Shortest Journeys in Directed Temporal Graphs*

Siu-Wing Cheng
Department of Computer Science and Engineering, HKUST,
Hong Kong, China
scheng@cse.ust.hk

Received (Day Month Year)
Accepted (Day Month Year)
Communicated by (xxxxxxxxxx)

Consider a directed temporal graph $G = (V, E)$ with time ranges on the edges. There can be more than one range on an edge, and each range carries a positive traversal time. Let $n = |V|$ and let Δ be the total number of time ranges in G. We assume that $\Delta = \Omega(n)$. We study the problem of computing shortest journeys that start from a fixed source vertex s within a given time interval $[t_b, t_e)$, where the cost of a journey is equal to the sum of traversal times of the edges on it at the times of crossing those edges. We can construct in $O(\Delta^2 \log \Delta)$ time a data structure of size $O(n\Delta)$ such that for any vertex v and any time t, we can report in $O(\log \Delta)$ time the cost of the shortest journey that starts from s within $[t_b, t_e)$ and arrives at v no later than t. The journey achieving the reported cost can be produced in time linear in its complexity.

Keywords: Temporal graphs; journey; continuous Dijkstra; priority search tree.

1. Introduction

The relations among a set of entities such as people, organisms, or cities may evolve over time. There is a natural interest in finding a path in such a network of entities that respect the time constraints. For example, in finding a sequence of connecting flights from a city A to another city B, one may optimize the total flight time or the elapsed time. A *temporal graph* is a construction that has been proposed to model connections that may change dynamically [6, 8, 9, 10, 14, 15]. The term evolving graph is used by some authors, but the term temporal graph is more popular.

Two representations of a temporal graph are proposed in [15]. The first one is a directed or undirected graph $G = (V, E)$ such that each edge $e \in E$ has a set $R(e)$ of interior-disjoint closed time intervals. We call these time intervals *ranges*. It signifies that an edge $e \in E$ is present during every range $\tau \in R(e)$, but e is absent at any time outside $\bigcup_{\tau \in R(e)} \tau$. The second representation assumes that there is a discrete universe of time instances of size U, which can be assumed without loss of generality to be integers in the range $[1, U]$. The temporal graph is specified

*Research supported by Research Grants Council, Hong Kong , China (project no. 16203718).

as a sequence of subgraphs $G_1, G_2, \ldots, G_U$ of G. It signifies that the vertices and edges in G_i are present at time step i. In this paper, we assume as in many other previous works (e.g. [6, 9, 10, 14]) that the vertices of G are always present, but its edges may appear and disappear. Both temporal graph representations require the full knowledge of the changes of the graph over time; for example, such a situation arises in inter-satellite communication as described in [15] and the flight example in the previous paragraph.

The first representation does not require the time instances to come from a discrete universe; the ranges can be continuous closed intervals or intervals of discrete time instances depending on the context; there is no *a priori* upper bound on the largest time instance in the input. We use the RAM model in this paper in which any real number can be stored in a computer word and it takes $O(1)$ time to perform an arithmetic operation. We call the first representation a *compact* temporal graph and the second representation a *discrete* temporal graph. In a discrete temporal graph, one can save storage by keeping a single vertex set and an array of U edge sets indexed by the time step. There is no difference in the asymptotical space complexity if there are $\Omega(n)$ edges in each subgraph G_i.

A *journey* in a temporal graph is a path that traverses edges in chronological order at times when those edges are available. It may be necessary to wait at an intermediate vertex for the appearance of the outgoing edge that one wants to take. Each edge carries a set of traversal times and a set of weights, which denote the duration required for crossing that edge and the cost of doing so at different times. The *cost* of a journey is the sum of the weights of edges on it at the times of crossing those edges. The *elapsed time* of a journey is the arrival time at the last vertex minus the start time at the first vertex. Let s be a fixed source vertex of G. The *single-source shortest journeys* problem is to compute journeys of minimum costs from s to all other vertices; the *single-source fastest journeys* problem is to compute journeys of minimum elapsed times from s to all other vertices. There are practical applications. For example, the reliable path problem in temporal graphs considered in [5] can be recast as a shortest journey problem. As another example, the fastest journeys problem finds application in efficient broadcast trees in time-varying networks [1].

