Computing Temporal Reachability Under Waiting-Time Constraints in Linear Time

Filippo Brunelli
Université Paris Cité, Inria, CNRS, IRIF, Paris, France

Laurent Viennot
Université Paris Cité, Inria, CNRS, IRIF, Paris, France

Abstract

This paper proposes a simple algorithm for computing single-source reachability in a temporal graph under waiting-time constraints, that is when waiting at each node is bounded by some time constraints. Given a space-time representation of a temporal graph, and a source node, the algorithm computes in linear-time which nodes and temporal edges are reachable through a constrained temporal walk from the source.

2012 ACM Subject Classification Theory of computation → Dynamic graph algorithms; Mathematics of computing → Graph algorithms

Keywords and phrases temporal reachability, temporal graph, temporal path, temporal walk, waiting-time constraints, restless temporal walk, linear time

Digital Object Identifier 10.4230/LIPIcs.SAND.2023.4

Related Version *Extended Version*: https://hal.inria.fr/hal-03864725v2

Funding This work was supported by the French National Research Agency (ANR) through project Tempogral with reference number ANR-22-CE48-0001.

1 Introduction

Reachability, that is connectivity through a path, is a fundamental notion in graphs. There exist simple and elegant algorithms that determine efficiently which nodes or edges can be reached from a given source node, and finding optimal paths realizing such reachability. On the other hand, temporal graphs, where edges evolve over time, offer a richer variety of temporal connectivity, especially when considering waiting constraints as we discuss below. By focusing on reachability without concern for any optimization criterion, we aim at designing a simple algorithm under waiting constraints.

Temporal graphs

Temporal graphs arose with the need to better model contexts where the appearance of interactions or connections depends on time, such as epidemic propagation or transport networks. Starting with the work on time-dependent networks [8] and the telephone problem [5], the discrete time version of temporal graphs we consider here was already investigated in [2, 16, 18] and introduced later in various contexts ranging from social interactions to mobile networks and distributed computing (see e.g. [6, 14, 13]). This classical point-availability model of temporal graph is the following. The availability of an edge (u, v) at time τ is modeled by a temporal edge $e = (u, v, \tau, \lambda)$. It represents the possibility to traverse the edge from u at time exactly τ with arrival in v at time $\tau + \lambda$. We refer to τ and $\tau + \lambda$ as the departure time and arrival time of e respectively, while $\lambda > 0$ is called the travel time of e. Notice that we consider travel time to be strictly positive, which is a natural assumption when it comes down to application like, for example, transport networks. A temporal walk can


© Filippo Brunelli and Laurent Viennot;
licensed under Creative Commons License CC-BY 4.0

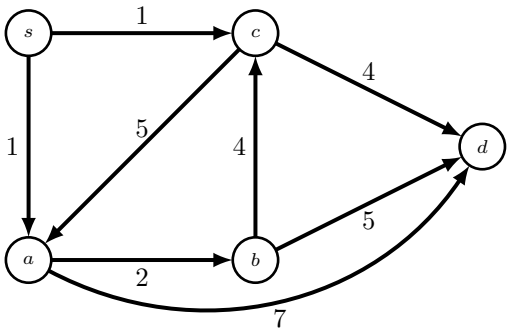

Figure 1 A temporal graph with waiting constraints. Each temporal edge in the picture is labeled with its departure time and has travel time one, each node has minimum waiting-time $\alpha = 0$ and maximum waiting-time $\beta = 1$. The only temporal edge entering d and reachable from s is $(a, d, 7, 1)$ through the temporal walk $(s, a, 1, 1), (a, b, 2, 1), (b, c, 4, 1), (c, a, 5, 1), (a, d, 7, 1)$. Indeed, following any of the two other edges $(b, d, 5, 1)$ and $(c, d, 4, 1)$ would require to wait $2 > \beta$ units of time either at b (after edge $(a, b, 2, 1)$) or c (after edge $(s, c, 1, 1)$).

then be defined as a sequence of temporal edges such that each temporal edge arrives at the departing node of the next one, and the arrival time of each temporal edge is less or equal to the departure time of the next one. The inequality means that it is possible to wait at the node in-between two consecutive temporal edges. In particular, the time elapsed between the arrival time of an edge and the departure of the next one represents the amount of time spent waiting at the node. We distinguish such a walk from a temporal path, which is a temporal walk visiting at most once any node.

Without waiting constraints, that is when waiting at a node is unrestricted, numerous works have investigated single-source temporal path computation with a primary focus on earliest arrival. After several works inspired by Dijkstra algorithm (see e.g. [2, 3, 16, 18]), a simple and elegant linear-time algorithm for earliest arrival time was first claimed in [20] with a similar algorithm as [10] through a single scan of temporal edges ordered by non-decreasing departure time. Assuming strictly positive travel times is important here as it ensures that the temporal edges of any temporal path appear in order during this scan. These algorithms indeed focus their target on temporal paths rather than walks, since unrestricted waiting allows to transform any walk into a path by waiting at nodes instead of performing any loops. In this setting they allow to determine which nodes or edges are reachable. However, we consider the following more general model.

