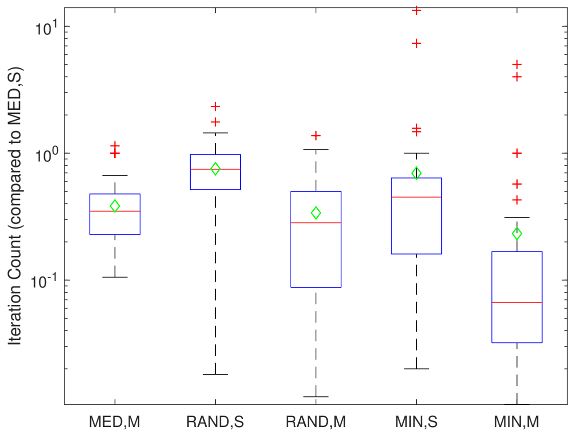


(c) Effect on number of breakpoints explored

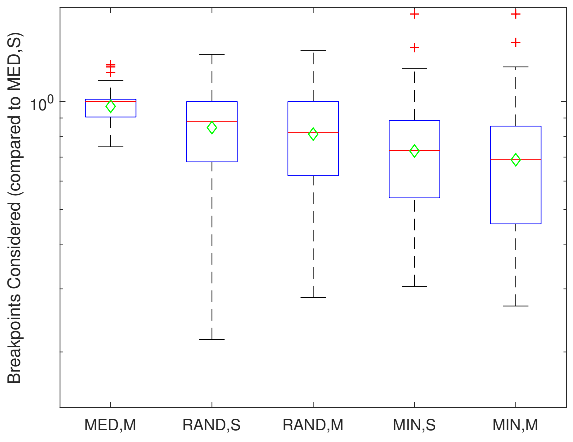
Figure 12: Comparison of breakpoint selection schemes on 120 instances of the MDP with $n = 20$, $T = 50$



(a) Effect on solve time



(b) Effect on iteration count



(c) Effect on number of breakpoints explored

Figure 13: Comparison of breakpoint selection schemes on 80 instances of the MTTP with $n = 20$, $T = 50$

$T = 50$) and for the different types of travel time function. We report the average number of breakpoints explored by our DDD algorithm, the number of breakpoints explored by the enumeration algorithm (i.e., the number of breakpoints in the instances), and the fraction of the number of breakpoints investigated by the DDD algorithm (given as a percentage). Furthermore, we report the solve time in seconds for both the DDD algorithm, the enumeration algorithm and the solve time of the DDD as a fraction of the solve time of the enumeration algorithm (again, given as a percentage), and the (average) number of arcs in the optimal solution. Note that for these instances the number of breakpoints is $|N-1| \times (T-1) + 2$, and not $|A| \times (T-1) + 2$ as in Foschini et al. (2014), since for arcs with a common tail node and breakpoint, we only need to explore the breakpoint once.

Table 1: MDP results.

