Jain and Sahni *Applied Network Science* (2022) 7:60
https://doi.org/10.1007/s41109-022-00499-3

Applied Network Science

RESEARCH Open Access

Algorithms for optimal min hop and foremost paths in interval temporal graphs

Anuj Jain[1,2*] and Sartaj K. Sahni[1]

This work is an extension of our paper ***Min Hop and Foremost Paths in Interval Temporal Graphs*** that appeared in ISCC 2021

*Correspondence: jainanuj99@gmail.com
[1] CISE Department, University of Florida, Gainesville, USA
[2] Engineering Department, Adobe Systems Inc., Lehi, USA

Abstract
Path problems are fundamental to the study of graphs. Temporal graphs are graphs in which the edges connecting the vertices change with time. Min hop paths problem in a temporal graph is the problem of finding time respecting paths from source vertex to every destination vertex such that the path goes through minimum number of edges. Foremost paths problem in a temporal graph requires to find time respecting paths that arrive at the destination vertices at earliest possible time. In this paper we present algorithms to find min hop paths and foremost paths in interval temporal graphs. These algorithms are benchmarked against the fastest algorithms known for foremost and min-hop paths by Wu et al. (IEEE Trans Knowl Data Eng 28(11):2927–2942, 2016a. https://doi.org/10.1109/TKDE.2016.2594065) that work on contact sequence temporal graph model. On the available test data, our foremost path algorithm provides a speedup of up to 1800 over the fastest algorithm for contact sequence graphs; the speedup for our min-hop algorithm is up to 6700. We also demonstrate that interval temporal graph model on which our algorithms work represents a superset of contact sequence temporal graphs. We show that path problems exist that are NP-hard in interval temporal graph model but polynomial in the contact sequence temporal graph model in terms of the number of vertices and edges in the input graph. This is due to the fact that the contact sequence graph model may require much larger number of edges than the corresponding interval temporal graph to represent a given temporal graph.

Keywords: Interval temporal graphs, Contact sequence temporal graphs, Foremost path, Min-hop path, NP-hard

Introduction

Temporal graphs are graphs in which the edges have time stamps. For example, in a temporal graph in which the vertices represent airports, a flight from (say) New York to Chicago that departs New York at 2 pm and has a duration of 3 h (i.e., it arrives at Chicago at 5 pm) could be represented by a directed edge from New York to Chicago with the label (2 pm, 3 h). Here, 2 pm is the time stamp and 3 h is the traditional edge weight. In a temporal graph that represents a road network, the vertices would represent road intersections; a directed edge from A to B would represent the road segment from A to B. This segment could be labeled by a sequence of tuples of the form $([t1 - t2], \lambda)$ where $t1$ and $t2$ are time stamps with the interpretation that if one departs A at a time t, $t1 \leq t \leq t2$,

© The Author(s) 2022. **Open Access** This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