Waiting constraints

We consider temporal graphs subject to waiting constraints. In such graphs, during a temporal walk, it is not possible to wait at a node less than α time or more than β time, before moving to another node. Such constraints can be used to model, for example, preferences of a user in a public transport network, or to take into account incubation time and recovery time of a disease in a temporal network of contacts.

We focus on the following rechability problem. We say that a temporal edge $e = (u, v, \tau, \lambda)$ is reachable from a node s if there exists a temporal walk from s ending with e and respecting waiting constraints. The *reachability problem* consists in identifying all the temporal edges that are reachable from a given source node s (see Figure 1 for an example). Note that we can easily compute which nodes are reachable from this. Moreover, this problem generalizes the

single-source earliest arrival time problem: indeed, given the set of the s-reachable edges, a linear scan allows to identify for each node v the s-reachable edge with head v that has lowest arrival time, and which corresponds to the earliest arrival time at v. Computing the number of nodes reachable from a given source can be interesting when measuring connectivity properties of a temporal network [9]. Reachability of edges additionally provides information about all times at which it is possible to reach such nodes. This is related to the profile problem [11], which consists in computing a function that for each possible departure time from the source return the earliest arrival time towards the destinations.

Waiting constraints significantly modify temporal connectivity. Most strikingly, it has been proved that the computation of temporal paths in this setting becomes NP-hard [7]. While unrestricted waiting makes reachability through temporal paths or walks equivalent, this result moves the interest to the sole case of temporal walks when dealing with bounded waiting. It is thus necessary to design algorithms following a different approach.

In a recent break-through, [1] proposes an algorithm computing single-source optimal temporal walks under waiting-time constraints in $O(M \log M)$ time, where M is the total number of temporal edges in the graph. It computes walks that optimize a linear combination of the most classical criteria, and can thus solve the reachability problem as well. The algorithm is quite involved. It relies on first transforming the temporal graph so as to zero all travel times and then performing a Dijkstra computation for each time instant when a temporal edge departs or arrives. More precisely, it first builds an equivalent temporal graph where all edges have zero travel time. This is done by adding a dummy node with appropriate waiting restrictions for each temporal edge. Note that this can considerably increase the number of nodes. Then, it scans time instants when temporal edges occur in increasing order. For each time instant t, a static directed graph is constructed and a Dijkstra computation allows to update the reachability of nodes with temporal edges up to time t. It feels natural to investigate whether in the context of bounded waiting it is possible to develop an easier and more efficient method to solve the simpler reachability problem.

Contribution

We develop an algorithm to solve the reachability problem in temporal graphs subject to waiting constraints. The main strength of the algorithm is its simplicity, which comes with no downplay in efficiency, since it runs in linear time. It thus improves by a factor $\log M$ the state of the art in the setting of positive travel times. The algorithm performs a linear scan of the list of temporal edges, in a spirit similar to [10]. In this case, however, the algorithm requires in input not one, but two, sorted lists, one containing the temporal edges sorted by departure time and the other by arrival time. Exploiting these two lists is a new technique for handling waiting constraints that could be useful for computing optimum temporal walks or other temporal connectivity problems. Interestingly, this representation of the temporal graph through two lists is closely related to the more classical "space-time" (or "time-expanded") graph [17, 18, 19, 15, 14] where each node is split into node events, one for each time where a temporal edge arrives to it or departs from it, and each temporal edge is turned into an arc between two node events. If the input is given as a list of temporal edges, the two appropriate lists can be obtained in $O(M \log M)$ time. However, our algorithm runs in linear time when given a space-time representation as input since two appropriate lists can easily be obtained from a topological ordering of the corresponding static directed graph. The positive travel time assumption can be loosened to a more general acyclic setting as described in a follow-up paper [4].

The paper is organized as follows. In Section 2 we define the main notions. In Section 3 we present our main algorithmic result. Finally, in Section 4 we show how to adapt our algorithm in order to take as input a space-time representation of a temporal graph.

2 Preliminary definitions

A *temporal graph* is a tuple $G = (V, E, \alpha, \beta)$, where V is the set of *nodes*, E is the set of temporal edges and $\alpha, \beta \in [0, +\infty]^V$ are minimum and maximum waiting-times at each node. A *temporal edge* e is a quadruple (u, v, τ, λ), where $u \in V$ is the *tail* of e, $v \in V$ is the *head* of e, $\tau \in \mathbb{R}$ is the *departure time* of e, and $\lambda \in \mathbb{R}_{>0}$ is the *travel time* of e. We also define the *arrival time* of e as $\tau + \lambda$, and we let $dep(e) = \tau$ and $arr(e) = \tau + \lambda$ denote the departure time and arrival time of e respectively. For the sake of brevity, we often say edge instead of temporal edge. We let $n = |V|$ and $M = |E|$ denote the number of nodes and edges respectively.