For every edge $e \in E$ in a compact temporal graph, different traversal times and weights may be associated with different ranges in $R(e)$. These traversal times and weights can be any non-negative real number. For a discrete temporal graph, different traversal times and weights may be associated with the same edge in different subgraphs. All traversal times are integers in the range $[1, U]$ because if we start traversing an edge at an instance in $[1, U]$, we need to finish at a later instance in $[1, U]$. The weights can be any non-negative real number. An undirected compact temporal graph can be modeled by a directed compact temporal graph with each undirected edge represented by two parallel, oppositely oriented directed edges.

In [15], for a compact temporal graph (directed or undirected), the single-source shortest journeys problem is solved in $O(mD)$ time when the traversal time and

weight of every edge are both equal to 1, where $m = |E|$ and D is the diameter of G. For an undirected discrete temporal graph in which the traversal time of every edge is 1, an algorithm for the single-source fastest journeys problem is proposed in [15] that runs in $O\big(nU \log(nU) + \sum_{i=1}^{U} m_i\big)$ time, where m_i is the number of edges in G_i.[a] Later, algorithms for the single-source shortest and fastest journeys problems are proposed in [14] for a directed discrete temporal graph in which the weight of every edge is equal to its traversal time. The shortest and fastest journeys can be found in $O\big(\big(\sum_{i=1}^{U} m_i\big) \cdot \log\big(\sum_{i=1}^{U} m_i\big)\big)$ and $O\big(\big(\sum_{i=1}^{U} m_i\big) \log U\big)$ time, respectively. The above results in [14, 15] also hold when the journeys are required to start from s within a given time interval. In [10], for a discrete temporal graph in which the traversal time of every edge is 1, algorithms for the single-source fastest and shortest journeys problems are proposed. The algorithm for fastest journeys runs in $O\big(nU + \sum_{i=1}^{U} m_i\big)$ time. The algorithm for shortest journeys works with in the special case that each graph edge appears in exactly one time step, but the weight associated with an edge can be any positive number. The running time of the algorithm is $O(n^3)$.

A key feature, or difficulty depending on the viewpoint, is that given a shortest or fastest journey from s to v, a prefix of it from s to a vertex u may not be a shortest or fastest journey to u. Therefore, one cannot solve the problem by running Dijkstra's algorithm without paying attention to the arrival time at the destination.

In this paper, we study the problem of computing single-source shortest journeys in a directed compact temporal graph $G = (V, E)$ in which the weight and traversal time associated with a range are equal and positive. Let $n = |V|$ and let Δ be the total number of ranges in G. We assume that $\Delta = \Omega(n)$. Let s be a fixed source, and let $[t_b, t_e)$ be a given time interval. We can construct in $O(\Delta^2 \log \Delta)$ time a data structure of size $O(n\Delta)$ such that for any vertex v and any time t, we can report in $O(\log \Delta)$ time the cost of the shortest journey that starts from s within $[t_b, t_e)$ and arrives at v no later than t. The journey achieving the reported cost can be produced in time linear in its complexity. It is interesting to note that we borrow the continuous Dijkstra paradigm for computing shortest paths in geometric settings [2, 3, 4, 7, 11, 12] to solve this shortest journey problem.

The assumption of $\Delta = \Omega(n)$ is mild. If we ignore the edge directions in G and take the connected component containing s, the subgraph $H \subseteq G$ that spans the vertices of this connected component satisfies the assumption. Moreover, there is no journey from s to any vertex that does not belong to H.

Other related problems on temporal graphs have also been studied. In [6], the authors study the question of exploring a discrete temporal graph. If there are n vertices and the traversal time of every edge is equal to 1, it is easy to traverse the entire temporal graph in $O(n^2)$ time steps, which is also known to be necessary in the worst case. It is shown in [6] that if the temporal graph has vertex degree at

[a] Although an $O(nU \log(nU))$ running time is reported in [15], an $O\big(\sum_{i=1}^{U} m_i\big)$ term must have been omitted because a graph search is conducted to find the connected components of every G_i.

most d every time step, $O(d \log d \cdot n^2 / \log n)$ time steps suffice; the result can be improved to $O(n^2 / \log n)$ if d is a constant. In [9], it is shown that Menger's Theorem in its original formulation fails for temporal graphs and that finding disjoint s-t journeys in a temporal graph is NP-complete. On the other hand, a temporal analog of Menger's Theorem is proved in [10]. Optimization problems on enforcing connectivity in discrete temporal graphs are also studied in [10]. In particular,the authors study the problems of converting a directed graph to a discrete temporal graph so as to preserve all paths or all pairwise reachability in the underlying directed graph with the minimum U or $\sum_{i=1}^{U} m_i$.

