

Article

# Polynomial Time Algorithm for Shortest Paths in Interval Temporal Graphs <sup>†</sup>

Anuj Jain <sup>1,‡</sup>  and Sartaj Sahni <sup>2,\*</sup> <sup>‡</sup><sup>1</sup> Adobe Systems Inc., Lehi, UT 84043, USA; anujjain@adobe.com<sup>2</sup> CISE Department, University of Florida, Gainesville, FL 32611, USA

\* Correspondence: sahani@ufl.edu or sahani@cise.ufl.edu

<sup>†</sup> This paper is an extended version of our paper published in 2024 Sixteenth International Conference on Contemporary Computing.<sup>‡</sup> These authors contributed equally to this work.

**Abstract:** We develop a polynomial time algorithm for the single-source all destinations shortest paths problem for interval temporal graphs (ITGs). While a polynomial time algorithm for this problem is known for contact sequence temporal graphs (CSGs), no such prior algorithm is known for ITGs. We benchmark our ITG algorithm against that for CSGs using datasets that can be solved using either algorithm. Using synthetic datasets, experimentally, we show that our algorithm for ITGs obtains a speedup of up to 32.5 relative to the state-of-the-art algorithm for CSGs.

**Keywords:** interval temporal graphs; contact sequence temporal graphs; shortest paths

## 1. Introduction

Temporal graphs are dynamic graphs that can change over time. Edges and Vertices in a temporal graph may evolve over time and thus have temporal information associated with them. Temporal graphs can be used to model many real-world problems such as modeling the spread of viral diseases, traffic flow in road networks, information flow in social networks and other such problems representing dynamic connectivity [1–7]. Recently, temporal graphs have been used in machine learning as temporal graph neural networks (TGNs) [8].

Different models can be used to represent a temporal graph [3]. One such popular model is a contact sequence temporal graph (CSG). Every instance of time when a communication can be initiated from a vertex  $u$  to a vertex  $v$  is represented by a unique temporal edge  $(u, v, t, \lambda)$  in the CSG model.  $t$  is the departure time and  $\lambda$  is the travel time along this edge, such that the arrival time at  $v$  when departing  $u$  along this edge would be  $t + \lambda$ . Figure 1 shows an example of a CSG.

An alternate representation of a temporal graph is an interval temporal graph (ITG). Temporal information on an edge connecting two vertices  $(u, v)$  in an ITG is modeled as a vector of contiguous time intervals  $i = (s_i, c_i)$ , with an associated travel time  $\lambda_i$  for each interval  $i$ . Figure 2 illustrates an example of an ITG. Communication from  $u$  to  $v$  may be initiated at any time  $t$  such that  $s_i \leq t \leq c_i$  for one of the intervals  $i$  along this edge.

While every CSG has an equivalent ITG, the reverse is true only when time is discrete.

**Citation:** Jain, A.; Sahni, S.Polynomial Time Algorithm for Shortest Paths in Interval Temporal Graphs. *Algorithms* **2024**, *17*, 468. <https://doi.org/10.3390/a17100468>

Academic Editor: Enrico Corradini

Received: 23 August 2024

Revised: 12 October 2024

Accepted: 19 October 2024

Published: 21 October 2024



**Copyright:** © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

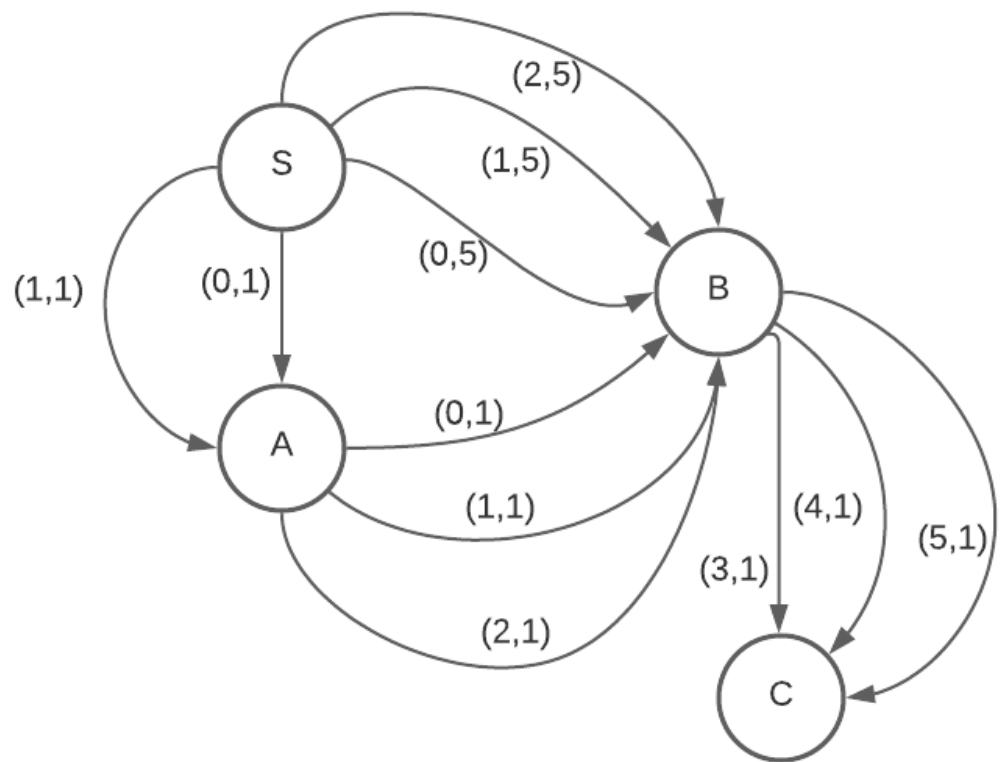


Figure 1. Contact sequence temporal graph.

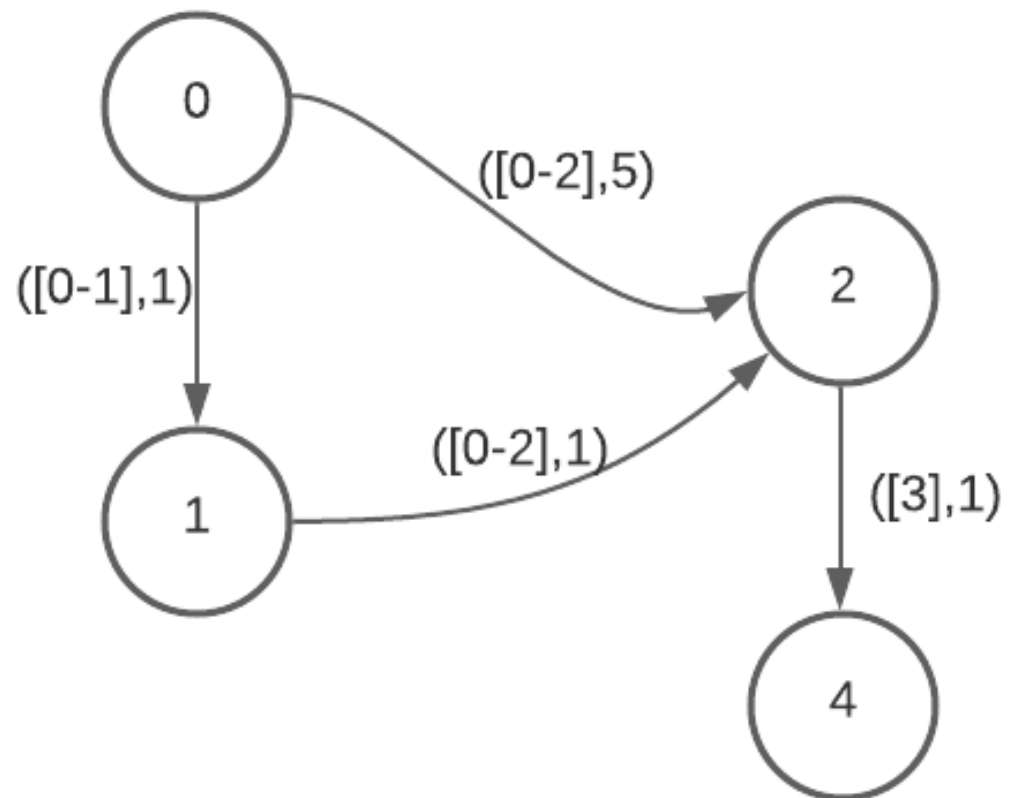


Figure 2. Interval temporal graph.

Some applications of temporal graphs are described below:

1. *Intelligent Transportation Systems*—Road networks can be modeled by temporal graphs where vertices represent street intersections [9]. Varying traffic and street conditions are represented as temporal edges. Such temporal graphs enable building intelligence in transportation systems to improve safety, mobility and efficiency.
2. *E-health and bioinformatics*—Temporal graphs can be used to model pandemic spread by building proximity networks [10–12]. This kind of modeling can help understand the source of pandemic spread and the best methods to contain the pandemic spread.
3. *Social Networks*—Social networks are usually large dynamic graphs. Refs. [1,13] present a framework for modeling social networks. Santoro et al. [5] propose an approach for studying temporal metrics in social graphs using temporal graphs.
4. *Artificial Intelligence*—Graph Neural Networks (GNNs) have become popular in recent times due to their ability to learn complex systems of relations or interactions. They provide a framework for deep learning models on graphs. Recently, new frameworks have been proposed for deep learning on dynamic or temporal graphs. For example, Rossi et al. [8] propose a framework for deep learning on temporal graphs that they call Temporal Graph Networks (TGNs). Smith et al. [14] propose a novel architecture for online learning with temporal neural networks.