n	ntype	ttype	BP	Total # BP	% BP	Avg. Time DDD (s)	Avg. Time Enum (s)	% Time	Opt. Arcs
		1	63.5	933	6.8	5.3	15.3	34.7	3.7
20	1	2	59.3	933	6.4	4.2	12.3	34.1	4.1
		3	131.9	933	14.1	7.9	12.6	62.6	12.0
		1	54.3	933	5.8	2.2	8.1	27.3	3.9
20	2	2	59.1	933	6.3	2.8	8.0	34.6	4.7
		3	134.5	933	14.4	4.7	8.2	57.7	13.5
20	3	1	98.3	933	10.5	2.2	6.4	35.0	10.7
20	3	2	89.5	933	9.6	2.3	6.4	35.7	10.5
20	3	3	138.0	933	14.8	3.1	6.4	48.8	12.1
20	4	1	77.1	933	8.3	1.9	5.8	33.2	7.2
20	4	2	69.9	933	7.5	1.7	5.8	29.5	7.2
20	4	3	131.3	933	14.1	3.4	5.9	57.8	13.4
30	1	1	61.4	1423	4.3	9.0	37.3	24.0	3.9
30	1	2	67.5	1423	4.7	9.3	37.2	24.9	3.9
30	1	3	185.5	1423	13.0	22.8	38.3	59.6	18.5
30	2	1	61.0	1423	4.3	4.9	23.0	21.4	4.4
30	2	2	62.3	1423	4.4	5.0	23.1	21.5	4.4
30	2	3	160.7	1423	11.3	11.1	23.6	47.0	18.5
30	3	1	138.0	1423	9.7	4.4	14.3	30.9	16.0
30	3	2	121.1	1423	8.5	3.8	14.4	26.5	16.2
30	3	3	185.5	1423	13.0	5.9	14.4	41.3	19.3
30	4	1	96.7	1423	6.8	3.4	14.3	23.9	7.9
30	4	2	77.2	1423	5.4	3.2	14.3	22.2	7.9
30	4	3	144.5	1423	10.2	5.3	14.5	36.8	19.3
50	1	1	82.5	2403	3.4	29.1	190.0	15.3	3.6
50	1	2	78.0	2403	3.2	26.7	189.6	14.1	3.6
50	1	3	184.3	2403	7.7	56.3	173.2	32.5	30.3
50	2	1	82.6	2403	3.4	15.2	124.4	12.2	5.0
50	2	2	76.5	2403	3.2	14.4	95.7	15.0	5.0
50	2	3	162.3	2403	6.8	27.0	95.8	28.2	28.7
50	3	1	161.4	2403	6.7	8.1	40.3	20.1	25.2
50	3	2	158.4	2403	6.6	7.9	40.8	19.3	25.3
50	3	3	199.4	2403	8.3	10.2	40.3	25.2	32.0
50	4	1	148.1	2403	6.2	8.5	43.0	19.8	11.9
50	4	2	127.3	2403	5.3	7.8	43.4	17.9	11.8
50	4	3	149.4	2403	6.2	8.9	43.8	20.4	32.1

The results show that the DDD algorithm investigates only a small fraction (3.2% to 14.8%) of the number of breakpoints. The results clearly demonstrate that the third type of travel time function is the most challenging for the DDD algorithm, with more breakpoints being explored and, as a consequence, higher solve times. As expected, the enumeration algorithm is insensitive to the type of travel time function, but is sensitive to the network density. The result also show that as the instance size increases, the benefits of using the DDD algorithm (as opposed to using the enumeration algorithm) also increases.

The results for MTTP instances can be found in Table 2, where we present averages over 10 randomly generated instances for networks of different types and sizes ($n = 20$ and 30, and $T = 50$) and for different travel time functions. Note that we have omitted travel time function type 3, because the solve times are significantly longer (for example, instances of network type 2 with 20 nodes and planning horizon 50 solve in about an hour), in part because the optimal solutions tend to have many more arcs (for example, instances of network type 2 with 20 nodes and planning horizon 50 have about 5 arcs in an optimal solution for travel time functions type 1 and 2 compared to about 17 arcs for travel time function type 3).

Table 2: MTTP results.

n	ntype	ttype	BP	Total # BP	% BP	Avg. Time DDD (s)	Avg. Time Enum (s)	% Time	Subpaths	Opt. Arcs
20	1	1	67.7	933	7.3	118.7	1505.2	7.9	2.0	3.7
20	1	2	51.9	933	5.6	77.3	1503.1	5.1	2.6	4.1
20	2	1	49.8	933	5.3	55.2	1534.7	3.6	1.4	3.9
20	2	2	45.0	933	4.8	45.3	1527.6	3.0	1.6	4.7
20	3	1	155.5	933	16.7	286.4	1570.4	18.2	3.1	10.6
20	3	2	128.4	933	13.8	244.8	1599.2	15.3	3.7	10.5
20	4	1	83.6	933	9.0	110.5	1654.2	6.7	2.2	7.4
20	4	2	51.0	933	5.5	48.5	1639.6	3.0	2.2	6.8
30	1	1	66.9	1423	4.7	249.1	9463.8	2.6	1.3	3.9
30	1	2	62.0	1423	4.4	196.8	9618.7	2.0	1.4	3.9
30	2	1	83.8	1423	5.9	285.8	9423.0	3.0	1.6	4.4
30	2	2	72.4	1423	5.1	215.5	10132.5	2.1	1.9	4.4
30	3	1	299.7	1423	21.1	2462.4	9506.8	25.9	3.7	16.3
30	3	2	341.5	1423	24.0	2886.1	9530.9	30.3	4.1	16.1
30	4	1	144.7	1423	10.2	671.0	9860.0	6.8	2.6	7.9
30	4	2	131.9	1423	9.3	631.5	9677.6	6.5	3.1	7.9