2. Preliminaries

We use $G = (V, E)$ to denote the directed compact temporal graph that we work with. Let $n = |V|$. For every edge $e \in E$, $R(e)$ denotes the set of ranges associated with e, which are interior-disjoint closed time intervals. For every range $\tau \in R(e)$, e is present during τ; on the other hand, e is absent at any time outside $\bigcup_{\tau \in R(e)} \tau$. Without loss of generality, we assume that $R(e)$ is non-empty for all $e \in E$. We define $\Delta = \sum_{e \in E} |R(e)|$.

For every edge $e \in E$, every range $\tau \in R(e)$ is associated with a traversal time $\alpha_\tau(e) \in \mathbb{R}_{\geq 0}$ and a real-valued weight. The interpretation is that it takes $\alpha_\tau(e)$ time to cross e during the range τ, and the cost of doing so is the weight associated with τ. In this paper, we study the special case in which $\alpha_\tau(e)$ is positive and the weight associated with τ is equal to $\alpha_\tau(e)$. Note that e may have different traversal times in different ranges in $R(e)$.

Given any time interval or range τ, we use $\text{start}(\tau)$ and $\text{end}(\tau)$ to denote the start and end times of τ, respectively, and we use $|\tau|$ to denote $\text{end}(\tau) - \text{start}(\tau)$. For consistency, we assume that for every edge $e \in E$ and every range $\tau \in R(e)$, $\alpha_\tau(e) \leq |\tau|$ so that the edge e can be traversed completely within τ.

A *journey* from a vertex v_1 to another vertex v_k is a sequence of ordered pairs $(v_1, t_1), (v_2, t_2), \ldots, (v_k, t_k)$ signifying that the trip starts from v_1 at time t_1 and visits v_i at time t_i for all $i \in [2, k]$, subject to the constraint that for every $i \in [k-1]$, there exists a range $\tau_i \in R(v_i v_{i+1})$ such that:

- $[t_{i+1} - \alpha_{\tau_i}(v_i v_{i+1}), t_{i+1}] \subseteq \tau_i$. In other words, the edge $v_i v_{i+1}$ must be present during the traversal of $v_i v_{i+1}$. Note that this makes τ_i unique.
- $t_i \leq t_{i+1} - \alpha_{\tau_i}(v_i v_{i+1})$. In other words, we must arrive at v_i early enough so that we can cross $v_i v_{i+1}$ to arrive at v_{i+1} at time t_{i+1}.

The *cost* of the journey is the sum of the traversal times $\sum_{i=1}^{k-1} \alpha_{\tau_i}(v_i v_{i+1})$. The *shortest journey* from v_1 to v_k is the journey with the minimum cost.

We use s to denote the specific source vertex from which we want to find shortest journeys to all other vertices. We use $[t_b, t_e)$ to denote the user requirement that the shortest journeys from s to other vertices must start at some time within $[t_b, t_e)$.

3. Continuous Dijkstra paradigm

We employ the continuous Dijkstra paradigm which has been used in finding shortest paths in geometric settings [2, 3, 4, 7, 11, 12]. The idea is to associate a semi-open time line $[t_b, \infty)$ with every vertex v other than s; the source s is associated with the interval $[t_b, t_e)$. For every vertex v other than s, we maintain a partition of $[t_b, \infty)$ into disjoint intervals denoted by $I(v)$; all intervals in $I(v)$ are closed at their start times and open at their end times; the rightmost interval in $I(v)$ is $[t, \infty)$ for some $t \geq t_b$. The sequence $I(s)$ is equal to $\{[t_b, t_e)\}$ at all times.

For every vertex v, every interval $\xi \in I(v)$ carries a cost $C_v(\xi)$, which is the cost of the shortest journey J we have found so far from s to v that arrives at time $\text{start}(\xi)$. Moreover, J is also the shortest journey discovered so far from s to v that arrives within the time interval ξ. The intervals $\xi \in I(v)$ for all $v \in V$ are stored in a priority queue Q in non-decreasing lexicographical order of $(C_v(\xi), \text{start}(\xi))$.