Path and walk problems on temporal graphs are studied, for example, in [9,15–20]. While [16,18,19] focus on *ITGs*, Refs. [15,17,20,21] use the *CSG* model. Wu et al. [15] demonstrate that the studied path problems can be solved faster using the algorithms proposed by them on the *CSG* representation when compared to the algorithms proposed on the *ITG* representation by Xuan et al. [16]. However, Jain et al. [18,19] subsequently developed algorithms for *ITGs* that outperform the *CSG* algorithms of Wu et al. [15] for most of the studied path problems as well as the *ITG* algorithms of Xuan et al. [16]. Gheibi et al. [20] proposed an alternate *TRG* data structure for representing *CSGs* that results in faster algorithms than the algorithms of Wu et al. [15] for most of the studied path problems.

Bentert et al. [17] developed a polynomial time algorithm, for *CSGs*, to compute walks that optimize any linear combination of the optimization criteria studied by [15,16,18,20] with min and max waiting time constraints at each vertex. Jain et al. [19] show that a linear combination of multiple criteria can be used to find walks and paths with a secondary optimization criterion for, e.g., min-hop foremost (*mhf*) paths or min-wait foremost (*mwf*) walks.

Jain et al. [19] present algorithms for dual criteria *min – hop – foremost (mhf)* and *min – wait – foremost (mwf)* for *ITGs*. Their *ITG* algorithms solve the *mhf* and *mwf* problems in less time than when the algorithm of Bentert et al. [17] is used on the corresponding *CSGs* and the coefficients of the linear combination optimization criteria in Bentert et al.'s algorithm are chosen so as to compute *mhf* and *mwf* walks. Jain et al. [21] also develop an algorithm to compute walks optimizing any linear combination of the criteria considered by Bentert et al. [17] with waiting time constraints, outperforming the algorithm by Bentert et al. by up to a factor of 77. However, the limitation of the algorithm by Jain et al. is that it only works on *CSGs* with no zero-duration cycle.

In this paper, we develop an algorithm to find the shortest paths in *ITGs*. A shortest path between any two vertices *A* and *B* in a temporal graph is a feasible path for which the travel time to reach from *A* to *B* is minimum. Xuan et al. [16] propose a polynomial time algorithm to find min-hop paths in *ITGs*. A min-hop path from vertex *A* to *B* is a feasible path that travels through the minimum number of edges to reach *B*. Such a path is also the shortest path from *A* to *B* when all edges have the same travel time. However, in general, travel times on the edges of an interval temporal graph may be different. No algorithm has been proposed for finding the shortest paths when the temporal graph is expressed as an *ITG*. Wu et al. [15], Bentert et al. [17], Jain et al. [21] and others proposed algorithms that can be used to find the shortest paths when temporal graphs are expressed as *CSGs*. As noted earlier, *ITGs* cannot be expressed as *CSGs* when time is continuous. Further, even when time is discrete, the size of the *CSG* corresponding to a given *ITG* could be quite

large (when the time intervals of edges are large). This paper fills this gap by providing a polynomial time shortest path algorithm for *ITGs*.

Experimentally, it is demonstrated that as the contiguous time intervals that allow travel on edges in temporal graphs become larger, our *ITG* shortest path algorithm shows increasing performance gains over the algorithm by Wu et al. [15] running on the same temporal graph expressed as a *CSG*.

Our main contributions are as follows:

1. We develop a polynomial time algorithm to find the shortest paths in an *ITG*. To the best of our knowledge, no such algorithm is presently available in the literature.
2. We provide the complexity analysis of our shortest path algorithm.
3. We benchmark our algorithm for *ITGs* against that of Wu et al. [15] for finding the shortest paths in *CSGs*. Experimentally, we show that as the activity factor in a temporal graph increases due to large contiguous travel intervals, our algorithm shows increasing performance gains over that of Wu et al. [15] on equivalent temporal graphs expressed as *CSGs*. Using synthetic datasets, we show that our algorithm for *ITGs* obtains a speedup of up to 32.5 relative to the state-of-the-art algorithm for *CSGs*.

## 2. Problem Description

### Definitions

**Definition 1** (Contact sequence temporal graph). *In a contact sequence temporal graph  $G = (V, E)$ , each edge  $e \in E$  is a tuple  $(u, v, t, \lambda)$ , where  $t$  is a permissible departure time for travel from  $u$  to  $v$  along the edge  $e$  and  $\lambda$  is the amount of time it takes to travel on edge  $e$  from  $u$  to  $v$  when departing at time  $t$ . Thus,  $v$  is reached at time  $t + \lambda$ . If there are  $m$  time instances when departures from  $u$  to  $v$  are permissible, there will be  $m$  such temporal edges  $[(u, v, t_1, \lambda_1); (u, v, t_2, \lambda_2) \dots; (u, v, t_m, \lambda_m)]$ .  $m$  is the amount of activity on the connection  $(u, v)$ .*

**Definition 2** (Interval temporal graphs). *In an interval temporal graph  $G = (V, E)$ , each edge  $e \in E$  is represented by a tuple  $(u, v, \text{intvls})$ . This tuple represents a connection from  $u$  to  $v$ .  $\text{intvls}$  is a time-ordered non-overlapping vector of tuples  $[(s_1, c_1, \lambda_1); (s_2, c_2, \lambda_2); \dots; (s_n, c_n, \lambda_n)]$ . The  $i$ th interval starts at time  $s_i$  and closes (ends) at time  $c_i$ ;  $\lambda_i$  is the time it takes to traverse the edge when departing  $u$  at a time  $t$  if  $[s_i \leq t \leq c_i]$  ( $v$  is reached at time  $t + \lambda_i$ ). The intervals are in ascending order of start times  $s_i$ , and collectively, they define the permissible departure times from  $u$ .*

The permissible travel intervals for any edge of an *ITG* (Definition 2) can be adjusted such that for any two consecutive intervals  $[(s_i, c_i, \lambda_i); (s_{i+1}, c_{i+1}, \lambda_{i+1})]$  we have  $(c_i + \lambda_i \leq s_{i+1} + \lambda_{i+1})$ . This transformation is explained in Jain and Sahni [18]. In this paper, we assume that the intervals associated with each edge of an *ITG* satisfy this constraint.

As is evident from Figures 3 and 4, a given temporal graph may need a much larger number of edges to be expressed as a *CSG* versus an *ITG*. This is especially true when the graph has large contiguous travel intervals on the edges. Jain et al. [18] demonstrated that several temporal path and walk problems on *CSGs* may be polynomial in the size of the input graph but NP-hard for the equivalent *ITG* graph.

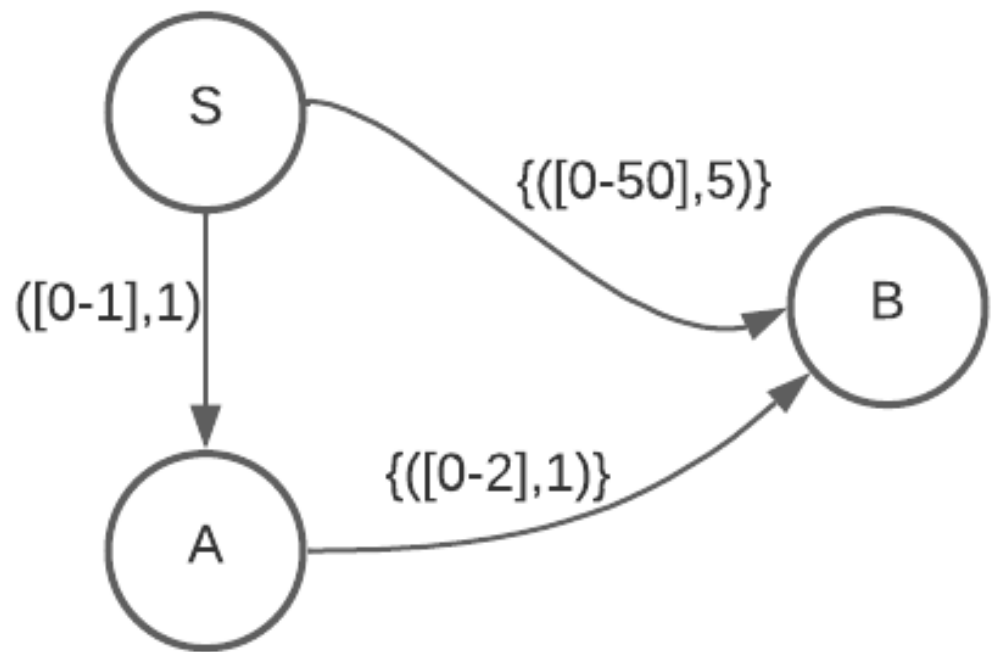


Figure 3. Interval temporal graph with large intervals.