Given a temporal graph $G = (V, E, \alpha, \beta)$ a *walk* Q from u to v, or a *uv-walk* for short, is a sequence of temporal edges $\langle e_1 = (u_1, v_1, \tau_1, \lambda_1), \ldots, e_k = (u_k, v_k, \tau_k, \lambda_k)\rangle \subseteq E^k$ such that $u = u_1$, $v_k = v$, and, for each i with $1 < i \leq k$, $u_i = v_{i-1}$ and $a_{i-1} + \alpha_{u_i} \leq \tau_i \leq a_{i-1} + \beta_{u_i}$ where $a_{i-1} = \tau_{i-1} + \lambda_{i-1}$ is the arrival time of e_{i-1}. Note that the waiting time $\tau_i - a_{i-1}$ at node u_i is constrained to be in the interval $[\alpha_{u_i}, \beta_{u_i}]$. Note that since travel times are positive, such a walk is *strict* in the sense that $\tau_{i-1} < \tau_i$ for $1 < i \leq k$ as the constraint $a_{i-1} + \alpha_{u_i} \leq \tau_i$ implies $\tau_i \geq a_{i-1} = \tau_{i-1} + \lambda_{i-1} > \tau_{i-1}$ for $\lambda_{i-1} > 0$. The *departing time* $dep(Q)$ of Q is defined as τ_1, while the *arrival time* $arr(Q)$ of Q is defined as $\tau_k + \lambda_k$. We say that a temporal edge $e = (x, y, \tau, \lambda)$ *extends* Q when $x = v_k$ and $arr(Q) + \alpha_x \leq \tau \leq arr(Q) + \beta_x$. When e extends Q, we can indeed define the walk $Q.e = \langle e_1, \ldots, e_k, e\rangle$ from u to y. Moreover, we also say that e extends e_k as it indeed extends any walk Q having e_k as last edge. We also say that an edge e is an *s-reachable edge* whenever there exists an *sv*-walk ending with edge e. Let us now introduce some orderings of temporal edges with respect to certain temporal criteria. Given an ordering of the temporal edges E^{ord} we use $e <_{E^{ord}} f$ to denote that e appears before f in E^{ord}. We say that an ordering E^{ord} of all the temporal edges is *departure sorted* if the edges are ordered by non-decreasing departure time in E^{ord}, that is we have $e <_{E^{ord}} f$ whenever $e, f \in E$ satisfy $dep(e) < dep(f)$. Similarly, we say that an ordering E^{ord} of all the temporal edges is *arrival sorted* if the edges are ordered by non-decreasing arrival time in E^{ord}, that is we have $e <_{E^{ord}} f$ whenever $e, f \in E$ satisfy $arr(e) < arr(f)$.

Finally, we define the *doubly-sorted representation* of a temporal graph (V, E, α, β) as a data-structure with two lists (E^{dep}, E^{arr}), containing $|E|$ quadruples each, representing all temporal edges in E, where E^{dep} is a list sorted by non-decreasing departure time and E^{arr} is a list sorted by non-decreasing arrival time. Moreover, we assume that we have implicit pointers between the two lists, that link each quadruple of one list to the quadruple representing the same temporal edge in the other list.

Without loss of generality, we can restrict our attention to nodes appearing as head or tail of at least one temporal edge and we thus assume $|V| = O(|E|)$. An algorithm is said to be linear in time and space when it runs in $O(|E|)$ time and uses $O(|E|)$ space. Given a doubly sorted representation (E^{dep}, E^{arr}), we also assume that we are given for each node v the list E^{dep}_v of pointers to temporal edges with tail v ordered by non-decreasing departure time, as it can be computed in linear time and space from E^{dep} through bucket sorting. We assume that each list E^{arr}, E^{dep}, or E^{dep}_v is stored in an array T such that each element $T[i]$ can be accessed directly through its index $i \in [1, |T|]$ in constant time. Given two indexes $i \leq j$, we also let $T[i : j]$ denote the sub-array of elements of T with index in $[i, j]$.

3 A linear-time Algorithm to compute reachability

In this section we will provide our main result: an algorithm that solves in linear time and space the reachability problem, which is defined as follows.

> Singles-Source Reachability Problem. Given a temporal graph with waiting constraints $G = (V, E, \alpha, \beta)$ and a source node s, compute the set of all temporal edges that are s-reachable.

In the following we will assume to be given a doubly-sorted representation (E^{dep}, E^{arr}) of the temporal graph. We design an algorithm which mainly consists in scanning linearly edges in E^{arr} while updating the set A_v of s-reachable edges terminating sv-walks in the temporal graph resulting from the edges read so far. To help identifying edges that will appear in such walks in next iterations, we also mark edges that extend these walks.

We now describe more precisely how edges are scanned and marked as formalized in Algorithm 1. We first build the lists E^{dep}_v of temporal edges with tail v by bucket sorting E^{dep} at Line 1. We then identify the s-reachable edges as follows. We linearly scan E^{arr}. In the temporal graph resulting from the temporal edges read up to edge $e = (u, v, \tau, \lambda) \in E^{arr}$, the only walks from s that have not been considered yet must contain e, and must have it as last edge as E^{arr} is sorted by non-decreasing arrival time. If its tail u is s, or if e is marked, then we know that there exists a walk from s to its head v. In that case, we add edge e to A_v at Line 9, and we then mark edges that extend e, that is edges in E^{dep}_v with departure time in $[a + \alpha_v, a + \beta_v]$, since the arrival time of e is $a = \tau + \lambda$. These edges appear consecutively in E^{dep}_v which is processed linearly as walks from s to v are identified. This process is done in Lines 10-14 in Algorithm 1, starting from the index p_v of the last processed edge in E^{dep}_v, and such edges f that extend e are marked at Line 13 before updating p_v. Moreover, we use classical parent pointers to be able to compute an sv-walk for each s-reachable edge with head v. Each parent pointer $P[f]$ of an edge f is initially set to a null value $\perp$ at Line 6. Whenever we mark edge f, that extend the currently scanned edge e, we set the parent pointer of f to e. If f is an s-reachable edge at v, we can then get an sv-walk by following the parent pointer $P[f], P[P[f]], \ldots$.