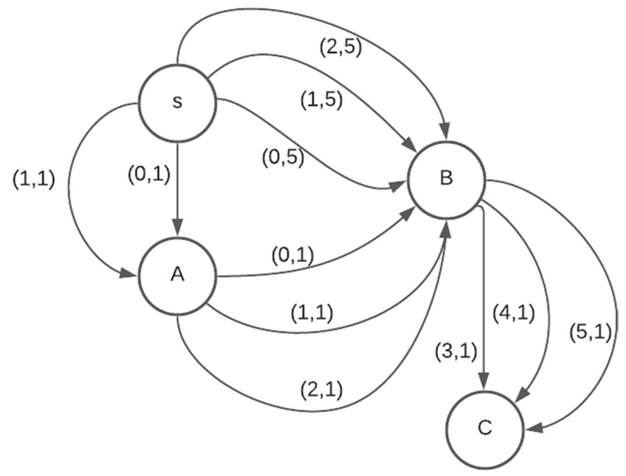

Fig. 1 Contact sequence temporal graph

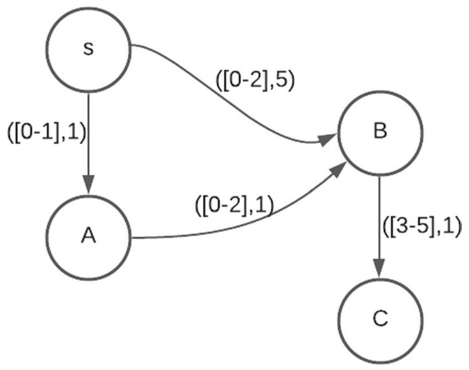

Fig. 2 Interval temporal graph

one will reach B at time $t + \lambda$ ($[t1 - t2]$) denotes the permissible departure interval from A). The sequence of triples would model different congestion intervals as well as intervals in which the road segment is closed for maintenance. In a contact sequence temporal graph, the edges are labeled as in the airport application and in an interval temporal graph, they are labeled as in the road network application. Figures 1 and 2 are examples of contact sequence and interval temporal graphs, respectively.

When time is discrete, every contact sequence temporal graph has an equivalent interval temporal graph and vice versa. Temporal graphs, which are also known as evolving graphs, dynamic graphs, and time varying graphs, have been used to study the spread of viral diseases, study information dissemination by means of physical/virtual contact between people, understanding the behavior on online social networks, modeling data transmission in phone networks, modeling traffic flow in road networks, and studying biological networks at the molecular level, for example Scheideler (2002), Stojmenović (2002), Holme and Saramäki (2012), Michail (2015), Santoro et al. (2011), Kuhn and Oshman (2011) and Bhadra and Ferreira (2012).

Temporal graph exploration and reachability problems are studied in Casteigts et al. (2020), Erlebach et al. (2021) and Michail and Spirakis (2016). Casteigts et al. (2020) investigates temporal reachability and temporal spanners in *random simple temporal* *graphs*. Such graphs are mathematically equivalent to random edge-ordered graphs. In Erlebach et al. (2021) authors consider the *TEXP* problem or the temporal graph exploration problem. This problem is defined as finding a temporal walk that starts at a given start vertex, visits all vertices of the graph and has smallest arrival time. Michail and Spirakis (2016) studies the traveling salesman problem for temporal graphs. They present approximation algorithms to find min-cost *TSP* tour that visits every vertex in a temporal graph. Temporal versions of other combinatorial optimization problems are also studied in this work.

Path problems on temporal graphs are studied in Wu et al. (2016a), Bui-Xuan et al. (2003), Bentert et al. (2020) and Guo et al. (2019). While Bui-Xuan et al. (2003) focuses on interval temporal graphs, Wu et al. (2016a) and Bentert et al. (2020) use the contact sequence model. Wu et al. (2016a) consider two representations of a contact sequence graph: a time ordered sequence of edges and a graph representation and demonstrate that on the datasets used by them, the studied path problems can be solved faster using the time ordered sequence of edges representation than either the graph representation proposed by them or the interval temporal graph representation used by Bui-Xuan et al. (2003). Bentert et al. (2020) consider extensions of the path problems studied in Wu et al. (2016a) and Bui-Xuan et al. (2003). This extension requires a specified minimum and maximum wait (stay) time at intermediate vertices. They also permit going through the same vertex multiple times. Hence, they consider walks from a source vertex to destination vertices while Wu et al. (2016a) and Bui-Xuan et al. (2003) are limited to paths. The algorithms of Bui-Xuan et al. (2003) are limited to the case when all triples on an edge have the same travel time λ. The path algorithms of Wu et al. (2016a) have been incorporated into the temporal graph library Tink (Lightenberg et al. 2018).

Our main contributions in this paper are:

1. We demonstrate the existence of path problems that are NP-hard for interval temporal graphs but polynomially solvable for contact sequence temporal graphs.
2. The algorithm of Bui-Xuan et al. (2003) for foremost paths in an interval temporal graph is extended to work when the triples on an edge may have different λ values.
3. We propose a different data structure for interval temporal graphs than proposed in Bui-Xuan et al. (2003). Using this data structure, our extended foremost path algorithm is faster than that of Wu et al. (2016a) on about half of the datasets used in Wu et al. (2016a); the maximum speedup obtained relative to Wu et al. (2016a) is 1800. On all synthetic datasets generated by us, we are faster with a maximum speedup of 3.77. We note that using the data structure in Bui-Xuan et al. (2003) and Wu et al. (2016a) report that their algorithm is faster on all datasets and achieves a speedup of up to 22 over the algorithm in Bui-Xuan et al. (2003).
4. We develop an algorithm for min-hop paths on interval temporal graphs that is faster than that of Bui-Xuan et al. (2003) and unlike the algorithm of Bui-Xuan et al. (2003) works even when the triples on an edge have different λ values. Our algorithm obtains a speedup of up to 31,000 relative to the min-hop algorithm of Bui-Xuan et al. (2003) on the datasets of Wu et al. (2016a).
5. Our min-hop algorithm is faster than that of Wu et al. (2016a) on all but 2 of the datasets used in Wu et al. (2016a) and all synthetic datasets generated by us. We

obtain a speedup of up to 6700 on the datasets of Wu et al. (2016a). On synthetic datasets, a speedup of up to 22.7 is achieved.

The roadmap of this paper is as follows. In "Preliminaries" section we provide formal definitions for contact sequence and interval temporal graphs, define path problems on temporal graphs, give the data structure we use for interval temporal graphs, and define a function used by our algorithms. In "NP-hard interval temporal graph path problems" section we demonstrate path problems that are NP-hard for interval temporal graphs but polynomial for contact sequence graphs. In "Redundant intervals" section, we introduce the notion of a redundant interval and show that the elimination of these redundant intervals does not change the foremost and min-hop paths in a temporal graph. Our extension of the foremost path algorithm of Bui-Xuan et al. (2003) to the case when intervals on an edge may have different λ values is described in "Foremost paths in interval temporal graphs" section and our algorithm for min-hop paths in interval temporal graphs is developed "Min-hop paths in interval temporal graphs" section. Experimental results are presented in "Experimental results" section and we conclude in "Conclusion" section.

Preliminaries

Definitions

Definition 1 (*Contact sequence temporal graphs*) In a *contact sequence temporal graph* $G=(V,E)$ each edge $e \in E$ is represented by a tuple (u, v, t, λ), where t is the permissible departure time for travel from u to v using the edge e and λ is the amount of time it takes to travel on edge e from u to v when one starts at time t. So, u is reached at time $t+\lambda$. If there are multiple time instances when departures from u to v are permissible, there will be multiple such temporal edges between u and v represented as a series of temporal edges $[(u, v, t_1, \lambda_1); (u, v, t_2, \lambda_2) \ldots; (u, v, t_n, \lambda_n)]$. The number of temporal edges between vertices u and v, which is also the number of distinct permissible departure times from u to v, gives 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 $(u, v, intvls)$. This tuple represents a connection from u to v. *intvls* is a time ordered sequence of tuples $[(s_1, c_1, \lambda_1); (s_2, c_2, \lambda_2); \ldots; (s_n, c_n, \lambda_n)]$. The ith interval starts at time s_i and closes (ends) at time c_i; λ_i is the time it takes to traverse the edge when travel departs u at a time t such that $[s_i \leq t \leq c_i]$ (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.

We note that in the interval temporal graph model used by Bui-Xuan et al. (2003), all intervals associated with an edge (u, v) have the same λ value. Further, c_i gives the time by which travel on (u, v) must finish rather than the last permissible departure time. So, in the model of Bui-Xuan et al. (2003), the permissible departure times for u defined by the ith interval are $s_i, \cdots, c_i - \lambda_i$.