Initially, $I(s) = \{[t_b, t_e)\}$ and $C_s([t_b, t_e)) = 0$, and for all $v \in V \setminus \{s\}$, $I(v) = \{[t_b, \infty)\}$ and $C_v([t_b, \infty)) = \infty$.

As long as Q is non-empty, we remove the interval $\xi \in Q$ with the minimum $(C_v(\xi), \text{start}(\xi))$. It follows that the first interval removed from Q is $[t_b, t_e)$ for s. In general, assume that the interval ξ removed from Q belongs to $I(v)$. The continuous Dijkstra paradigm ensures that $C_v(\xi)$ is the cost of the shortest journey J from s to v that arrives at time $\text{start}(\xi)$, and J is also the shortest journey from s to v that arrives within the time interval ξ. Therefore, ξ and $C_v(\xi)$ will not be changed in the future.

The update after the removal of ξ from Q works as follows. We first mark ξ in $I(v)$; only intervals removed from Q will be marked. Take any outgoing edge vw of v and any range $\sigma \in R(vw)$. If $\text{end}(\sigma) - \text{start}(\sigma \cap \xi) < \alpha_\sigma(vw)$, we cannot extend J to visit w during σ and no update is triggered by σ. Otherwise, we can extend J by leaving from v at time $\text{start}(\sigma \cap \xi)$ and arriving at w at time $\text{start}(\sigma \cap \xi) + \alpha_\sigma(vw)$. In other words, we obtain a journey J' by appending $(w, \text{start}(\sigma \cap \xi) + \alpha_\sigma(vw))$ to J. The cost of J' is $C_v(\xi) + \alpha_\sigma(vw)$.

Find the interval $\zeta \in I(w)$ that contains the instance $\text{start}(\sigma \cap \xi) + \alpha_\sigma(vw)$. If $C_v(\xi) + \alpha_\sigma(vw) \geq C_w(\zeta)$, we do not update $I(w)$ because J' does not lead to a shorter journey to w. Suppose that $C_v(\xi) + \alpha_\sigma(vw) < C_w(\zeta)$. Note that ζ must be unmarked as there cannot be any shorter journey to w that arrives at any time within a marked interval. In this case, J' is the shortest journey from s to w discovered so far that arrives during $[\text{start}(\sigma \cap \xi) + \alpha_\sigma(vw), \text{start}(\zeta''))$, where ζ'' is the first interval in $I(w)$ after ζ such that $C_w(\zeta'') \leq C_v(\xi) + \alpha_\sigma(vw)$. Therefore, every interval that lies strictly between ζ and ζ'' must be unmarked. The updating of $I(w)$ proceeds as follows. We introduce a field $\text{pred}(\cdot)$ to every interval in $I(w)$ to record some information of the last stop in the shortest journey before w.

- Case 1: $\text{start}(\zeta) = \text{start}(\sigma \cap \xi) + \alpha_\sigma(vw)$.
  - Create an interval $\zeta_1 = [\text{start}(\zeta), \text{start}(\zeta''))$. Set $C_w(\zeta_1) = C_v(\xi) +$

$\alpha_\sigma(vw)$. Set $\text{pred}(\zeta_1) = (v, \xi)$.
- Delete the intervals between ζ and ζ'', including ζ but excluding ζ'', from Q and $I(w)$.
- Insert ζ_1 into Q and $I(w)$.

- Case 2: $\text{start}(\zeta) < \text{start}(\sigma \cap \xi) + \alpha_\sigma(vw)$.
  - Create an interval $\zeta_0 = [\text{start}(\zeta), \text{start}(\sigma \cap \xi) + \alpha_\sigma(vw))$. Set $C_w(\zeta_0)$ to be $C_w(\zeta)$. Set $\text{pred}(\zeta_0) = \text{pred}(\zeta)$.
  - Create an interval $\zeta_1 = [\text{start}(\sigma \cap \xi) + \alpha_\sigma(vw), \text{start}(\zeta''))$. Set $C_w(\zeta_1) = C_v(\xi) + \alpha_\sigma(vw)$. Set $\text{pred}(\zeta_1) = (v, \xi)$.
  - Delete the intervals between ζ and ζ'', including ζ but excluding ζ'', from Q and $I(w)$.
  - Insert ζ_0 and ζ_1 into both Q and $I(w)$.

Lemma 1 below proves some properties of $I(v)$ throughout the course of the algorithm. In particular, the proof of Lemma 1(iii) makes use of the fact that the weight and traversal time associated with a range are equal and positive.