▶ **Theorem 1.** *Given a doubly-sorted representation of a temporal graph with waiting constraints $G = (V, E, \alpha, \beta)$ and a source node $s \in V$, Algorithm 1 computes all s-reachable temporal edges in linear time and space.*

Proof.

Correctness. Let us denote by $G_k = (V, E^{arr}[1 : k], \alpha, \beta)$ the temporal graph induced by the first k temporal edges in E^{arr}. We will prove, by induction on k, the following two invariants:

(I^1_k) For every node v, A_v contains all s-reachable edges with head v in G_k.
(I^2_k) The marked edges are all the edges in E that extend a walk from s in G_k.

The correctness of the algorithm will follow from invariant (I^1_k) for $k = |E|$. The invariants are satisfied for $k = 0$ since there are no edges in G_0 while the sets $(A_v)_{v \in V}$ of s-reachable edges are initially empty and no edge is initially marked.

Now suppose that the two invariants hold for $k - 1$, with $k \geq 1$, and let us prove that they still hold for k after scanning the kth edge $e_k = (u, v, \tau, \lambda)$ in E^{arr}. To prove (I^1_k) and (I^2_k), we first show that the condition of the if statement at Line 8 is met when e_k is an s-reachable edge in G_k. It is obviously the case when $u = s$ as $\langle e_k \rangle$ is in G_k, or when e_k was

Algorithm 1 Computing, for each node v, the set A_v of all s-reachable edges with head v.

Input: A doubly-sorted representation (E^{arr}, E^{dep}) of a temporal graph G with waiting constraints (α, β), and a source node $s \in V$.
Output: The sets $(A_v)_{v \in V}$ of s-reachable edges at each node v sorted by non-decreasing arrival time.

1 For each node v, generate the list E_v^{dep} by bucket sorting E^{dep}.
2 **For** *each node* v **do**
3 Set $A_v := \emptyset$. `/* Set of s-reachable edges (as a sorted list). */`
4 Set $p_v := 0$. `/* Index of the last processed edge in` E_v^{dep}`. */`
5 Set all the edges in E^{arr} as unmarked.
6 Set $P[e] := \perp$ for each edge $e \in E^{arr}$.`/* Parent of` e`, initially null. */`
7 **For** *each edge* $e = (u, v, \tau, \lambda)$ *in* E^{arr} **do**
8 **If** $u = s$ *or* e *is marked* **then**
  `/*` e `is` s`-reachable. */`
9 $A_v := A_v \cup \{e\}$
10 Let $a = \tau + \lambda$ be the arrival time of e.
  `/* Process further edges from` v `with dep. time` $\leq a + \beta_v$`: */`
11 Let $l > p_v$ be the first index of an edge $(v, w, \tau', \lambda') \in E_v^{dep}$ such that $\tau' \geq a + \alpha_v$ (set $l := |E_v^{dep}| + 1$ if no such index exists).
12 Let $r \geq l$ be the last index of an edge $(v, w, \tau', \lambda') \in E_v^{dep}$ such that $\tau' \leq a + \beta_v$ (set $r := l - 1$ if no such index exists).
  `/* Mark unmarked edges with dep. time in` $[a + \alpha_v, a + \beta_v]$`: */`
13 **If** $l \leq r$ **then** mark each edge $f \in E_v^{dep}[l : r]$ and set $P[f] := e$.
14 Set $p_v := r$.
15 **Return** *the sets* $(A_v)_{v \in V}$.

previously marked, as Invariant (I_{k-1}^2) then implies that it extends a walk Q from s in G_{k-1} and that $Q.e_k$ is a walk in G_k. The converse also holds: if e_k is an edge of a walk Q from s in G_k, then either it is the first edge and we have $u = s$ or the sequence Q' of edges before e_k in Q is a walk in G_{k-1} and (I_{k-1}^2) implies that it is marked.

Note that when e_k appears in a walk Q of G_k, it must be the last edge of Q as E^{arr} is sorted by non-decreasing arrival time and edges have positive travel time. This allows to prove (I_k^1): as we assume (I_{k-1}^1), we just have to consider walks from s that are in G_k but not in G_{k-1}, that is those containing e_k. Since all these walks have e_k as last edge, and e_k is the only edge added to A_v when such walks exist, we can conclude that (I_k^1) holds.