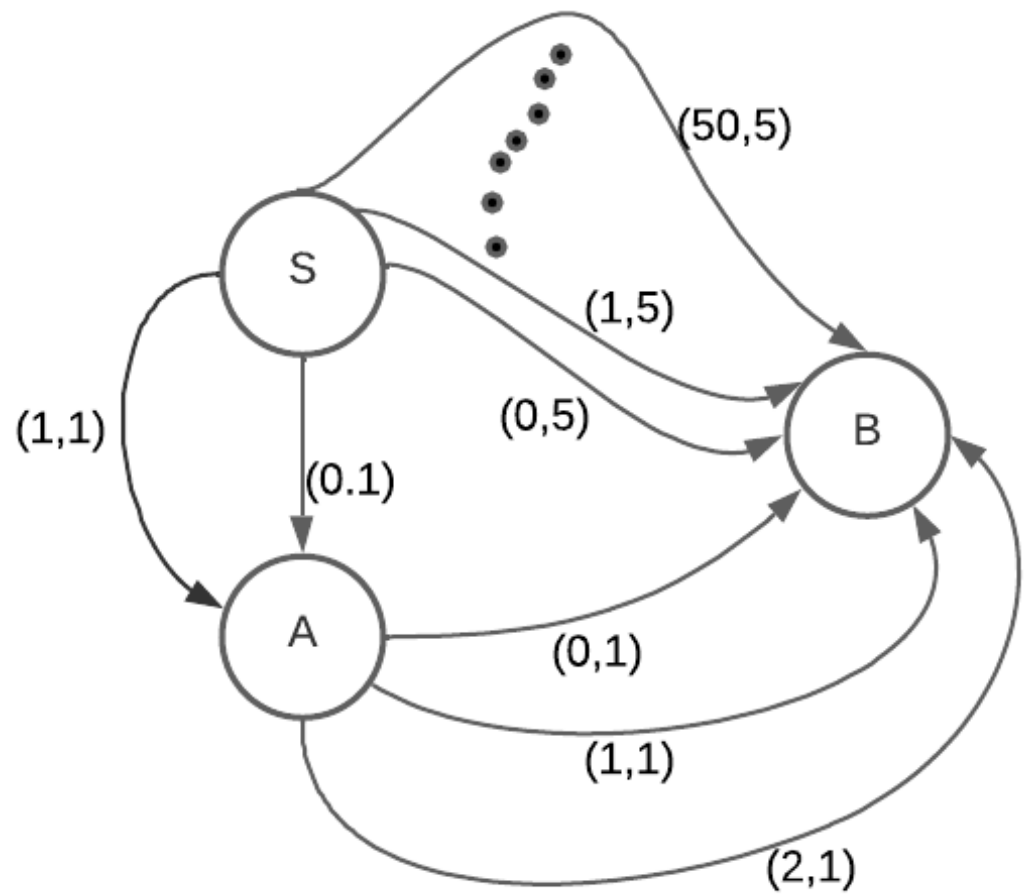


Figure 4. High-activity contact sequence graph.

**Definition 3** (Temporal paths and walks). A path (equivalently, valid path, feasible path, temporal path or time-respecting path),  $p = \{u_1, t_1, u_2, t_2, \dots, u_k\}$ , in a temporal graph is an alternating sequence of vertices and departure times from those vertices starting from a source vertex  $u_1$  to a destination vertex  $u_k$  such that no vertex appears more than once in this sequence.  $t_i$  is a permissible departure time from  $u_i$  to  $u_{i+1}$  and  $(t_i + \lambda_i) \leq t_{i+1}$ ,  $1 \leq i < k$ .  $(t_i + \lambda_i)$  is the arrival time at  $u_{i+1}$  when departing  $u_i$  at  $t_i$ ,  $1 \leq i < k$ . For path  $p$ ,  $u_1$  is the source or start vertex and  $u_k$  is the destination vertex. The number of hops is  $k - 1$ . When vertices are permitted to repeat in  $p$ ,  $p$  is a walk. Note that every path is also a walk but some walks are not paths.

**Definition 4** (Shortest paths). A shortest path  $p$  in a temporal graph is a feasible temporal path from a start vertex  $u_1$  to a destination vertex  $u_k$  with minimum length, where the length,  $len(p)$ , of a path  $p$  is the sum of the travel times  $\lambda_i$  on the edges  $e_i$  on this path.

$$len(p) = \sum_{i=1}^{k-1} \lambda_i \tag{1}$$

By contrast, a fastest path from  $u_1$  to  $u_k$  is a temporal path that minimizes the difference between the arrival time at  $u_k$  and the departure time from  $u_1$ . Such a path accounts for wait times at all intermediate vertices. Algorithms for fastest paths in interval and contact sequence temporal graphs may be found in [15–17,21]. Shortest paths are of interest in applications where there is a cost associated with traveling on an edge but no cost associated with waiting at a vertex. For example, the fuel cost depends on the sum of the  $\lambda_i$ s and is independent of the wait times at the vertices.

It is to be noted that the shortest path  $p$  from vertex  $u_1$  to  $u_k$  in a temporal graph is also the shortest walk from  $u_1$  to  $u_k$ . To see this, assume there is a walk  $w$  of shorter length. The walk  $w$  can be converted to a path  $p'$  from  $u_1$  to  $u_k$  by removing all of its cycles. Since edge travel times are non-negative,  $len(p') \leq len(w) < len(p)$ , contradicting the fact that  $p$  is a shortest path from  $u_1$  to  $u_k$ .

### 3. Shortest Paths

We need to find the shortest paths  $p$  from a start vertex  $s$  to all possible destination vertices  $u_k$  in a given ITG.

#### 3.1. Dominance Criteria

Let  $p_1$  and  $p_2$  be two paths from the start vertex  $s$  to the same end vertex  $v$ . We say that  $p_1$  dominates  $p_2$  iff every valid extension of  $p_2$  is also a valid extension of  $p_1$  and has the same or smaller length. A valid extension of a path  $p$  from  $s$  to  $v$  is a valid path  $p'$  from  $s$  to  $v'$ , such that  $p'$  goes through  $v$  and is the same as  $p$  from  $s$  to  $v$ . Let the arrival time and length of a path  $p$  from  $s$  (to its end vertex) be denoted by  $arr(p)$  and  $len(p)$ , respectively. It is easy to see that when Equation (2) is true,  $p_1$  dominates  $p_2$  (see Figure 5).

$$\begin{aligned} arr(p_1) &\leq arr(p_2) \\ len(p_1) &\leq len(p_2) \end{aligned} \tag{2}$$

When the arrival times and lengths of the paths  $p_1$  and  $p_2$  are as in Equation (3), then neither of the paths dominates the other. When neither path dominates the other, we say that  $p_1$  and  $p_2$  are pairwise non-dominant.

$$\begin{aligned} arr(p_1) &< arr(p_2) \\ len(p_1) &> len(p_2) \end{aligned} \tag{3}$$

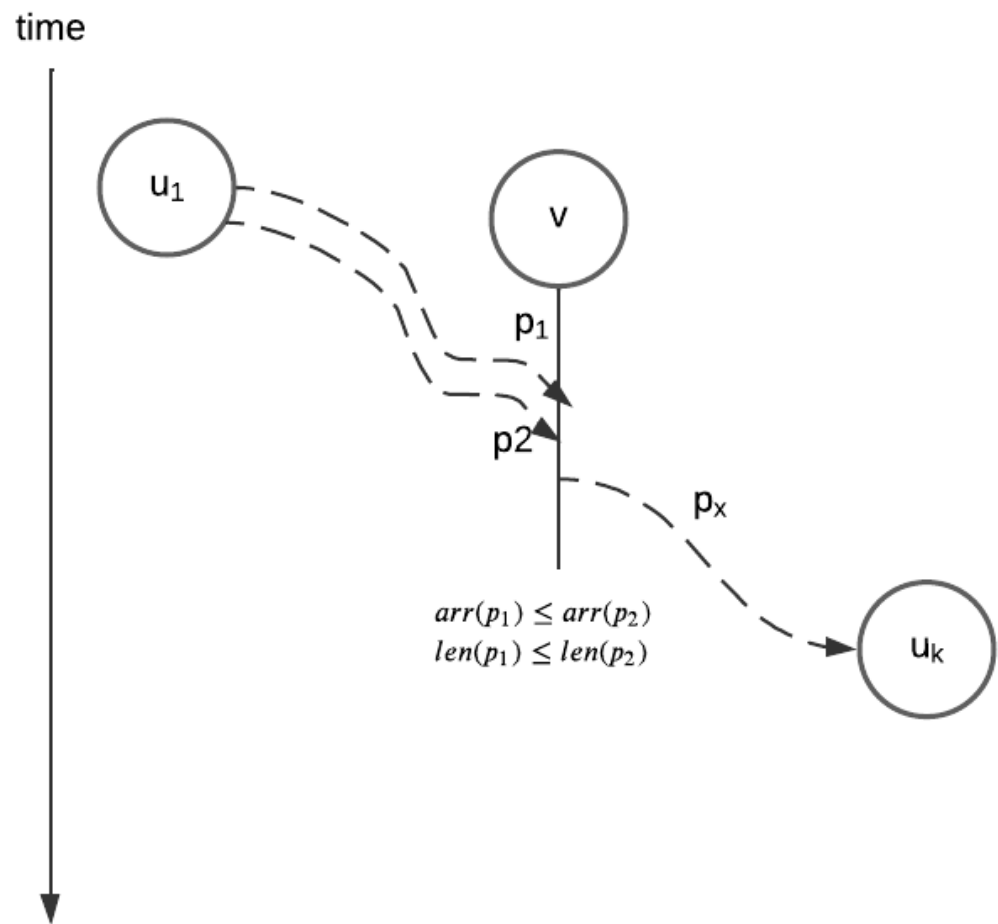


Figure 5.  $p_1$  dominates  $p_2$ .