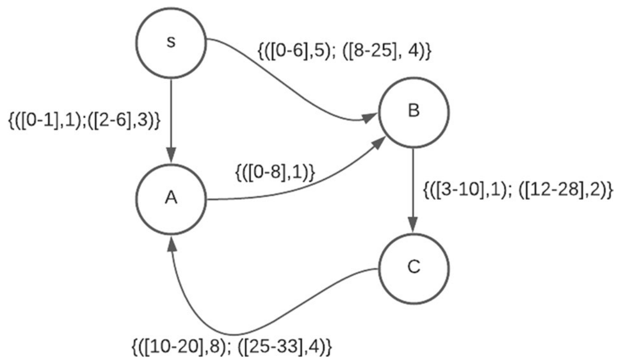

Fig. 3 Example interval temporal graph

A *path* (equivalently, valid path, temporal path or time respecting path) in a temporal graph is an alternating sequence of vertices and departure times $u_1, t_1, u_2, t_2, \ldots, u_k$ where t_i is a permissible departure time from u_i to u_{i+1} where $1 \leq i < k$, and $(t_i + \lambda_i) \leq t_{i+1}$, $t_i + \lambda_i$ is the arrival time at u_{i+1} when departing u_i at t_i using the connection (u_i, u_{i+1}). For this path, u_1 is the *source* vertex and u_k the *destination*. $P_1 = S, 0, B, 5, C$ is a path from S to C in the temporal graph of Fig. 2. This path leaves S at 0 and arrives at B at time 5. It then leaves B immediately at time 5 and arrives at C at time 6. P_1 is a 2-hop path from S to C with a first hop to B and then a second hop to C. $P_2 = S, 0, A, 1, B, 3, C$ is another valid path from S to C. This path also leaves S at time 0. It gets to A at time 1 and departs immediately for vertex B where it arrives at time 2. At B, it waits for 1 unit until time 3 and departs for C getting there at time 4. P_2 is a 3-hop path from S to C.

We are interested in paths in a temporal graph that start at a vertex u at a time $\geq t_{start}$ and end at another vertex v. Let $S(u, v)$ comprise all valid paths from u to v that depart u at a time $\geq t_{start}$. A *foremost path* is a path in $S(u, v)$ that gets to v at the earliest time; a *min-hop* path is a path in $S(u, v)$ that has the fewest number of hops; a *fastest path* is a path in $S(u, v)$ for which (arrival time at v—departure time from u) is minimum; and a *shortest path* is a path in $S(u, v)$ that minimizes the sum of the λs on the path.[1]

Bui-Xuan et al. (2003) develop polynomial time algorithms for foremost, min-hop, and fastest paths in interval temporal graphs. Wu et al. (2016a) do this for contact sequence graphs.[2] They also develop algorithms for reverse-foremost paths (paths with the latest departure time and terminating at a specified vertex).

Data structures

Bui-Xuan et al. (2003) use linked adjacency lists to represent an interval temporal graph. We, instead, use array adjacency lists (Sahni 2004). For example, the interval temporal graph of Fig. 3 is represented by the array adjacency list of Fig. 4. The data structure comprises a (say) C++ vector with one slot for each vertex in the graph. This is the vertical vector in the figure. Slot for any vertex u itself contains a vector of vertices adjacent

[1] Bui-Xuan et al. (2003) use the term shortest path to mean a min-hop path.

[2] While Wu et al. (2016a) does not explicitly consider min-hop paths, their shortest path algorithm is easily modified to find min-hop paths.

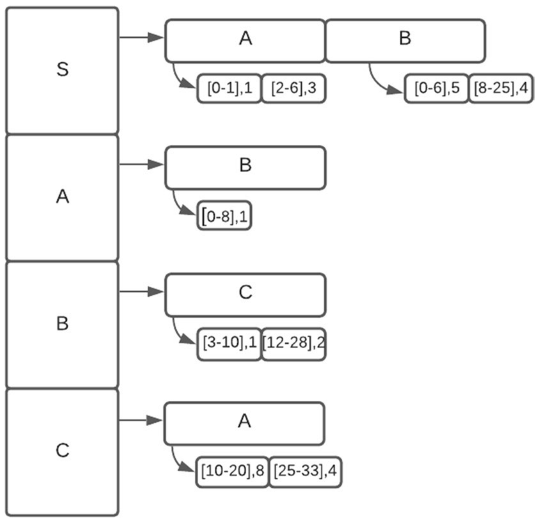

Fig. 4 Data structure representing Interval temporal graph of Fig. 3

from u. For example, the slot for S in the vertical vector has the adjacent vertices vector ($A.B$). Associated with each adjacent vertex v from u, there is a vector of time ordered tuples for the edge (u, v). In Fig. 4, ([0–1],1), ([2–6],3) is the time ordered vector for the edge (S, A).