Lemma 1. *The following properties hold throughout the course of the algorithm.*

(i) *For every $v \in V$ and every $\xi \in I(v)$, the journey that arrives at v at time $\text{start}(\xi)$ does not visit any vertex twice.*
(ii) *For every $v \in V$ and every pair $\zeta, \xi \in I(v)$, if ζ precedes ξ in $I(v)$, then $C_v(\zeta) > C_v(\xi)$.*
(iii) *For every $v \in V$, $|I(v)| \leq \Delta + 1$.*

Proof. Consider (i). Let J_ξ be the journey from s to v that arrives at time $\text{start}(\xi)$. Assume to the contrary that J_ξ visits a vertex u twice. This happens only if the algorithm created an interval ζ in $I(u)$ and then another interval ζ' in $I(u)$ later when constructing J_ξ. By the removal order of intervals from Q, we know that $C_u(\zeta) \leq C_u(\zeta')$. As J_ξ uses ζ before ζ', we have $\text{start}(\zeta) \leq \text{start}(\zeta')$. In fact, $\text{start}(\zeta) < \text{start}(\zeta')$ because the algorithm would not have created ζ' if $\text{start}(\zeta) = \text{start}(\zeta')$ given that $C_u(\zeta) \leq C_u(\zeta')$. By the working of the algorithm, after the creation of ζ, every interval in $I(u)$ that follows ζ has cost no more than $C_u(\zeta) \leq C_u(\zeta')$. But then the algorithm could not have created ζ' because no part of ζ or any interval in $I(u)$ that follows ζ can be subsumed by the journey from s to u that arrives at time $\text{start}(\zeta')$. This is a contradiction.

Consider (ii). Suppose that ζ precedes ξ in $I(v)$, i.e., $\text{start}(\zeta) < \text{start}(\xi)$. By the working of the algorithm, we know that $C_v(\zeta) \geq C_v(\xi)$. If $C_v(\zeta) = C_v(\xi)$, by the removal order of intervals from Q, ζ must be created before ξ. But then we can reason as in the previous paragraph that the algorithm would not have created ξ, a contradiction.

Consider (iii). Take any interval $\xi \in I(v)$. There is at most one interval in $I(v)$ with start time equal to t_b. Suppose that $\text{start}(\xi) > t_b$. The interval ξ must be created when we update $I(v)$ using a range σ on some incoming edge uv of v. If

$\text{start}(\xi) = \text{start}(\sigma) + \alpha_\sigma(uv)$, we set up an active charging link directed from ξ to σ. At most one interval in $I(v)$ can have an active charging link to σ. The other possibility is that $\text{start}(\xi) = \text{start}(\zeta) + \alpha_\sigma(uv)$ for some $\zeta \in I(u)$. In this case, we set up a passive charging link directed from ξ to ζ. For us to set up a passive charging link directed from ξ to ζ, we must have $[\text{start}(\zeta), \text{start}(\zeta) + \alpha_\sigma(uv)] \subseteq \sigma$, which makes the choice of σ unique. It follows that there is at most one passive charging link from the intervals in $I(v)$ to ζ.

We set up active and passive charging links for all $I(v)$'s in G as described above.

Fix a vertex $v \in V$. We count each interval $\xi \in I(v)$ as follows. Suppose that ξ has a passive charging link to another interval ξ'. We send a unit charge from ξ to ξ'. If ξ' has a passive charging link to another interval ξ'', we forward that unit charge to ξ'' and so on. Eventually, by (i), the unit charge must reach an interval ζ in some $I(w)$ such that ζ has an active charging link to some range $\sigma \in R(uw)$ for some incoming edge uw of w. In the extreme case, we can have $\xi = \zeta$ and $v = w$, meaning that ξ has an active charging link to σ, and there is no charge forwarding.