### 3.2. Algorithm

We propose a hop-by-hop algorithm to determine a shortest path from a start vertex  $s$  to all possible destinations  $u_k$  in an *ITG*. At every hop count  $k$ , we construct a list of non-dominated paths from  $s$  to every reachable vertex  $v$  by extending the list of such paths for hop count  $k - 1$ . The algorithm terminates after a maximum of  $n - 1$  hops, where  $n$  is the total number of vertices in the *ITG* or when no new non-dominated paths are discovered in a given hop.

### 3.3. Overview

Let  $P(v, k)$  be the list of pairwise non-dominant  $r$ -hop paths  $(t, l)$  from  $s$  to  $v$  that arrive at  $v$  at time  $t$  and have length  $l, r \leq k$ ; the paths are in increasing order of  $t$  (equivalently, decreasing order of  $l$ ). Let  $P_{new}(v, k)$  be the list of non-dominated  $k$ -hop paths from  $s$  to  $v$ . A high-level description of our shortest paths algorithm can be found in Algorithm 1.

The lists  $P(v, 0)$  and  $P_{new}(v, 0)$  are initialized in lines 1 and 2 to be lists of 0-hop paths from  $s$  to  $v$ . The  $(k - 1)$ -hop paths in  $P_{new}(u, k - 1)$  are extended to  $k$ -hop paths by adding an outgoing edge  $(u, v)$  in lines 8 and 10. Line 8 considers one edge extensions that use the interval  $(i.st, i.c)$  for the case  $t < i.st$ . Of the possible extensions using this interval, the one that departs  $u$  at time  $i.st$  dominates the others and so only this one is added to  $P_{new}(v, k)$ . Line 10 considers the remaining case for a legitimate extension,  $i.st \leq t \leq i.c$ . Again, only the single non-dominated extension is added to  $P_{new}(v, k)$ . Line 13 removes dominated paths from  $P_{new}(v, k)$ . In Line 14, the paths in  $P(v, k - 1)$  and  $P_{new}(v, k)$  are merged to obtain  $P(v, k)$  (recall that  $P(v, k)$  is to contain only non-dominated paths). Additionally, paths in  $P_{new}(v, k)$  that are dominated by a path in  $P(v, k - 1)$  are eliminated from  $P_{new}(v, k)$ .

**Algorithm 1** Shortest Paths Pseudocode

---

```

1:  $P(s, 0) \leftarrow \{(0, 0)\}, \text{shrtst}P(s) = (0, 0); \forall v \neq s, P(v, 0) \leftarrow \emptyset, \text{shrtst}P(v) = \emptyset$ 
2:  $P_{\text{new}}(s, 0) \leftarrow \{(0, 0)\}; \forall v \neq s, P_{\text{new}}(v, 0) \leftarrow \emptyset$ 
3: for  $k = 1, \dots, n - 1$  do
4:    $\forall v, P_{\text{new}}(v, k) = \emptyset$ 
5:   for every edge  $(u, v) | P_{\text{new}}(u, k - 1) \neq \emptyset$  do
6:     for all  $(t, l) \in P_{\text{new}}(u, k - 1)$  do
7:        $\forall i \in (u, v) | t < i.st$  add  $(t', l') =$ 
8:          $(i.st + i.\lambda, l + i.\lambda)$  to  $P_{\text{new}}(v, k)$ 
9:       If there is an  $i$ , such that  $i.st \leq t \leq i.c$ , add
10:       $(t', l') = (t + i.\lambda, l + i.\lambda)$  to  $P_{\text{new}}(v, k)$ 
11:     end for
12:   end for
13:    $\forall v$ , eliminate dominated paths from  $P_{\text{new}}(v, k)$ 
14:    $\forall v$ , merge  $P_{\text{new}}(v, k)$  and  $P(v, k - 1)$  to get  $P(v, k)$  eliminating dominated paths from
15:    $P(v, k - 1)$  as well as paths in  $P_{\text{new}}(v, k)$  that are dominated by paths in  $P(v, k - 1)$ 
16:    $\forall v | P_{\text{new}}(v, k) \neq \emptyset, \text{shrtst}P(v) = P(v, k).shrtst()$ 
17:   If  $\forall v, P_{\text{new}}(v, k) = \emptyset$  terminate!
18: end for
19: return  $\text{shrtst}P[]$ 

```

---

**3.4. Pseudocode Example Walk-Through**

We will walk through the pseudocode in Algorithm 1 with an example graph of Figure 6. We are to find the shortest paths from  $s$  to all destination vertices  $v \in \{a, b, c, d\}$  in Figure 6. The lists  $P(v, 0)$  and  $P_{\text{new}}(v, 0)$  are initialized as  $\emptyset$  in lines 1 and 2 for  $v \in \{a, b, c, d\}$  as 0-hop paths.  $P(s, 0)$  and  $P_{\text{new}}(s, 0)$  are initialized as  $\{(0, 0)\}$ .  $\text{shrtst}P(s) = (0, 0)$

1. In hop 1,  $k = 1$ . The condition in line 5 evaluates to true only for the edge  $(s, a)$ . Therefore, lines 6 to 11 add two new paths to  $P_{\text{new}}(a, 1)$  as  $\{(2, 1), (4, 1)\}$ , extending  $(0, 0)$  along the two available intervals on edge  $(s, a)$ . Line 13 eliminates the path  $(4, 1)$  from  $P_{\text{new}}(a, 1)$  as it is dominated by the path  $(2, 1)$  as per Equation (2). Merging  $P_{\text{new}}(a, 1)$  with  $P(a, 0) = \emptyset$  in line 14 gives  $P(a, 1) = P_{\text{new}}(a, 1) = \{(2, 1)\}$ .  $\text{shrtst}P(a) = (2, 1)$
2. In hop 2,  $k = 2$ . Line 5 evaluates to true for edges  $(a, b)$  and  $(a, c)$  since  $P_{\text{new}}(a, 1) = \{(2, 1)\}$ . Hence,  $(2, 1)$  is extended along each of these edges in lines 6 to 11.
  - (a) Extending  $(2, 1)$  along the two intervals on edge  $(a, b)$  gives  $P_{\text{new}}(b, 2) = \{(3, 2), (17, 5)\}$
  - (b) Extending  $(2, 1)$  along the two intervals on edge  $(a, c)$  gives  $P_{\text{new}}(c, 2) = \{(6, 3), (10, 2)\}$

Line 13 eliminates the dominated path  $(17, 5)$  from  $P_{\text{new}}(b, 2)$ , whereas none of the paths are dominated in  $P_{\text{new}}(c, 2)$ . Lists  $P_{\text{new}}(b, 2)$  and  $P_{\text{new}}(c, 2)$  are merged with  $P(b, 1) = \emptyset$  and  $P(c, 1) = \emptyset$ , respectively, in line 14. Therefore, at the end of this iteration,  $P_{\text{new}}(b, 2) = P(b, 2) = (3, 2)$  and  $P_{\text{new}}(c, 2) = P(c, 2) = \{(6, 3), (10, 2)\}$ .  $\text{shrtst}P(b) = (3, 2); \text{shrtst}P(c) = (10, 2)$ .

3. In hop 3,  $k = 3$ . We have  $P_{\text{new}}(b, 2)$  and  $P_{\text{new}}(c, 2)$  as non-empty. Therefore, line 5 evaluates to true for edges  $(b, d)$  and  $(c, d)$ .
  - (a) Extension of paths in  $P_{\text{new}}(b, 2) = \{(3, 2)\}$  along the intervals on edge  $(b, d)$  gives  $P_{\text{new}}(d, 3) = \{(5, 4)\}$ .
  - (b) Extension of paths in  $P_{\text{new}}(c, 2) = \{(6, 3), (10, 2)\}$  along the intervals on edge  $(c, d)$  appends  $\{(11, 4), (11, 3)\}$  to  $P_{\text{new}}(d, 3)$ . Therefore,  $P_{\text{new}}(d, 3) = \{(5, 4), (11, 4), (11, 3)\}$ .