The results show that the DDD algorithm investigates only a small fraction (4.4% to 24.0%) of the number of breakpoints. The results also reveal that MTTP is indeed a much harder problem than MDP, as it requires significantly larger solve times. It is interesting to observe that the DDD algorithm has noticeably higher solve times for instances of network type 3, but that these instances also have significantly more arcs in the optimal solutions (on average more than double compared to the other network types). Another observation is that the enumeration algorithm is less affected by the network type, with similar solve times for all network types and travel time function types. This is likely due to the fact that the number of nodes in the full time-expanded network is the same for the four network types. This is a consequence of the connectivity of the networks, it is possible to get from any i to any $j > i$ in all network types, and, therefore, the mangroves are all the same size.

6 Real-world case study

To supplement the study of random instances, we also run the DDD algorithm for the MDP on real-world data from part of the Atlanta road network, which comprises 306 nodes and 618 arcs and is obtained via Open Street Map. A map view of the area as well as the corresponding network representation are shown in Figure 14.

The arc travel times were derived from GPS traces obtained from a variety of input streams (e.g., navigational systems and cell phones) and snapped to the nearest arc. The collected data are then aggregated to create piecewise linear arc travel time functions with breakpoints at every 15 minutes for a 24-hour period, giving a total of 192 data points for each arc (Marshall 2019). For a sample of the arc travel time functions, see Figure 15.

To illustrate the difference between a minimum duration, an earliest arrival, and a latest departure paths from an origin to a destination is a specific time interval, we selected an origin in North Atlanta, a destination in Atlanta's Midtown, and a time interval from 9:00am to 1:00pm. The earliest arrival path starts at the origin at 9:00am, the latest arrival path ends at the destination at 1:00pm, and the minimum duration path can start at any time within the [09:00, 13:00] interval.

The minimum duration path (see Figure 16(b)) leaves at 10:37am and takes 45.8 minutes. It is different from both the earliest arrival and latest departure paths, which coincide (see Figure 16(a)), and take 47.5 and 47.8 minutes, respectively. The minimum duration path departs in the lull between the morning and noon rush hours, and uses the most direct path (taking the interstates I-75 and I-75/85), whereas the earliest arrival/latest departure path avoids the interstates and instead uses streets in residential neighborhoods in the eastern part of Midtown (Peachtree St. and Juniper St.). The apparent detour at the end of the path is due to one-way streets.

Although the above example illustrates that the minimum duration path can be different from the earliest arrival and latest departure paths, we have found that the difference in duration is not that large. This is likely due to the fact that the chosen section of Atlanta is relatively small. Furthermore, this section has few schools, and school zones are known to greatly influence travel times at certain times of the day.

Finding the minimum duration path in the network representing this section of Atlanta took a few seconds, which shows that the technology developed might be useful in real-life applications.

7 Concluding Remarks

We have given new dynamic discrete discovery algorithms for time-dependent shortest path problems where the travel time on an arc is a piecewise linear function of the departure time on the arc. These algorithms make use of the breakpoints of the piecewise linear functions and the previous result of the existence of an algorithm polynomial in the total number of breakpoints for the minimum duration objective. However our algorithms, in contrast with the earlier result, investigate only a small fraction of the travel time function breakpoints in searching for an optimal path and proving its optimality. The two problems we have studied are the minimum duration time-dependent path problem MDP, i.e., minimizing the time between arrival at the sink and departure from the source, and the minimum travel time-dependent path problem MTTP, i.e., minimizing the total travel time from the source to the sink. In the MDP waiting is allowed at the source and in the MTTP waiting is allowed at all nodes except the sink. But in both problems there are no costs or constraints associated with waiting. In a subsequent paper (He et al. 2019), we have extended our results to consider a linear cost penalty for waiting or a constraint on the total amount of waiting allowed. There are several variations depending on the cost and constraint parameters, many of which can be solved by algorithms similar to the ones for MDP and MTTP and others that are NP- hard.