Similarly, to prove (I_k^2) when (I_{k-1}^2) holds, we just have to consider the edges extending a walk Q from s which is in G_k but not in G_{k-1}. As discussed above, when such a walk Q exists, e_k is its last edge and the condition of the if statement Line 8 holds. Edges extending such a walk Q are thus those extending e_k, that is all edges $f \in E_v^{dep}$ such that $a + \alpha_v \leq dep(f) \leq a + \beta_v$. Note that the ordering of E_v^{dep} implies that these edges are consecutive in E_v^{dep}. If no such edges exist, let l' and r' designate the first and last indexes respectively where they are placed in E_v^{dep}. To prove (I_k^2), it thus suffices to prove that all edges in $E_v^{dep}[l' : r']$ are marked after scanning e_k and that only edges in $E_v^{dep}[l' : r']$ are marked during the iteration for e_k (if no such edges exist we prove that we mark no edges). Consider the values l and r computed at Lines 11 and 12 respectively. If no edge f extends e_k, then we get $r = l - 1$ and no edge is marked. Now, we assume that such edges exist and

that l' and r' are well defined. First assume $l \leq r$ and thus that l was not set to $|E_v^{dep}| + 1$. The choice of l, r then imply $a + \alpha_v \leq dep(E_v^{dep}[l])$ and $dep(E_v^{dep}[r]) \leq a + \beta_v$. We thus have $l' \leq l \leq r \leq r'$ and all marked edges at Line 13 are in $E_v^{dep}[l' : r']$. Moreover, the choice of r indeed then implies $r = r'$. We still need to prove that edges in $E_v^{dep}[l' : l-1]$ have already been marked. Otherwise, when $r = l - 1$, no edge is marked. This occurs when $p_v \geq r'$ and we then have $l = p_v + 1$. In both cases, it remains to prove that all edges $f \in E_v^{dep}[l' : \min\{l-1, r'\}]$ have already been marked. This interval is non empty when $l' \leq l - 1$ and thus $p_v = l - 1$ by the choice of l. We thus have $p_v \geq \min\{l-1, r'\}$. Let i be the index of f in E_v^{dep} and consider the iteration $j < k$ when p_v was updated from a value smaller than i to a value $r'' \geq i$ where l'' and r'' denote the indexes computed for variables l and r respectively during the j-th iteration for edge $e_j \in E^{arr}$.

Since E^{arr} is sorted by non-decreasing arrival time, the arrival time a' of e_j satisfies $a' \leq a$ and we thus have $dep(f) \geq a + \alpha_v \geq a' + \alpha_v$. The choice of index l at Line 11 in that iteration thus guarantees that the index l'' must satisfy $l'' \leq i$. We thus have $l'' \leq i \leq r''$ and f was marked at Line 13 during the jth iteration. This completes the proof of (I_k^2).

We finally prove that the parent pointers allow us to compute for each s-reachable edge $f = (u, v, \tau, \lambda)$ with head v an sv-walk ending with f. If $f \in A_v$ and it is not marked, then $P[f] = \perp$, and we must have $u = s$ as f was added to A_v. In this case, $\langle f \rangle$ is an sv-walk itself. Now consider the case $f \in A_v$ and f is marked. Consider the iteration k where f was marked. By (I_{k-1}^2) and (I_k^2), f extends a walk from s ending with e_k, where e_k is the edge scanned at iteration k, and $P[f]$ was then set to e_k. This guarantees by a simple induction that, if $P[f] \neq \perp$, by following the parent pointers in classical manner, namely $P[f], P[P[f]], \ldots,$ until $\perp$ is found, it is possible to obtain a walk terminating with edge f.

Complexity analysis. The preprocessing of E^{dep} and the initialization from Line 1 to Line 6 clearly take linear time. The main for loop scans each temporal edge $e = (u, v, \tau, \lambda)$ in E^{arr} exactly once. For each iteration there are three operations that may require non-constant time: the computation of l and r at Lines 11 and 12, and marking edges in $E_v^{dep}[l, r]$ at Line 13. They all take $O(r - p_v)$ time as l and r can be found by scanning edges in E_v^{dep} from $p_v + 1$. Thanks to the update of the index p_v to r, each edge in E_v^{dep} is processed at most once for a total amortized cost of $O(|E_v^{dep}|)$. Overall, this leads to a time complexity of $O(|E| + \sum_{v \in V} |E_v^{dep}|) = O(|E|)$. Algorithm 1 thus runs in linear time. Finally, let us notice that for all nodes v, the set A_v has size bounded by the number of temporal edges with head v. We thus have $\sum_{v \in V} |A_v| \leq |E|$, and the space complexity of Algorithm 1 is linear. ◀

4 Taking a space-time representation as input

Let us first recall the definition of the "space-time" representation [17]. It consists in a transformation of a temporal graph into a static graph by introducing a copy of each node for each possible time instant. Each temporal edge is then turned into a static edge from the two corresponding copies of its tail and head. We consider here a variant where we introduce copies of a node only for time instants corresponding to a departure time of an edge from that node, or an arrival time of an edge to that node, following the approach of [19].

Formally, given a temporal graph $G = (V, E)$, its *space-time representation* is a directed graph $D = (W, F^c \cup F^w)$, where:

- The nodes in W are labeled nodes v_τ, where $v \in V$ refers to a node of G and τ is a time label. More precisely, $v_\tau \in W$ if and only if there exists a temporal edge in E with tail v and departure time τ or a temporal edge with head v and arrival time τ. We will also refer to such nodes as *copies of* v. Let us denote with $Pred^w(v_\tau)$ the copy of v in W with maximum time label less than τ, if it exists.