Line 13 eliminates the dominated paths from  $P_{\text{new}}(d, 3)$ . Therefore, the paths surviving are  $P_{\text{new}}(d, 3) = \{(5, 4), (11, 3)\}$ . After merging  $P_{\text{new}}(d, 3)$  with  $P(d, 2) = \emptyset$  in line 14 we obtain,  $P_{\text{new}}(d, 3) = P(d, 3) = \{(5, 4), (11, 3)\}$ .  $\text{shrtst}P(d) = (11, 3)$ .



Line 5 does not evaluate to true for any other edges when  $k = 4$ , as even though  $P_{new}(d, 3)$  is non-empty, there are no outgoing edges from  $d$ . Therefore, the algorithm terminates. The shortest paths to each of the vertices  $(s, a, b, c, d)$  are available in the array  $shrtstP$ .

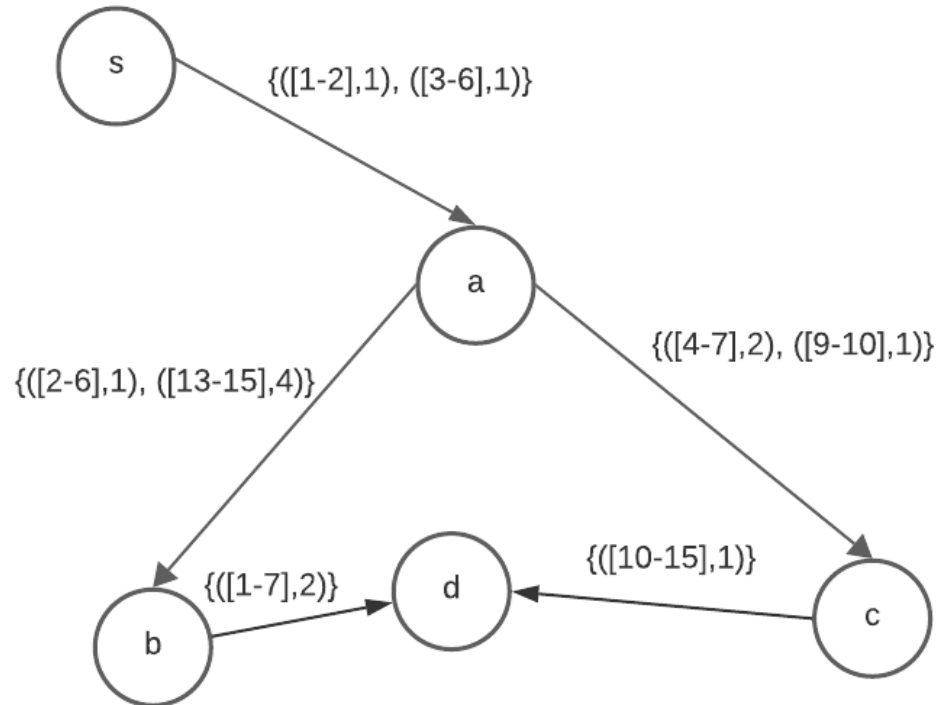


Figure 6. Shortest paths in ITG example.

### 3.5. Detailed Algorithm

Algorithm 2 is a refinement of Algorithm 1 that provides a greater level of detail.

The function  $nextI((u, v), (t, l))$  (line 12 of Algorithm 2) finds the earliest possible departure time  $fstout$  and corresponding interval  $i_F$  on the edge  $(u, v)$ . If  $t$  is in the interval  $(i_F.st, i_F.c)$ ,  $fstout$  is the same as  $t$ , otherwise it is  $i_F.st$ . The lists of paths  $(t, l)$  arriving at a vertex are always kept in increasing order of  $t$ . In a list of non-dominated paths, the increasing order of  $t$  is the same as the decreasing order of  $l$  due to the dominance criteria of Section 3.1. In line 17,  $prependDom(P''_{new}(v, k), (t', l'))$  prepends extended path  $(t', l')$  to the list  $P''_{new}(v, k)$  since paths  $(t, l)$  from the predecessor vertex  $u$  are extended in decreasing order of  $t$ . After prepending  $(t', l')$  to  $P''_{new}(v, k)$ ,  $prependDom$  eliminates any subsequent paths in the list that  $(t', l')$  dominates.

In lines 11 to 24, every  $(t, l)$  is extended using an outgoing edge  $(u, v)$  and the earliest possible departure time  $fstout (\geq t)$  from  $u$ . Then,  $(t, l)$  is also extended at  $i.st$  for every subsequent interval  $i$  on  $(u, v)$ , such that  $i.st < prev(fstout)$ , where  $prev(fstout)$  is the extension time of a previously examined path  $(t_p, l_p)$  at  $u$ . This is because extension of  $(t, l)$  would be dominated by the extension of  $(t_p, l_p)$  if it starts at  $t_x \geq prev(fstout)$  with the same travel time  $\lambda_x$  since  $l_p < l$ .

In line 25, the list of extensions of all paths from  $u$  to  $v$  is merged with any other such list from a different incoming edge  $(u', v)$  in the current hop, retaining only non-dominated paths. The resulting  $P'_{new}(v, k)$  is added to  $nxtPLists$ , which maintains all such lists obtained at every vertex  $v$  in hop  $k$ .

**Algorithm 2** Shortest Paths Detailed Algorithm

---

```

1: Initialize  $P(v,0) \leftarrow NULL, \forall v \neq s; P(s,0) \leftarrow (0,0)$ 
2: Initialize  $newPLists$  as empty list.  $P_{new}(s,0) \leftarrow (0,0)$ .
3:  $\forall v \neq s, shrtstP[v] \leftarrow (\infty, \infty); shrtstP[s] \leftarrow (0,0)$ 
4: Add  $[s, P_{new}(s,0)]$  to  $newPLists$ 
5: Initialize  $k \leftarrow 0; newPaths \leftarrow 1$ 
6: while  $k < n - 1$  and  $newPaths > 0$  do
7:    $newPaths \leftarrow 0; k++$ 
8:   for all  $u$  in  $newPLists$  do
9:     for all neighbors  $v$  of  $u$  do
10:      Initialize  $prevPath(fstout) \leftarrow \infty$ 
11:      for all  $(t,l) \in P_{new}(u,k-1)$  in dec order of  $t$  do
12:         $(i_F, fstout) \leftarrow nextI((u,v), (t,l))$ 
13:        if  $fstout \geq prevPath(fstout)$  then
14:          continue
15:        end if
16:         $(t', l') \leftarrow (fstout + i_F.\lambda, l + i_F.\lambda)$ 
17:         $prpndDom(P''_{new}(v,k), (t', l'))$ 
18:        for  $\{i_x \in (u,v) \mid$ 
19:           $i_x.c \leq i_x.st < prevPath(fstout)\}$  do
20:           $(t', l') \leftarrow (i_x.st + i_x.\lambda, l + i_x.\lambda)$ 
21:           $insrtDom(P''_{new}(v,k), (t', l'))$ 
22:        end for
23:         $prevPath(fstout) \leftarrow fstout$ 
24:      end for
25:       $addOrmerge([v, P''_{new}(v,k)], nxtPLists)$ 
26:    end for
27:  end for
28:  for all  $u$  in  $nxtPLists$  do
29:     $(P(u,k), P_{new}(u,k)) \leftarrow$ 
30:     $EvalAndMerge(P(u,k-1), P'_{new}(u,k))$ 
31:     $shrtstP[u] \leftarrow P(u,k).last()$ 
32:    if  $P_{new}(u,k).size() > 0$  then
33:       $newPaths++$ 
34:    end if
35:  end for
36:   $newPLists \leftarrow nxtPLists$ 
37:   $nxtPLists.clear()$ 
38: end while
39: return  $shrtstP$ 

```

---

Finally, in line 30, all  $P'_{new}(u,k)$  at every vertex  $u$  available in  $nxtPLists$ , are evaluated against their respective  $P(u,k-1)$  to obtain the non-dominated paths in hops 1 through  $k$  as  $P(u,k)$  and the new paths in current hop as  $P_{new}(u,k)$ .

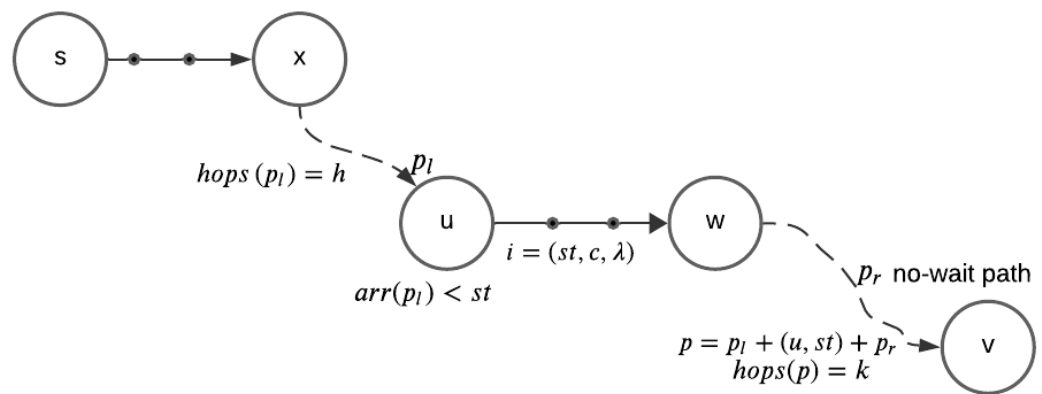
The algorithm continues until the terminating conditions are met.