Acknowledgment

We would like to thank Luke Marshall (and Microsoft Research) for supplying the road network and travel time data for our case study. This material is based upon work supported by the National Science Foundation under Grant No. 1662848.

References

Bellman, Richard. 1958. On a routing problem. *Quarterly of applied mathematics* **16**(1) 87–90.

Boland, Natashia, Mike Hewitt, Luke Marshall, Martin Savelsbergh. 2017. The continuous-time service network design problem. *Operations Research* **65**(5) 1303–1321.

Chabini, Ismail. 1998. 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) 170–175.

Cooke, Kenneth L, Eric Halsey. 1966. The shortest route through a network with time-dependent internodal transit times. *Journal of mathematical analysis and applications* **14**(3) 493–498.

Dean, Brian C. 2004. Shortest paths in fifo time-dependent networks: Theory and algorithms. *Rapport technique, Massachusetts Institute of Technology* .

Demiryurek, Ugur, Farnoush Banaei-Kashani, Cyrus Shahabi, Anand Ranganathan. 2011. Online computation of fastest path in time-dependent spatial networks. *International Symposium on Spatial and Temporal Databases*. Springer, 92–111.

Ding, Bolin, Jeffrey Xu Yu, Lu Qin. 2008. Finding time-dependent shortest paths over large graphs. *Proceedings of the 11th international conference on Extending database technology: Advances in database technology*. ACM, 205–216.

Figliozzi, Miguel Andres. 2012. The time dependent vehicle routing problem with time windows: Benchmark problems, an efficient solution algorithm, and solution characteristics. *Transportation Research Part E: Logistics and Transportation Review* **48**(3) 616–636.

Ford, Lester Randolph, Delbert Ray Fulkerson. 1962. *Flows in networks*. Princeton University Press.

Foschini, Luca, John Hershberger, Subhash Suri. 2014. On the complexity of time-dependent shortest paths. *Algorithmica* **68**(4) 1075–1097.

Gunturi, Venkata M.V., Kenneth Joseph, Shashi Shekhar, Kathleen M. Carley. 2012. Information lifetime aware analysis for dynamic social networks. Tech. rep., University of Minnesota.

He, Edward, Natashia Boland, George Nemhauser, Martin Savelsbergh. 2019. Computational complexity of time-dependent shortest path problems. *Optimization Online* Eprint ID 2019–02–7083.

Jabali, Ola, T Van Woensel, AG De Kok. 2012. Analysis of travel times and co2 emissions in time-dependent vehicle routing. *Production and Operations Management* **21**(6) 1060–1074.

Kanoulas, Evangelos, Yang Du, Tian Xia, Donghui Zhang. 2006. Finding fastest paths on a road network with speed patterns. *Data Engineering, 2006. ICDE'06. Proceedings of the 22nd International Conference on*. IEEE, 10–10.

Marshall, Luke. 2019. Private communication .

Nachtigall, Karl. 1995. Time depending shortest-path problems with applications to railway networks. *European Journal of Operational Research* **83**(1) 154–166.

Omer, Jérémy, Michael Poss. 2018. Time-dependent shortest path with discounted waits. URL `https://hal.archives-ouvertes.fr/hal-01836007`. Working paper or preprint.

Orda, Ariel, Raphael Rom. 1990. Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length. *Journal of the ACM (JACM)* **37**(3) 607–625.

Appendix. Arc travel time functions for the example shown in Figure 4

8 Arc travel time functions for the example shown in Figure 4

Below are the functional forms of the arc travel time functions for the example given in Figure 4. These show the forward travel time on the arc as a function of the departure time at its tail node. Alongside each is shown the reverse travel time function, which gives the travel time on the arc as a function of the arrival time at its head node. The reverse travel time at each breakpoint of the reverse travel time function of each arc is shown in Table 3.