Wu et al. (2016a) consider two data structures for contact sequence graphs. In the first of these, the graph is simply represented as a sorted sequence of edges (tuples) of the form $e = (u, v, t, \lambda)$; this sequence is in non-decreasing order of t. The second representation is a graph that is quite different from that of Fig. 4 and which their experiments show to be inferior for all path problems studied by them except the fastest path problem where the two representations are competitive. Since we do not consider the fastest path problem here, we do not describe their graph representation. However, we mention that their graph representation has more edges than their sorted sequence representation. When time is discrete, every interval temporal graph can be transformed into a contact-sequence graph that has the same foremost, min-hop, shortest, fastest, and reverse-fastest paths. In this transformation, we replace each edge in the interval temporal graph by as many contact sequence edges as the number of permissible departure times for edge intervals. For example, if time is an integer, then the the connection (S, A) in Fig. 3 with the interval sequence ([0–1],1), ([2–6],3) gets replaced by the tuples (S, A, 0, 1), (S, A, 1, 1), (S, A, 2, 3), (S, A, 3, 3), (S, A, 4, 3), (S, A, 5, 3), (S, A, 6, 3). As is evident, this transformation preserves valid paths but has the potential for explosive growth in the number of edges and consequently in the memory needed. For example, the integer interval [1, 100,000] would result in 100,000 contact sequence edges. As we shall see in the next section, this explosive growth in instance size can result in a path problem being NP-hard in the interval model but polynomial in the contact sequence model.

We note also that every contact sequence temporal graph may be transformed into an equivalent interval temporal graph by coalescing the multiple edges that connect a pair of vertices (u, v) into a single edge with an appropriate time ordered sequence of intervals; each interval's start time is $\leq$ its close time.

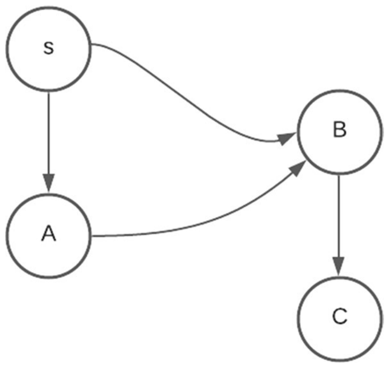

Fig. 5 Underlying static graph for temporal graphs of Figs. 1 and 2

The function *next*

Bui-Xuan et al. (2003) define a function that given a time t and vertices u and v finds the earliest permissible departure time greater than or equal to t on the edge (u, v). The function is denoted as $f((u, v), t)$. This function simply does a binary search on the intervals associated with the edge (u, v). It runs in $O(log(k))$ time, where k is the number of intervals associated with the edge. This function is used by Bui-Xuan et al. (2003) in their path algorithms and also used by us in our algorithms. We call this function *next.*

NP-hard interval temporal graph path problems

Several problems are known to be NP-hard for contact sequence temporal graphs. For example, Bhadra and Ferreira (2003) show that computing several types of strongly connected components is NP-hard; Casteigts et al. (2019) show that determining the existence of a no-wait path between[3] two vertices is NP-hard; and Zschoche et al. (2018) show that computing several types of separators is NP-hard. Additional complexity results for contact sequence temporal graphs appear in Casteigts et al. (2019). Since contact sequence temporal graphs can be modeled by interval temporal graphs with at most a constant factor increase in the instance size (see "Data structures" section), every problem that is NP-hard for the contact sequence model remains NP-hard in the interval model. However, the reverse may not be true as the transformation from the interval model to the contact sequence model entails a possible explosion in the instance size. In this section we demonstrate path problems that are NP-hard in the interval model but polynomially solvable in the contact sequence model.

No wait acyclic path problem (NAPP)

The underlying *static graph* for any contact sequence temporal graph is the graph that results when each edge (u, v, t, λ) is replaced by the edge (u, v) and then multiple occurrences of the same edge (u, v) are replaced by a single edge (u, v). For an interval temporal graph, its underlying static graph is obtained by replacing each edge $(u, v, intvls)$ by the edge (u, v). Figure 5 shows the underlying static graphs for the temporal graphs of Figs. 1 and 2.

[3] In a no-wait path, the arrival and departure times at each intermediate vertex are the same.