**Theorem 1.** Algorithm 2 finds the shortest paths from the start vertex  $s$  to all reachable vertices  $v$ .

**Proof.** In an  $n$ -vertex temporal graph, a shortest path from a start vertex  $s$  to any destination vertex  $v$  is a non-dominated path with  $\leq n - 1$  hops. In Algorithm 2, we examine all non-dominated paths starting from  $s$  and keep a record of the shortest path discovered in line 31. Due to our scheme of resolving the ties described in Section 3.2, any walks with cycles are eliminated as these are dominated by paths that have no cycles and have fewer hops. Therefore, the shortest path obtained is a shortest valid path with no cycles.  $\square$

### 3.6. Complexity Analysis

We may associate a label  $(eid, st, h)$  with each path  $p \in P(v, k)$ ,  $v \neq s$ , and  $k > 0$ , where  $eid$  is the id of the last edge,  $(u, w)$ , in  $p$  with a departure time that is the start,  $st$ , of some travel interval of  $(u, w)$  and  $h$  is the number of hops in the path  $p$  from  $s$  to  $u$ . Note that every  $p \in P(v, k)$  has such a  $(u, w)$  as the departure time for the first edge  $(s, x)$  of  $p$  is the start of some interval of  $(s, x)$  and the number of edges in  $p$  is finite. Let  $p_l$  be the portion of  $p$  from  $s$  to  $u$  and  $p_r$  that from  $u$  to  $v$  (see Figure 7). From the way our algorithm works, it follows that  $p_l$  is an  $h$ -hop path in  $P(u, h)$ . Further, since the departure time for no edge in  $p_r$  is a start time of a travel interval for that edge (except  $st$  on  $(u, w)$ ),  $p_r$  encounters no wait at intermediate vertices and the length,  $len(p_r)$ , (i.e., sum of the edge travel times) of  $p_r$  is the same as its duration (arrival time at  $v$  - departure time from  $u$ ).



**Figure 7.** A  $p \in P(v, k)$  and its label  $(eid, st, h)$ .

**Theorem 2.** No  $P(v, k)$  has two paths with the same label.

**Proof.** Assume there is a  $P(v, k)$  that has two paths  $p_1$  and  $p_2$  with the same label  $(eid, st, h)$ . We shall show that this assumption leads to a contradiction and so must be false. Note that  $p_1$  and  $p_2$  are pairwise non-dominant.

1. Case  $h = 0$ . Now,  $p_{1l}$  and  $p_{2l}$  are empty. Thus,  $len(p_1) = len(p_{1r}) = arr(p_1) - st$  and  $len(p_2) = arr(p_2) - st$ . When  $len(p_1) < len(p_2)$ , we have  $arr(p_1) < arr(p_2)$ . Thus,  $p_1$  dominates  $p_2$ , which contradicts the fact that  $p_1$  and  $p_2$  are pairwise non-dominant. A contradiction is similarly obtained for the cases  $len(p_1) = len(p_2)$  and  $len(p_1) > len(p_2)$ .
2. Case  $h > 0$ . Now,  $p_{1l}$  and  $p_{2l}$  are  $h$ -hop paths from  $s$  to the same vertex  $u$ . The first edge of  $p_{1r}$  and  $p_{2r}$  is  $(u, w)$  and both these paths start from  $u$  at time  $st$ . Thus,

$$arr(p_1) = st + len(p_{1r}) \tag{4}$$

$$arr(p_2) = st + len(p_{2r}) \tag{5}$$

$$len(p_1) = len(p_{1l}) + len(p_{1r}) \tag{6}$$

$$len(p_2) = len(p_{2l}) + len(p_{2r}) \tag{7}$$

- (a) Case  $len(p_{1l}) = len(p_{2l})$ . Now,  $p_{1l}$  and  $p_{2l}$  must be the same path as, otherwise, one dominates the other, which is not possible as both are in  $P(u, r)$ . If  $arr(p_1) < arr(p_2)$ , then  $len(p_{1r}) < len(p_{2r})$  (Equations (4) and (5)). Thus,  $len(p_1) < len(p_2)$  (Equations (6) and (7)). Hence,  $p_1$  dominates  $p_2$ , a contradiction. A similar proof shows that when  $arr(p_1) > arr(p_2)$ , we obtain the contradiction that  $p_2$  dominates  $p_1$ .
- (b) Case  $len(p_{1l}) < len(p_{2l})$ . Let  $p_3$  be the  $s$  to  $v$  path obtained by concatenating  $p_{1l}$  and  $p_{2r}$ . Note that  $p_3$  is a  $k$ -hop time-respecting path with  $arr(p_3) = arr(p_2)$  and  $len(p_3) = len(p_{1l}) + len(p_{2r}) < len(p_2)$  (Equation (7)). Hence,  $p_3$  domi-

nates  $p_2$ . Thus, either  $p_3$  or a path  $p_4$  that dominates both  $p_2$  and  $p_3$  must be in  $P(v, k)$ . This means that  $p_2$  cannot be in  $P(v, k)$ .

- (c) Case  $\text{len}(p_{1_i}) > \text{len}(p_{2_i})$ . This is similar to the previous case.

□

**Theorem 3.** No  $P_{\text{new}}(v, k)$  has two paths with the same label.

**Proof.** This is similar to that of Theorem 2. □

The number of distinct path labels  $(eid, st, h)$  is at most  $i * n$ , where  $i$  is the total number of travel intervals across all edges of the temporal graph. From Theorems 2 and 3, it follows that  $|P(v, k)| \leq in$  and  $|P_{\text{new}}(v, k)| \leq in$  for all  $v$  and  $k$ .

For each value of  $k$ ,  $k > 0$ , the at most  $in$  paths in  $P_{\text{new}}(u, k - 1)$  are considered in increasing order of length (equivalently, decreasing order of arrival time at  $u$ ) and extended by one hop using an out edge from  $u$ . When extending using the edge  $(u, v)$  for each of these up to  $in$  paths, a binary search is performed over the travel intervals of this edge to determine the first interval for a valid extension (line 12 of Algorithm 2). The remaining (larger start time) intervals are then examined serially until the first interval that has already been used for the extension of a previously considered path in  $P(v, k - 1)$  is encountered. Let  $\delta$  be the maximum number of travel intervals on any edge. We see that  $O(in \log \delta)$  time is spent performing the binary searches for the up to  $in$  paths and  $O(\delta + in)$  time in examining the remaining intervals. The total time to process  $(u, v)$  is therefore  $O(in \log \delta)$  (note that  $\delta \leq i$ ). The number of  $k$ -hop paths generated by extending the paths in  $P_{\text{new}}(u, k - 1)$  using the edge  $(u, v)$  is  $O(in + \delta) = O(in)$ .

To compute  $P_{\text{new}}(v, k)$ , we need to compute a list of extensions of the paths in  $P_{\text{new}}(u, k - 1)$  for all  $u$  such that  $(u, v)$  is an edge of the temporal graph and merge these lists together, eliminating dominated paths. There are at most  $\text{indegree}(v)$  such path lists to be merged. Each of these has  $O(in)$  paths and takes  $O(in \log \delta)$  time to compute. The time needed to compute all of these lists is therefore  $O(in \log \delta * \text{indegree}(v))$ . Pairwise merging these path lists takes  $O(in * \text{indegree}(v))$  time (note that during the pairwise merge of two path lists, dominated paths are eliminated, so, from Theorem 3, it follows that the list size remains  $O(in)$ ; two ordered path lists may be merged in linear time eliminating dominated paths). An alternative to pairwise merging of the path lists is to merge the  $\text{indegree}(v)$  lists simultaneously using a loser tree. This reduces the merging time to  $O(in * \log \text{indegree}(v))$ . Regardless, the total time needed to compute  $P_{\text{new}}(v, k)$  is  $O(in \log \delta * \text{indegree}(v))$ . Hence, for any  $k$ , all  $P_{\text{new}}(v, k)$ s may be computed in  $O(\text{ine} \log \delta)$  time, where  $e = \sum \text{indegree}(v)$  is the number of edges in the temporal graph.

For any  $k$  and  $v$ ,  $P_{\text{new}}(v, k)$  may be merged with  $P(v, k - 1)$ , eliminating dominated pairs from both lists in  $O(in)$  time as both lists are ordered by arrival time and of size  $O(in)$  (Theorems 2 and 3). Thus, the overall time taken by Algorithm 2 for each  $k$  is  $O(\text{ine} \log \delta)$  and the time over all  $ks$  is  $O(\text{in}^2 e \log \delta)$ , which is polynomial in the number of inputs.

#### 4. Experimental Results

In this section, we compare the relative performance of Algorithm 2 for ITGs with the shortest path algorithm of Wu et al. [15] running on equivalent CSGs. Our experimental platform was an Intel Core i9-7900X CPU with a 3.30 GHz processor and 64 GB RAM. The C++ code for the shortest path algorithm of Wu et al. was obtained from the authors of [17]. Our algorithm was also coded in C++. The codes were compiled using the g++ ver. 7.5.0 compiler with option O2. The datasets used for comparison of the relative performance are described in Section 4.1.