$$c_{1,2}(t) = \begin{cases} 1.34 - 0.68t, & 0 \leq t < 1 \\ 1.18 - 0.52t, & 1 \leq t < 2 \\ 0.40 - 0.13t, & 2 \leq t < 3 \\ -1.01 + 0.34t, & 3 \leq t < 4 \\ -2.25 + 0.65t, & 4 \leq t < 5 \end{cases} \qquad c^{rev}_{1,2}(t) = \begin{cases} 1.34 - 2.13t, & 1.34 \leq t < 1.66 \\ 0.66 - 1.08t, & 1.66 \leq t < 2.14 \\ 0.14 - 0.15t, & 2.14 \leq t < 3.01 \\ 0.01 + 0.25t, & 3.01 \leq t < 4.35 \\ 0.35 + 0.39t, & 4.35 \leq t < 6.00 \end{cases}$$

$$c_{1,3}(t) = \begin{cases} 2.85 + 0.10t, & 0 \leq t < 1 \\ 2.90 + 0.05t, & 1 \leq t < 2 \\ 3.04 - 0.02t, & 2 \leq t < 3 \\ 3.22 - 0.08t, & 3 \leq t < 4 \\ 3.46 - 0.14t, & 4 \leq t < 5 \end{cases} \qquad c^{rev}_{1,3}(t) = \begin{cases} 2.85 + 0.09t, & 2.85 \leq t < 3.95 \\ 2.95 + 0.05t, & 3.95 \leq t < 5.00 \\ 3.00 - 0.02t, & 5.00 \leq t < 5.98 \\ 2.98 - 0.09t, & 5.98 \leq t < 6.90 \\ 2.90 - 0.16t, & 6.90 \leq t < 7.76 \end{cases}$$

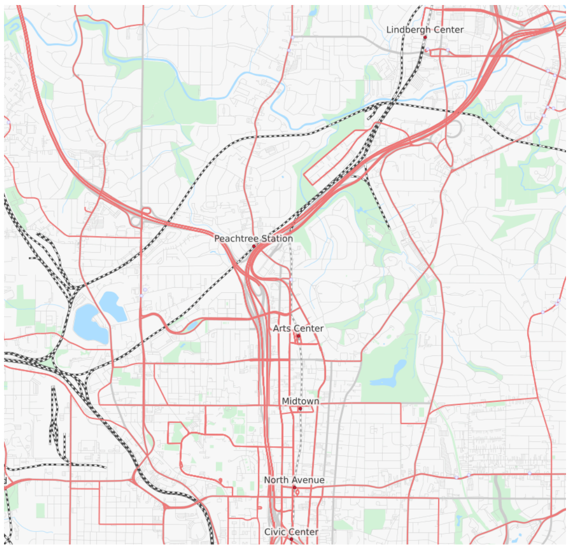
$$c_{2,3}(t) = \begin{cases} 1.99 - 0.17t, & 0 \leq t < 1 \\ 2.13 - 0.31t, & 1 \leq t < 2 \\ 2.33 - 0.41t, & 2 \leq t < 3 \\ 2.39 - 0.43t, & 3 \leq t < 4 \\ 2.15 - 0.37t, & 4 \leq t < 5 \end{cases} \qquad c^{rev}_{2,3}(t) = \begin{cases} 1.99 - 0.20t, & 1.99 \leq t < 2.82 \\ 1.82 - 0.45t, & 2.82 \leq t < 3.51 \\ 1.51 - 0.69t, & 3.51 \leq t < 4.10 \\ 1.10 - 0.75t, & 4.10 \leq t < 4.67 \\ 0.67 - 0.59t, & 4.67 \leq t < 5.30 \end{cases}$$

$$c_{2,4}(t) = \begin{cases} 1.29 - 0.27t, & 0 \leq t < 1 \\ 0.41 + 0.61t, & 1 \leq t < 2 \\ -0.25 + 0.94t, & 2 \leq t < 3 \\ 1.28 + 0.43t, & 3 \leq t < 4 \\ 4.84 - 0.46t, & 4 \leq t < 5 \end{cases} \qquad c^{rev}_{2,4}(t) = \begin{cases} 1.29 - 0.37t, & 1.29 \leq t < 2.02 \\ 1.02 + 0.38t, & 2.02 \leq t < 3.63 \\ 1.63 + 0.48t, & 3.63 \leq t < 5.57 \\ 2.57 + 0.30t, & 5.57 \leq t < 7.00 \\ 3.00 - 0.85t, & 7.00 \leq t < 7.54 \end{cases}$$