Fix some range $\sigma \in R(e)$ for some outgoing edge e of a vertex x. Consider the path(s) traversed by the charge(s) that go from interval(s) in $I(v)$ to σ. We claim that no two such paths can bifurcate at any vertex y. Assume to the contrary that there are two such paths that do. Let J_1 and J_2 be the two corresponding journeys that start from s, cross the edge e at time $\text{start}(\sigma)$, and eventually arrive at y. Both J_1 and J_2 use the same interval $\xi \in I(x)$ such that ξ contains $\text{start}(\sigma)$. It implies that J_1 and J_2 share a common prefix from s to x which defines the interval ξ. Therefore, it costs the same for J_1 and J_2 to go from s to x. Let C denote this common cost. The arrival times of J_1 and J_2 at y are equal to $\text{start}(\sigma) + \gamma_1$ and $\text{start}(\sigma) + \gamma_2$, respectively, where γ_i is the total traversal time taken by J_i from x to y. The cost of the portion of J_i from s to y is equal to $C + \gamma_i$. If $\gamma_1 < \gamma_2$, J_1 reaches y at an earlier time than J_2, but then our algorithm would not create an interval in $I(y)$ when J_2 reaches y because J_2 does not lead to a shorter journey to y. This is a contradiction. We get a contradiction in a similar way if $\gamma_2 < \gamma_1$. If $\gamma_1 = \gamma_2$, the algorithm would let J_1 or J_2 reach y first, and then our algorithm would not allow the other journey to create an interval in $I(y)$ later, a contradiction again. This proves our claim.

By our claim, there is at most one path from a single interval in $I(v)$ to σ. We conclude that σ receives at most one unit of charge. The total number of charges generated by the intervals in $I(v)$ is thus at most Δ. Hence, $|I(v)| \leq \Delta + 1$, including the possibility that there is an interval in $I(v)$ with start time equal to t_b. □

Corollary 2. *The algorithm inserts at most $n\Delta + n$ intervals into Q.*

Proof. After removing an interval ξ from Q, where $\xi \in I(v)$ for some $v \in V$, $C_v(\xi)$ is already the minimum possible for $\xi \in I(v)$, so no future update of $I(v)$ will affect the existence of ξ as an interval in $I(v)$. Then, by Lemma 1(iii), at most $n\Delta + n$ intervals can be removed from Q, meaning that at most $n\Delta + n$ intervals can be

inserted into Q. □

4. Data structures

Corollary 2 tells us that there are $O(n\Delta)$ intervals to be inserted and deleted from Q. The running time of these insertions and deletions is $O(n\Delta \log(n\Delta)) = O(n\Delta \log \Delta)$ as Δ is assumed to be $\Omega(n)$.

Suppose that we delete an interval $\xi \in I(v)$ from Q. If we update $I(w)$ for every outgoing edge vw of v in a straightforward manner as described in Section 3, it will take $O\big(|R(vw)|\Delta \log(n\Delta)\big) = O\big(|R(vw)|\Delta \log \Delta\big)$ time for every edge vw. By Lemma 1(iii), the total time spent in updating $I(w)$ for every edge vw throughout the algorithm is $O\big(|I(v)| \sum_{vw\in E} |R(vw)|\Delta \log \Delta\big) = O\big(\Delta^2 \log \Delta \sum_{vw\in E} |R(vw)|\big)$.

The total running time is thus

$$O\Big(n\Delta \log \Delta + \Delta^2 \log \Delta \sum_{v\in V} \sum_{vw\in E} |R(vw)|\Big) = O\big(n\Delta \log \Delta + \Delta^3 \log \Delta\big) = O(\Delta^3 \log \Delta).$$

In this section, we reduce the update time of every $I(w)$ to $O\big(|R(vw)| \log \Delta\big)$. The total running time will be improved to $O(\Delta^2 \log \Delta)$.

When processing an range in $R(vw)$, the bottleneck are the potential deletions of $O(\Delta)$ intervals from Q and $I(w)$ This is too costly. Since we are only interested in the interval in Q with the minimum cost, it suffices to store the interval in $I(w)$ with the minimum cost in Q. By Lemma 1(ii), there is exactly one such interval. This strategy immediately eliminates the need to delete a lot of intervals from Q.

We still have to deal with the deletions of $O(\Delta)$ intervals from $I(w)$. Since these intervals are consecutive in $I(w)$, we can delete them efficiently in a batch with the right data structure. We store the intervals in $I(w)$ in a red-black tree T_w ordered by their costs. By Lemma 1(ii), the order of the intervals in decreasing costs coincides with their left-to-right order. The red-black tree T_w supports insertion, deletion, join, and split in $O(\log |I(w)|) = O(\log \Delta)$ time [13]. Insertion and deletion are standard. We use the definitions of join and split from [13] adapted to our context:

- join(T_1, ζ, T_2): Let T_1 and T_2 be two trees that store two subsets of $I(w)$, each consisting of contiguous intervals, and the intervals in T_1 are to the left of those in T_2. Let ζ be an interval that lies between those in T_1 and T_2. This function returns a tree that contains the intervals in T_1, ζ, and the intervals in T_2 in this order.
- split(ζ, T): Let T be a tree that store a subset of contiguous intervals in $I(w)$ that includes ζ. This function returns a pair of trees (T_1, T_2); T_1 consists of all intervals in T to the left of ζ; T_2 consists of all intervals in T to the right of ζ.