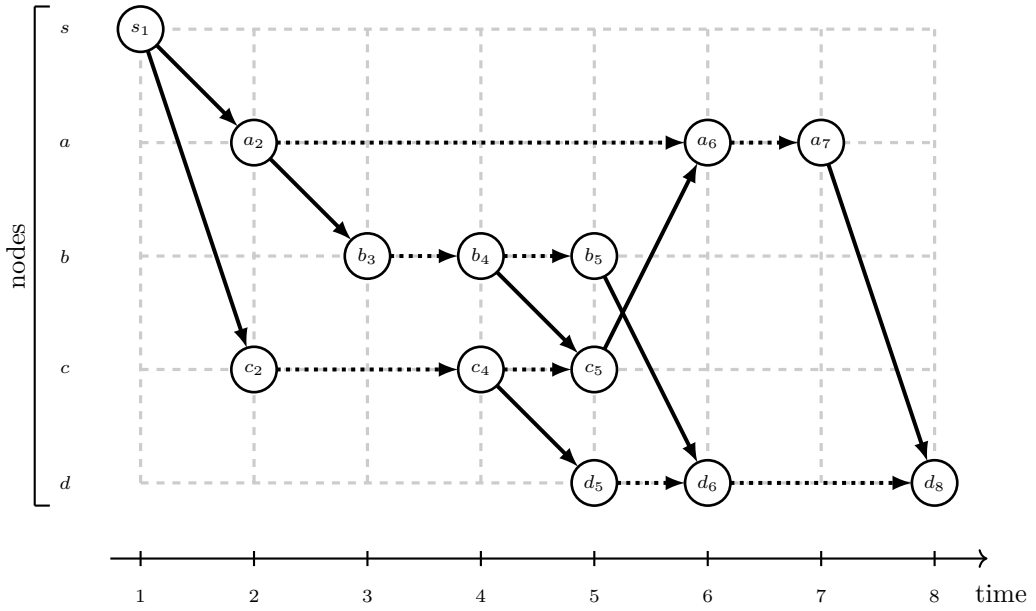

Figure 2 Space-time representation of the temporal graph of Figure 1. Plain arcs correspond to temporal edges while dotted arcs correspond to waiting at a node. The temporal walk $(s, a, 1, 1), (a, b, 2, 1), (b, c, 4, 1), (c, a, 5, 1), (a, d, 7, 1)$ corresponds to the directed path $s_1, a_2, b_3, b_4, c_5, a_6, a_7, d_8$. Note that the directed path $s_1, a_2, b_3, b_4, b_5, d_6$ does not corresponds to a valid temporal walk, since waiting at node b from time 3 to time 5 violates the constraint $\beta = 1$.

- We distinguish two types of arcs F^c and F^w called connection arcs and waiting arcs respectively. The set F^c contains an arc $(u_\tau, v_{\tau+\lambda})$ for each temporal edge $e = (u, v, \tau, \lambda) \in E$. These arcs represent a temporal connection between nodes in V and are called *connection arcs*. Note that each arc (v_τ, w_ν) in F^c satisfies $\tau < \nu$, since travel times are positive. The set F^w is defined to contain an arc $(Pred^w(v_\tau), v_\tau)$ for each $v \in V$ and for each copy v_τ of v such that $Pred^w(v_\tau)$ is defined. These arcs represent the possibility to wait at a node in $v \in V$ during a walk in G and are called *waiting arcs*. Note that each arc (v_τ, v_ν) in F^w satisfies $\tau < \nu$.

The main property of this representation is that any temporal walk Q corresponds to a directed path in the representation using arcs in F^c corresponding to temporal edges of Q plus waiting arcs in F^w each time the walk waits at a node (see Figure 2 for an example). Note that the converse is also true in the unrestricted waiting setting but not with waiting-time constraints.

We will now show that Algorithm 1 runs correctly when E^{arr} and E^{dep} satisfy weaker requirements and how to compute such lists from a space-time representation.

Let us introduce some orderings of temporal edges with respect to certain temporal criteria. We say that an ordering E^{ord} of the edges of a temporal graph G is *walk-respecting* when the edges of any walk Q in G appear in order in E^{ord}. Equivalently, E^{ord} is walk-respecting when for any pair $e, f \in E$ of edges such that f extends e, then $e <_{E^{ord}} f$, where $e <_{E^{ord}} f$ means that e appears before f in E^{ord}. Moreover, we say that an ordering E^{ord} of all the temporal edges is node-departure sorted if all edges departing from the same node are ordered by non-decreasing departure time in E^{ord}, that is we have $e <_{E^{ord}} f$ whenever $e, f \in E$ have same tail and satisfy $dep(e) < dep(f)$. Similarly, we say that an ordering E^{ord}

of all the temporal edges is node-arrival sorted if all edges arriving to the same node are ordered by non-decreasing arrival time in E^{ord}, that is we have $e <_{E^{ord}} f$ whenever $e, f \in E$ have same head and satisfy $arr(e) < arr(f)$.