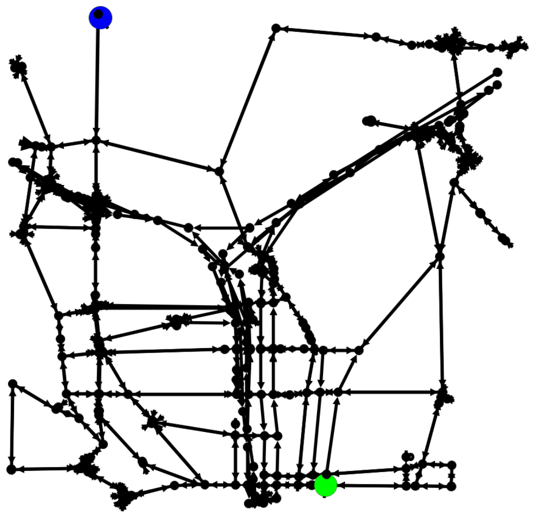
$$c_{3,4}(t) = \begin{cases} 0.61 + 0.12t, & 0 \leq t < 1 \\ 0.63 + 0.10t, & 1 \leq t < 2 \\ 0.72 + 0.06t, & 2 \leq t < 5 \end{cases} \qquad c^{rev}_{3,4}(t) = \begin{cases} 0.61 + 0.11t, & 0.61 \leq t < 1.73 \\ 0.73 + 0.09t, & 1.73 \leq t < 2.83 \\ 0.83 + 0.05t, & 2.83 \leq t < 6.00 \end{cases}$$

BP Time	(1,2)	BP Time	(1,3)	BP Time	(2,3)	BP Time	(2,4)	BP Time	(3,4)
1.34	1.34	2.85	2.85	1.99	1.99	1.29	1.29	0.61	0.61
1.66	0.66	3.95	2.95	2.82	1.82	2.02	1.02	1.73	0.73
2.14	0.14	5.00	3.00	3.51	1.51	3.63	1.63	2.83	0.83
3.01	0.01	5.98	2.98	4.10	1.10	5.57	2.57	—	—
4.35	0.35	6.90	2.90	4.67	0.67	7.00	3.00	—	—
6.00	1.00	7.76	2.76	5.30	0.30	7.54	2.54	6.00	1.00

Table 3: Reverse Arc Travel Times at Each Breakpoint (BP)