We revisit the update process after removing an interval ξ from Q. Assume that ξ comes from T_v. We mark ξ in T_v, find the rightmost unmarked interval ξ' in T_v,

and insert ξ' into Q. To allow a fast search for the rightmost unmarked interval in T_v, we stored a bit at each node of T_v which is 1 if and only if there is an unmarked interval in the subtree rooted at that node. Then, we use this bit to go down T_v to find ξ' in $O(\log \Delta)$ time. The insertion of ξ' into Q also takes $O(\log \Delta)$ time.

Next, we go over every outgoing edge vw of v and every range $\sigma \in R(vw)$. Suppose that $\text{end}(\sigma) - \text{start}(\sigma \cap \xi) \geq \alpha_\sigma(vw)$. We search in T_w in $O(\log \Delta)$ time to find the interval ζ that contains the instance $\text{start}(\sigma \cap \xi) + \alpha_\sigma(vw)$. If $C_v(\xi) + \alpha_\sigma(vw) \geq C_w(\zeta)$, we do not update T_w. Suppose that $C_v(\xi) + \alpha_\sigma(vw) < C_w(\zeta)$. In this case, we call $\text{split}(\zeta, T_w)$ and let (T_1, T_2) be the ordered pair of trees returned. We search for the leftmost interval $\zeta'' \in T_2$ such that $C_w(\zeta'') \leq C_v(\xi) + \alpha_\sigma(vw)$. Then, we call $\text{split}(\zeta'', T_2)$ and let (T_3, T_4) be the ordered pair of trees returned. This takes $O(\log \Delta)$ time. The creation of the new T_w proceeds as follows.

- Case 1: $\text{start}(\zeta) = \text{start}(\sigma \cap \xi) + \alpha_\sigma(vw)$.
  - Create an interval $\zeta_1 = [\text{start}(\zeta), \text{start}(\zeta''))$. Set $C_w(\zeta_1) = C_v(\xi) + \alpha_\sigma(vw)$. Set $\text{pred}(\zeta_1) = (v, \xi)$.
  - Insert ζ'' into T_4 to form a new tree T_5.
  - Set the new T_w to be $\text{join}(T_1, \zeta_1, T_5)$.
- Case 2: $\text{start}(\zeta) < \text{start}(\sigma \cap \xi) + \alpha_\sigma(vw)$.
  - Create an interval $\zeta_0 = [\text{start}(\zeta), \text{start}(\sigma \cap \xi) + \alpha_\sigma(vw))$. Set $C_w(\zeta_0)$ to be $C_w(\zeta)$. Set $\text{pred}(\zeta_0) = \text{pred}(\zeta)$.
  - Create an interval $\zeta_1 = [\text{start}(\sigma \cap \xi) + \alpha_\sigma(vw), \text{start}(\zeta''))$. Set $C_w(\zeta_1) = C_v(\xi) + \alpha_\sigma(vw)$. Set $\text{pred}(\zeta_1) = (v, \xi)$.
  - Insert ζ_1 and ζ'' into T_4 to form a new tree T_5.
  - Set the new T_w to be $\text{join}(T_1, \zeta_0, T_5)$.

Cases 1 and 2 run in $O(\log \Delta)$ time. Finally, if the interval in Q that comes from T_w is not the rightmost unmarked interval in the new T_w, delete that interval from Q and insert the rightmost unmarked interval in the new T_w into Q. This takes $O(\log \Delta)$ time.

In summary, for every range in $R(vw)$, we can update $I(w)$ in $O(\log \Delta)$ time; therefore, it takes $O(|R(vw)| \log \Delta)$ time to handle each outgoing edge vw of v. Consequently, the total running time is improved to $O(\Delta^2 \log \Delta)$.