Let us consider a temporal graph G and its temporal edges E. Let (E^{dep}, E^{arr}) be two lists representing all temporal edges in E, where E^{dep} is node-departure sorted, and E^{arr} is walk-respecting and node-arrival sorted. Notice that these hypothesis are weaker than the doubly-sorted representation we defined and used earlier. Indeed, if E^{arr} is sorted by non-decreasing arrival time, then it is trivially node-arrival sorted. It is also walk-respecting since we are assuming positive travel time of the temporal edges, thus the edges of a temporal walk Q have strictly increasing arrival times. On the other side, a simple example can prove that the opposite does not hold. Let $e = (s, u, \tau_1, \lambda_1)$ and $f = (s, v, \tau_2, \lambda_2)$ be two temporal edges such that $arr(e) < arr(f)$. Then $\{f, e\}$ is a node-arrival and walk-respecting sorted list, but it is not sorted by non-decreasing arrival time. We can now state the following.

▷ Claim 2. Given two lists (E^{dep}, E^{arr}), where E^{dep} is node-departure sorted, and E^{arr} is walk-respecting and node-arrival sorted, that represent a temporal graph with waiting constraints $G = (V, E, \alpha, \beta)$, and a source node $s \in V$, Algorithm 1 computes all s-reachable temporal edges in linear time and space.

We sketch a proof of Claim 2 by going through the key points that exploited the order of the lists of the proof of correctness in Theorem 1:

- In the preprocessing, it is still possible to compute in linear time, for each node v, the list E^{dep}_v by bucket sorting E^{dep}.
- Let e_k be the edge scanned during the k-th iteration. Then, if e_k appears in a walk Q in G_k it is still its last edge. The reason is that if it appears in a previous position in Q, then Q would contradict the walk-respecting hypothesis of E^{arr}.
- When proving that the algorithm correctly marks edges in $E^{dep}_v[l' : r']$, we used the non-decreasing arrival time property of E^{arr} to leverage that the edges entering v are scanned in E^{arr} by non-decreasing arrival time. This property actually coincides with the node-arrival definition.

We now show that an appropriate pair of lists (E^{dep}, E^{arr}) can easily be computed from a space-time representation.

▷ Claim 3. Given the space-time representation $D = (W, F^c \cup F^w)$ of a temporal graph $G = (V, E)$, it is possible to compute in linear time ans space two lists (E^{dep}, E^{arr}), where E^{dep} is node-departure sorted, and E^{arr} is walk-respecting and node-arrival sorted.

In order to prove Claim 3 we provide a simple algorithm based on Kahn's algorithm [12]. Indeed, because of the positive travel assumption, D is a directed acyclic graph. It is then possible to use Kahn's algorithm to compute a topological ordering of D in linear time that is an ordering W^{ord} of W such that $u_\tau <_{W^{ord}} v_\nu$ for all arcs $(u_\tau, v_\nu) \in F^c \cup F^w$.

Let us see how to compute a list of temporal edges E^{arr} that is node-arrival and walk-respecting sorted from such a topological ordering W^{ord} of D (see Figure 3 for an example). We start with an empty list E^{arr}. Then, for each node $v_\nu \in W^{ord}$, from the first one to the last one, we consider its incoming arcs (u_τ, v_ν) in F^c. For each of such arc (u_τ, v_ν), we append to E^{arr} the temporal edge $(u, v, \tau, \nu - \tau)$. The list we obtain this way is:

- Walk-respecting sorted: A temporal walk Q in G corresponds to a path P in D. Moreover, the time labels of the nodes in P are strictly increasing. Thus the topological order guarantees that we consider the arcs in $P \cap F^c$ in the same order as the corresponding temporal edges appear in Q.

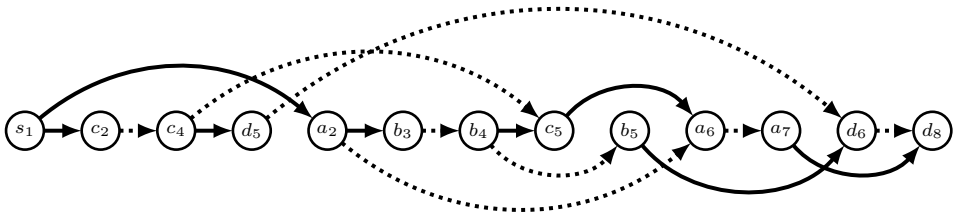

Figure 3 The topological order $s_1, c_2, c_4, d_5, a_2, b_3, b_4, c_5, b_5, a_6, a_7, d_6, d_8$ of the space-time representation of Figure 2 leads to the node-arrival sorted and walk-respecting ordering $E^{arr} = (s, c, 1, 1)$, $(c, d, 4, 1)$, $(s, a, 1, 1)$, $(a, b, 2, 1)$, $(b, c, 4, 1)$, $(c, a, 5, 1)$, $(b, d, 5, 1)$, $(a, d, 7, 1)$. It also results in the node-departure sorted and walk-respecting ordering $E^{dep} = (s, a, 1, 1)$, $(s, c, 1, 1)$, $(c, d, 4, 1)$, $(a, b, 2, 1)$, $(b, c, 4, 1)$, $(c, a, 5, 1)$, $(b, d, 5, 1)$, $(a, d, 7, 1)$.