#### 4.1. Datasets

Ref. [15] uses datasets from the Koblenz network [22]. These datasets have an (edge) activity that ranges from a low of about 1 to a high of about 3.67, which is rather low. Further, all  $\lambda$  values are 1. The Koblenz datasets are described in Table 1. These datasets are in the form of CSGs where every allowed travel from one vertex to another is a single time instance. To benchmark our algorithm against the shortest paths algorithm of [15] on a wide variety of temporal graphs, we prepared synthetic datasets using the Koblenz graphs as follows:

1. Obtain the underlying static graph from each Koblenz CSG. For this, we replaced each temporal edge  $(u, v, t, \lambda)$  by the static edge  $(u, v)$  and eliminated duplicate static edges.
2. Represent the static graph obtained in the previous step as an array adjacency list.
3. On every static edge, we randomly add temporal intervals. The temporal intervals are added using three random variables, number of allowed travel intervals  $I$ , duration of each interval  $D$  and the travel time on each interval  $T$ . Values are assigned to each of these random variables using a normal distribution around three parameters  $(\mu_I, \mu_D, \mu_T)$ , respectively. These parameters are the mean values for the normal distribution defining the three random variables  $(I, D, T)$ , respectively.

**Table 1.** Koblenz graph statistics.

Dataset	$ V $	$ E_s $	<i>cs</i> – Edges	Activity
epin	131.8 K	840.8 K	841.3 K	1
elec	7119	103.6 K	103.6 K	1
fb	63.7 K	817 K	817 K	1
growth	1870.7 K	39,953 K	39,953 K	1
youtube	3223 K	9375 K	9375 K	1
digg	30.3 K	85.2 K	87.6 K	1.02
slash	51 K	130.3 K	140.7 K	1.07
conflict	118 K	2027.8 K	2917.7 K	1.43
arxiv	28 K	3148 K	4596 K	1.45
enron	87,274	320.1 K	1148 K	3.58

We increase the activity factor in the temporal graph by gradually increasing the value of the parameter  $\mu_D$ . For each *ITG* obtained using the method described above, we obtain an equivalent CSG to benchmark against the shortest path algorithm of [15]. The synthetic datasets obtained as outlined above are described in Table 2. For the *growth* dataset, the number of contact sequence edges is too large to accommodate in a 32-bit integer for  $\mu_D = 20$  and  $\mu_D = 50$ . Therefore, it was infeasible to build a corresponding CSG and run the algorithm from [15] on it. However, we could still run our algorithm on the *ITG* representation; the run times were quite reasonable. Table 3 reports the size of each of the synthetic datasets on disk. As we increase the activity factor of the temporal graph by increasing the  $\mu_D$  parameter, the size of the CSG representation of the graph increases significantly. For example, for the *enron* dataset, the size of the *ITG* is 19.5 MB for  $\mu_D = 50$ , as compared to 10.7 GB for the CSG representation of the same temporal graph, which is a ratio of approximately 563.

**Table 2.** Synthetic graphs' statistics.

Graphs with $\mu_I = 4, \mu_T = 3$					
Dataset	$ V $	$ E_s $	cs-Edges ( $\mu_D = 5$ )	cs-Edges ( $\mu_D = 20$ )	cs-Edges ( $\mu_D = 50$ )
epin	131.8 K	840.8 K	17.8 M	65.2 M	160.6 M
elec	7119	103.6 K	2.2 M	8 M	19.8 M
fb	63.7 K	817 K	17.3 M	63.3 M	156 M
growth	1870.7 K	39,953 K	848.3 M	–	–
youtube	3.2 M	9.3 M	199 M	727.2 M	1.8B
digg	30.3 K	85.2 K	1.8 M	6.6 M	16.2 M
slash	51 K	130.3 K	2.7 M	10.1 M	24.9 M
conflict	118 K	2027.8 K	43.07 M	157.3 M	387.5 M
arxiv	28 K	3148 K	66.8 M	244.2 M	601.7 M
enron	87,274	320.1 K	6.8 M	24.8 M	61.1 M

**Table 3.** Synthetic graphs' size on disk.

Dataset	ITG			CSG		
	$\mu_D = 5$	$\mu_D = 20$	$\mu_D = 50$	$\mu_D = 5$	$\mu_D = 20$	$\mu_D = 50$
epin	51.93 MB	51.94 MB	51.95 MB	323 MB	1.18 GB	2.9 GB
elec	6.2 MB	6.23 MB	6.23 MB	36.1 MB	132.2 MB	325.7 MB
fb	50.5 MB	50.5 MB	50.5 MB	314.8 MB	1.1 GB	2.83 GB
growth	2.55 GB	2.55 GB	2.55 GB	17.1 GB	–	–
youtube	609.3 MB	609.3 MB	609.3 MB	4.2 GB	15.5 GB	38.1 GB
digg	5.2 MB	5.2 MB	5.2 MB	31.4 MB	114.8 MB	282.6 MB
slash	8 MB	8 MB	8 MB	49 M	179.4 MB	442.2 MB
conflict	126 MB	126 MB	126 MB	794.5 MB	2.9 GB	7.1 GB
arxiv	191.8 MB	191.8 MB	191.8 MB	1.15 GB	4.21 GB	10.37 GB
enron	19.58 MB	19.58 MB	19.58 MB	1.18 GB	4.3 GB	10.7 GB

#### 4.2. Run Times

Since the CSGs are much larger in size as compared to the corresponding ITGs, the reading time (from disk) of the CSG is also significantly larger. The comparison of the reading times for the graphs by our algorithm and by that of Wu et al. [15] is reported in Table 4. It is important to note that the reading times for our algorithm are almost the same for the different values of  $\mu_D$ . However, for the CSGs, the reading time increases proportionately to the increase in the activity factor. This is because the number of contact sequence edges also increases proportionately. The size of the ITG representation, however, remains almost similar across different values of the parameter  $\mu_D$ .

Comparisons of run times (excluding the time to read in the temporal graph) of the two algorithms on all the datasets are reported in Table 5. As expected, the run times of our algorithm remains almost the same across the different values of  $\mu_D$ . However, the run times of the algorithm in [15] increases proportionately to the increase in the value of  $\mu_D$  and, hence, the number of contact sequence edges in the graph. Therefore, our algorithm shows increasing performance gains in run time and in memory consumption over that of Wu et al. [15] when the temporal graph has larger contiguous travel intervals. For our experiments, we limited the duration of the travel intervals to follow a normal distribution around  $\mu_D = 50$ . This is because for much larger travel intervals, the algorithm of Wu et al. [15] runs out of memory due to the large number of contact sequence edges. In contrast, our algorithm comfortably handles large travel durations. One example of this is the *growth* dataset for which our algorithm runs in reasonable times for the values of  $\mu_D = 20$  and  $\mu_D = 50$ , whereas it is infeasible to represent the CSG using 32-bit integers for  $\mu_D = 20$ .

The performance gains obtained by our algorithm over that of Wu et al. [15] with increasing activity factor are shown in Figure 8.

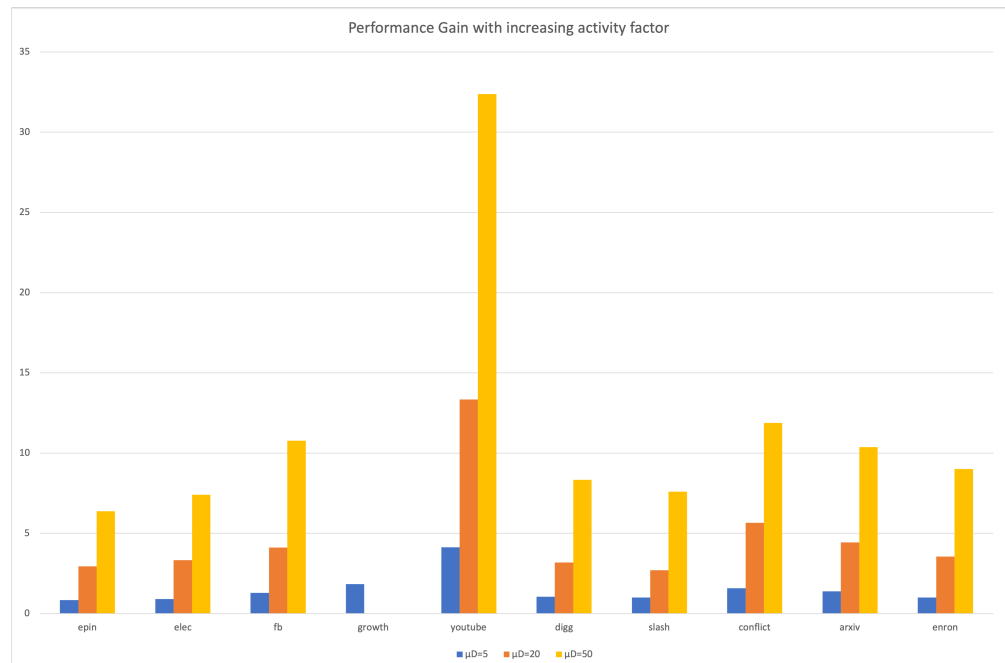


Figure 8. Performance gains with increasing activity factor.