The final red-black trees T_v for every $v \in V \setminus \{s\}$ can be used to answer queries. Needless to say, any query for journeys that arrive at s has zero cost. For any $v \in V \setminus \{s\}$ and any given time t, we search T_v to find the interval ξ that contains t, and we return $C_v(\xi)$ as the desired journey cost. This takes $O(\log \Delta)$ time. If we want to retrieve the journey from s that arrives at v no later than time t, we use $\text{pred}(\xi) = (u, \xi')$ to get to ξ' in $I(u)$. We can thus continue this way to retrace the journey from v to s in linear time.

Theorem 3. *Let $G = (V, E)$ be a directed compact temporal graph in which the weight and traversal time of a range are equal and positive. Let $n = |V|$ and let*

$\Delta = \sum_{e \in E} |R(e)|$. *Assume that* $\Delta = \Omega(n)$. *For any fixed source* $s \in V$ *and any interval* $[t_b, t_e)$, *we can construct in* $O(\Delta^2 \log \Delta)$ *time a data structure of size* $O(n\Delta)$ *such that for any vertex* v *and any time* t, *we can report in* $O(\log \Delta)$ *time the cost of the shortest journey that starts from* s *within the time interval* $[t_b, t_e)$ *and arrives at* v *no later than time* t. *The journey achieving the reported cost can be produced in time linear in its complexity.*

5. Conclusion

We present an $O(\Delta^2 \log \Delta)$-time algorithm for finding single-source shortest journeys in a directed compact temporal graph in which the weight and traversal time of a range are equal and positive. The algorithm outputs a data structure that answers shortest journey queries in $O(\log \Delta)$ time. A research direction is to allow the weight of a range to be different from the traversal time. On the other hand, it seems challenging to allow the weights to be completely arbitrary. Another possible research direction is to apply the continuous Dijkstra paradigm to the single-source fastest journeys problem.

References

[1] A. Casteigts, P. Flocchini, B. Mans, and N. Santoro. Measuring temporal lags in delay-tolerant networks. *IEEE Transactions on Computers*, 63:397–410, 2014.
[2] J. Chen and Y. Han. Shortest paths on a polyhedron, part I: computing shortest paths. *International Journal of Computational Geometry and Applications*, 6:127–144, 1996.
[3] S.-W. Cheng and J. Jin. Approximate shortest descending paths. *SIAM Journal on Computing*, 43:410–428, 2014.
[4] S.-W. Cheng and J. Jin. Shortest paths on polyhedral surfaces and terrains. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, pages 373–382, 2014.
[5] M. H. Eiza and Q. Ni. An evolving graph-based reliable routing scheme for vanets. *IEEE Transactions on Vehicular Technology*, 63:1493–1504, 2013.
[6] T. Erlebach and J.T. Spooner. Faster exploration of degree-bounded temporal graphs. In *Proceedings of the 43rd International Symposium on Mathematical Foundations of Computer Science*, pages 36:1–36:13, 2018.
[7] J. Hershberger and S. Suri. An optimal algorithm for Euclidean shortest paths in the plane. *SIAM Journal on Computing*, 28:2215–2256, 1999.
[8] P. Holme and J. Saramäki. Temporal networks. *Physics Reports*, 519:97–125, 2012.
[9] D. Kempe, J. Kleinberg, and A. Kumar. Connectivity and inference problems for temporal networks. *Journal of Computer and System Sciences*, 64:820–842, 2002.
[10] G.B. Mertzios, O. Michail, and P.G. Spirakis. Temporal network optimization subject to connectivity constraints. *Algorithmica*, 81:1416–1449, 2019.
[11] J.S.B. Mitchell, D.M. Mount, and C.H. Papadimitriou. The discrete geodesic problem. *SIAM Journal on Computing*, 16:647–668, 1987.
[12] J.S.B. Mitchell and C.H. Papadimitriou. The weighted region problem: finding shortest paths through a weighted planar subdivision. *Journal of the ACM*, 38:18–73, 1991.

[13] R.E. Tarjan. *Data Structures and Network Algorithms*. Society for Industrial and Applied Mathematics, 1987.

[14] H. Wu, J. Cheng, Y. Ke, S. Huang, Y. Huang, and H. Wu. Efficient algorithms for temporal path computation. *IEEE Transactions on Knowledge and Data Engineering*, 28:2927–2942, 2016.

[15] B. Bui Xuan, A. Ferreira, and A. Jarry. Computing shortest, fastest, and foremost journeys in dynamic networks. *International Journal of Foundations of Computer Science*, 14:167–181, 2003.