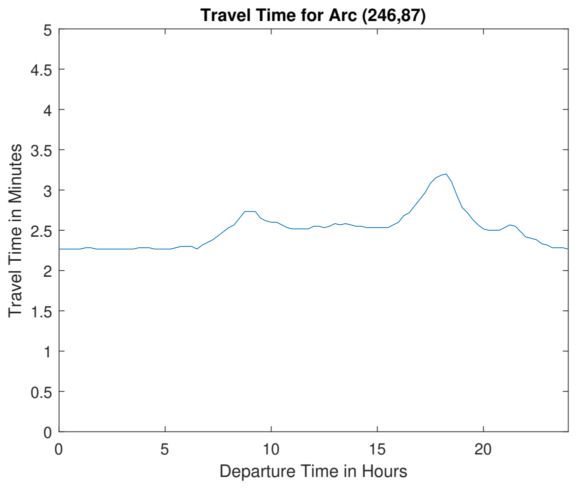


(a) Map of section of Midtown and North Atlanta

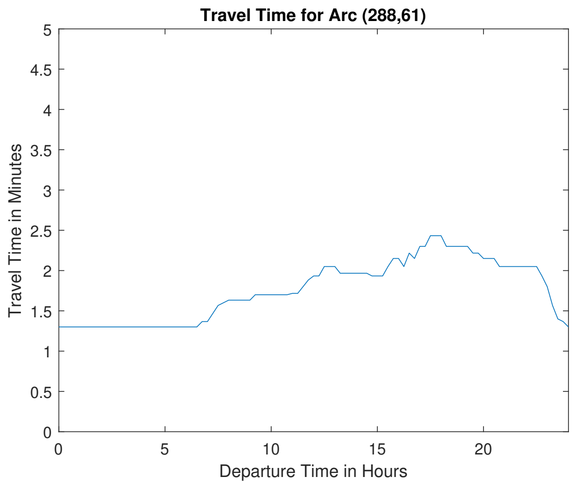


(b) Network representation

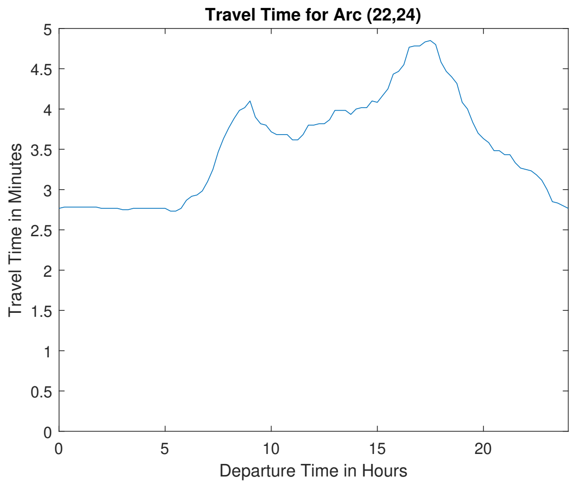
Figure 14: Map and corresponding network representation of of the section of Midtown and North Atlanta.



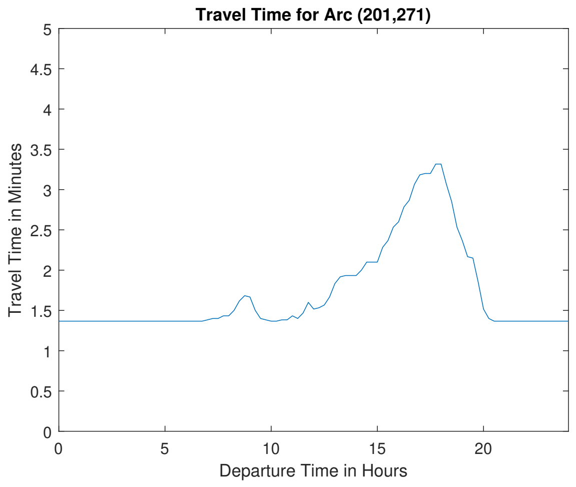
(a) Travel time of an arc along Peachtree St.



(b) Travel time of an arc along Ponce de Leon Ave. NE

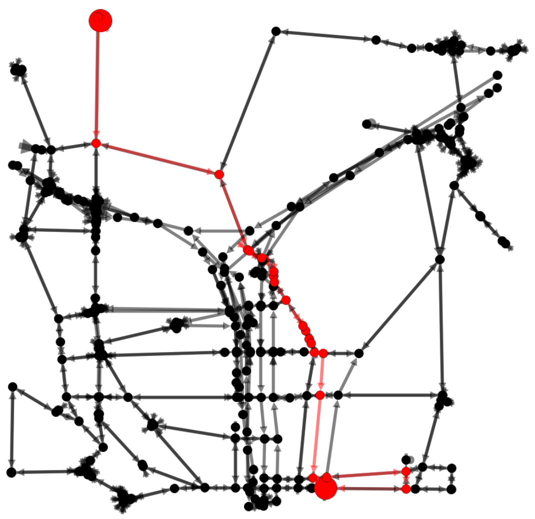


(c) Travel time of an arc along North Ave. NE

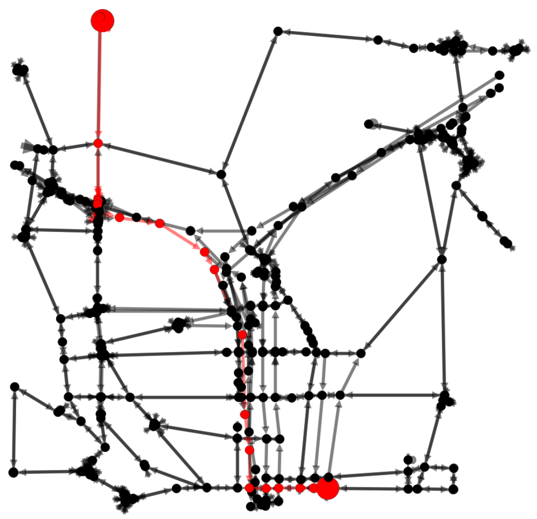


(d) Travel time of an arc along Midtown section of I-75/85

Figure 15: Travel times for four different arcs in Atlanta.



(a) Earliest arrival / latest departure path



(b) Minimum duration path

Figure 16: An example where the minimum duration path differs from the earliest arrival/latest departure path.