Table 4. Synthetic graphs’ reading times in seconds.

Dataset	Ours on ITG			CSG Algorithm		
	$\mu_D = 5$	$\mu_D = 20$	$\mu_D = 50$	$\mu_D = 5$	$\mu_D = 20$	$\mu_D = 50$
epin	1.03	1.03	1.03	5.6	20.2	51.5
elec	0.14	0.13	0.12	0.69	2.5	6.1
fb	1	0.99	0.97	5.4	19.2	48.8
growth	50.5	50	50.5	271.65	–	–
youtube	11.8	12.09	12.34	64.4	235.1	578.1
digg	0.12	0.12	0.1	0.57	2.03	4.9
slash	0.17	0.16	0.16	0.88	3.1	7.6
conflict	2.4	2.4	2.4	13.3	48.9	121.7
arxiv	3.6	3.6	3.6	20.4	75.7	189.6
enron	0.4	0.39	0.39	2.1	7.7	19.2

Table 5. Synthetic graphs’ run time in seconds comparison.

Dataset	Ours on ITG			CSG Algorithm		
	$\mu_D = 5$	$\mu_D = 20$	$\mu_D = 50$	$\mu_D = 5$	$\mu_D = 20$	$\mu_D = 50$
epin	0.78	0.77	0.91	0.66	2.27	5.8
elec	0.044	0.045	0.05	0.04	0.15	0.37
fb	0.17	0.17	0.18	0.22	0.7	1.94
growth	105	121	140	192.8	–	–
youtube	0.63	0.63	0.63	2.6	8.4	20.4
digg	0.019	0.022	0.024	0.02	0.07	0.18
slash	0.1	0.1	0.11	0.1	0.27	0.76
conflict	0.57	0.53	0.64	0.9	3	7.6
arxiv	0.71	0.79	0.81	0.97	3.5	8.3
enron	0.08	0.09	0.1	0.08	0.32	0.9



### 4.3. Memory Footprint

The ratio of the memory footprint of the algorithm of [15] to the memory footprint of our algorithm is similar to the ratio of the sizes of the corresponding temporal graph representation as CSG and ITG, respectively. This is because both algorithms require the entire graph to be present in memory. As is evident from Table 3, this ratio increases significantly as the activity factor increases, going as high as 560 for our synthetic graphs.

## 5. Conclusions

We have developed a polynomial time shortest paths algorithm for the ITG representation of temporal graphs. To the best of our knowledge, such an algorithm is not available in the literature. Our algorithm is suitable for temporal graphs with a high activity factor and large contiguous travel intervals. Our algorithm shows increasing performance gains over the known state-of-the-art shortest paths algorithm for CSGs as the activity factor on the temporal graphs increases. Using synthetic datasets, experimentally, we show that our algorithm for ITGs obtains a speedup of up to 32.5 relative to the state-of-the-art algorithm for CSGs. For graphs with very large activity factors, the CSG algorithm is infeasible, while our algorithm can handle such datasets comfortably.

**Author Contributions:** Conceptualization, A.J. and S.S.; methodology, A.J. and S.S.; software, A.J.; validation, A.J.; formal analysis, A.J. and S.S.; data curation, A.J.; writing—original draft preparation, A.J. and S.S.; writing—review and editing, S.S.; visualization, A.J.; supervision, S.S.; project administration, S.S. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research received no external funding.

**Data Availability Statement:** Data sharing is not applicable.

**Conflicts of Interest:** Author Anuj Jain was employed by the company Adobe Systems Inc. The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

## Abbreviations and Notations

The following abbreviations and notations are used in this manuscript:

ITGs	Interval temporal graphs
CSGs	Contact sequence (temporal) graphs
TRG	Time-Respecting Graph
OSes	Ordered Sequence of Edges
G(V,E)	A graph G with V vertices and E edges
$\lambda$	Denotes the length or travel time of an edge in the graph

## References

- Scheideler, C. Models and Techniques for Communication in Dynamic Networks. In *Annual Symposium on Theoretical Aspects of Computer Science*; Springer: Berlin/Heidelberg, Germany, 2002; Volume 2285, pp. 27–49.
- Stojmenović, I. Location Updates for Efficient Routing in Ad Hoc Networks. In *Handbook of Wireless Networks and Mobile Computing*; John Wiley & Sons, Ltd.: Hoboken, NJ, USA, 2002; Chapter 21, pp. 451–471. [[CrossRef](#)]
- Holme, P.; Saramäki, J. Temporal networks. *Phys. Rep.* **2012**, *519*, 97–125. [[CrossRef](#)]
- Michail, O. An Introduction to Temporal Graphs: An Algorithmic Perspective. *arXiv* **2015**, arXiv:1503.00278.
- Santoro, N.; Quattrociocchi, W.; Flocchini, P.; Casteigts, A.; Amblard, F. Time-Varying Graphs and Social Network Analysis: Temporal Indicators and Metrics. *arXiv* **2011**, arXiv:1102.0629.
- Kuhn, F.; Oshman, R. Dynamic Networks: Models and Algorithms. *SIGACT News* **2011**, *42*, 82–96. [[CrossRef](#)]
- Bhadra, S.; Ferreira, A. Computing multicast trees in dynamic networks and the complexity of connected components in evolving graphs. *J. Internet Serv. Appl.* **2012**, *3*, 269–275. [[CrossRef](#)]
- Rossi, E.; Chamberlain, B.; Frasca, F.; Eynard, D.; Monti, F.; Bronstein, M. Temporal Graph Networks for Deep Learning on Dynamic Graphs. *arXiv* **2020**, arXiv:2006.10637. [[CrossRef](#)]
- Guo, F.; Zhang, D.; Dong, Y.; Guo, Z. Urban link travel speed dataset from a megacity road network. *Sci. Data* **2019**, *6*, 61. [[CrossRef](#)] [[PubMed](#)]
- Holme, P. Temporal network structures controlling disease spreading. *Phys. Rev. E* **2016**, *94*, 022305. [[CrossRef](#)] [[PubMed](#)]



11. Holme, P.; Saramäki, J. Temporal networks as a modeling framework. In *Temporal Networks*; Springer: Berlin/Heidelberg, Germany, 2021. [[CrossRef](#)]
12. Bearman, P.S.; Moody, J.; Stovel, K. Chains of Affection: The Structure of Adolescent Romantic and Sexual Networks. *Am. J. Sociol.* **2004**, *110*, 44–91. [[CrossRef](#)]
13. Stehlé, J.; Barrat, A.; Bianconi, G. Dynamical and bursty interactions in social networks. *Phys. Rev. E* **2010**, *81*, 035101. [[CrossRef](#)] [[PubMed](#)]
14. Smith, J.E. A Temporal Neural Network Architecture for Online Learning. *arXiv* **2020**, arXiv:2011.13844. [[CrossRef](#)]
15. Wu, H.; Cheng, J.; Ke, Y.; Huang, S.; Huang, Y.; Wu, H. Efficient Algorithms for Temporal Path Computation. *IEEE TKDE* **2016**, *28*, 2927–2942. [[CrossRef](#)]
16. Bui-Xuan, B.M.; Ferreira, A.; Jarry, A. Evolving graphs and least cost journeys in dynamic networks. In *WiOpt'03: Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks*; Sophia Antipolis: Valbonne, France, 2003; 10p.
17. Bentert, M.; Himmel, A.S.; Nichterlein, A.; Niedermeier, R. Efficient computation of optimal temporal walks under waiting-time constraints. *Appl. Netw. Sci.* **2020**, *5*, 73. [[CrossRef](#)]
18. Jain, A.; Sahni, S.K. Algorithms for optimal min hop and foremost paths in interval temporal graphs. *Appl. Netw. Sci.* **2022**, *7*, 60. [[CrossRef](#)]
19. Jain, A.; Sahni, S. Foremost Walks and Paths in Interval Temporal Graphs. *Algorithms* **2022**, *15*, 361. [[CrossRef](#)]
20. Gheibi, S.; Banerjee, T.; Ranka, S.; Sahni, S. An Effective Data Structure for Contact Sequence Temporal Graphs. In Proceedings of the 2021 IEEE Symposium on Computers and Communications (ISCC), Athens, Greece, 5–8 September 2021; pp. 1–8. [[CrossRef](#)]
21. Jain, A.; Sahni, S. Optimal Walks in Contact Sequence Temporal Graphs with No Zero Duration Cycle. In Proceedings of the 2023 IEEE Symposium on Computers and Communications (ISCC), Gammarth, Tunisia, 9–12 July 2023; pp. 392–398. [[CrossRef](#)]
22. Kunegis, J. KONECT: The Koblenz Network Collection. In Proceedings of the 22nd International Conference on World Wide Web (WWW'13 Companion), Rio de Janeiro, Brazil, 13–17 May 2013; pp. 1343–1350. [[CrossRef](#)]

**Disclaimer/Publisher's Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.