- Node-arrival sorted: Since there is a path connecting the copies of each node $v \in V$ through increasing time labels, we are guaranteed to extract each copy by increasing time label. Thus the edges entering v will be considered and appended to E^{arr} by non-decreasing arrival time.

A list E^{dep} of temporal edges which is node-departure sorted (and walk respecting) can be similarly obtained in linear time by scanning out-arcs of each node in the topological ordering instead of in-arcs. As a consequence of Theorem 1, Claim 2 and Claim 3, we thus obtain:

▶ **Theorem 4.** *Given a doubly-sorted representation, or a space-time representation, of a temporal graph with waiting constraints $G = (V, E, \alpha, \beta)$ and a source node $s \in V$, it is possible to compute all s-reachable temporal edges in linear time and space.*

5 Conclusion

We provided an algorithm that, given in input a space-time representation of a temporal graph with waiting constraints, computes in linear time and space all the reachable edges from a given source. In particular, this also solves the single-source earliest arrival time problem. We are working on extending this technique for computing single-source optimal temporal walks optimizing classical criteria such as shortest duration or number of edges [4].

A future line of work consists in considering a more flexible model in the context of applications to public transport networks, for example by also allowing *footpaths* arcs that are available at any point in time.

References

1 Matthias Bentert, Anne-Sophie Himmel, André Nichterlein, and Rolf Niedermeier. Efficient computation of optimal temporal walks under waiting-time constraints. *Appl. Netw. Sci.*, 5(1):73, 2020.

2 Kenneth A. Berman. Vulnerability of scheduled networks and a generalization of menger's theorem. *Networks*, 28(3):125–134, 1996.

3 Gerth Stølting Brodal and Riko Jacob. Time-dependent networks as models to achieve fast exact time-table queries. *Electron. Notes Theor. Comput. Sci.*, 92:3–15, 2004.

4 Filippo Brunelli and Laurent Viennot. Minimum-cost temporal walks under waiting-time constraints in linear time. *CoRR*, abs/2211.12136, 2022. doi:10.48550/arXiv.2211.12136.

5 Richard T. Bumby. A problem with telephones. *SIAM. J. on Algebraic and Discrete Methods*, 2(1):13–18, 1981.

6 Arnaud Casteigts, Paola Flocchini, Walter Quattrociocchi, and Nicola Santoro. Time-varying graphs and dynamic networks. *IJPEDS*, 27(5):387–408, 2012.

7 Arnaud Casteigts, Anne-Sophie Himmel, Hendrik Molter, and Philipp Zschoche. Finding temporal paths under waiting time constraints. *Algorithmica*, 83(9):2754–2802, 2021.

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

9 Pierluigi Crescenzi, Clémence Magnien, and Andrea Marino. Approximating the temporal neighbourhood function of large temporal graphs. *Algorithms*, 12(10):211, 2019.

10 Julian Dibbelt, Thomas Pajor, Ben Strasser, and Dorothea Wagner. Intriguingly Simple and Fast Transit Routing. In *Experimental Algorithms*, Lecture Notes in Computer Science, pages 43–54. Springer, 2013.

11 Julian Dibbelt, Thomas Pajor, Ben Strasser, and Dorothea Wagner. Connection scan algorithm. *ACM Journal of Experimental Algorithmics*, 23:1.7:1–1.7:56, 2018.

12 Arthur B. Kahn. Topological sorting of large networks. *Commun. ACM*, 5(11):558–562, 1962.

13 Matthieu Latapy, Tiphaine Viard, and Clémence Magnien. Stream graphs and link streams for the modeling of interactions over time. *Social Netw. Analys. Mining*, 8(1):61:1–61:29, 2018.

14 Othon Michail. An introduction to temporal graphs: An algorithmic perspective. *Internet Mathematics*, 12(4):239–280, 2016.

15 Matthias Müller-Hannemann, Frank Schulz, Dorothea Wagner, and Christos D. Zaroliagis. Timetable information: Models and algorithms. In *ATMOS*, volume 4359 of *Lecture Notes in Computer Science*, pages 67–90. Springer, 2004.

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

17 Stefano Pallottino and Maria Grazia Scutellà. Shortest path algorithms in transportation models: classical and innovative aspects. Technical Report TR-97-06, University of Pisa, 1997.

18 Stefano Pallottino and Maria Grazia Scutellà. *Equilibrium and Advanced Transportation Modelling*, chapter Shortest path algorithms in transportation models: classical and innovative aspects, pages 245–281. Kluwer Academic Publishers, 1998.

19 Frank Schulz, Dorothea Wagner, and Karsten Weihe. Dijkstra's algorithm on-line: An empirical case study from public railroad transport. *ACM J. Exp. Algorithmics*, 5:12, 2000.

20 Huanhuan Wu, James Cheng, Silu Huang, Yiping Ke, Yi Lu, and Yanyan Xu. Path Problems in Temporal Graphs. *VLDB Endowment*, 7(9):721–732, 2014.