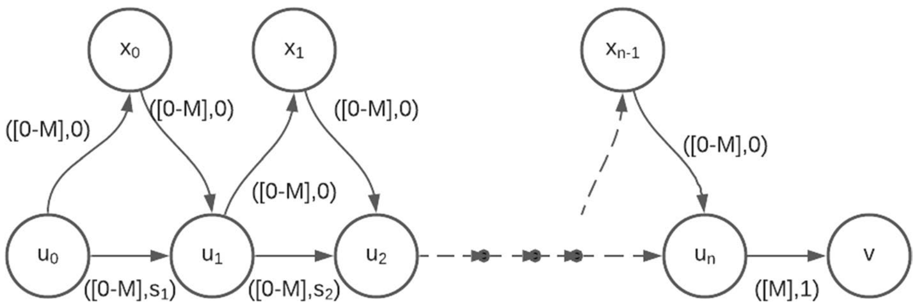

Fig. 6 Interval temporal graph demonstrating NP-hard algorithms

The *no-wait acyclic path problem* (NAPP) is to find a no-wait (time respecting) path from a vertex u to a vertex v in a temporal graph whose underlying static graph is acyclic. As noted above, Casteigts et al. (2019) have shown that determining the existence of such a path is NP-hard for contact sequence graphs and hence for interval temporal graphs when the graphs are not limited to be acyclic. We show below that NAPP is NP-hard for the interval model but polynomially solvable for the contact sequence model (for acyclic graphs).

NAPP is NP-hard

Theorem 1 *NAPP is NP-hard for the interval model but polynomially solvable for the contact sequence model.*

Proof For the NP-hard proof, we use the sum of subsets problem that is known to be NP-hard. In this problem, we are given n natural numbers $S = \{s_1, s_2, \cdots, s_n\}$ and another natural number M. We are to determine a subset of S that sums to M. For any instance of the sum of subsets problem, we can construct, in polynomial time, the acyclic interval temporal graph shown in Fig. 6. For all edges other than (u_n, v) the permissible departure times are from 0 through M (i.e., their associated interval is $[0–M]$) and the edge (u_n, v) has the single permissible departure time M (equivalently, its associated interval is $[M–M]$ or simply $[M]$). The travel time (λ) for edge (u_i, u_{i+1}) is s_i, that for (u_n, v) is 1, and that for the remaining edges is 0. It is easy to see that the underlying static graph is acyclic and that for every subset of S, there is a no-wait path from u_0 to u_n that arrives at u_n at a time equal to the sum of the s_is in that subset. Further, all no-wait paths from u_0 to v must get to u_n at time M. Hence, there is a no-wait path u_0 to v iff there is a subset of S that sums to M; this path gets to v at time $M + 1$. Hence, NAPP is NP-hard.

The NAPP may be solved in polynomial time for contact sequence temporal graphs whose underlying static graph is acyclic by considering vertices in topological order. Suppose that the start vertex of the desired no-wait path is u and the destination vertex is v. For each vertex w, we maintain a list of possible arrival times of no-wait paths from u. Initially, this list is empty for all vertices other than u. The initial list for u is $\{0\}$. We note that the size of the list for vertex w cannot exceed its in-degree as t is a possible arrival time at w only if there is an edge $(x, w, t1, \lambda)$ such that $t = t1 + \lambda$. Hence, when the vertices are examined in topological order, the list for the vertex being currently

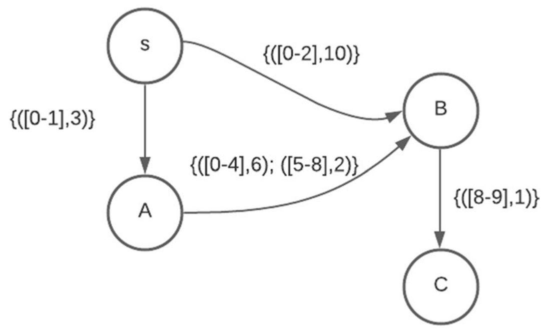

Fig. 7 Interval graph with some slow intervals

examined may be computed from the lists of its incoming neighbors in polynomial time. When done, if the list of v is empty, there is no no-wait path from u to v; if this list is not empty, a no-wait path may be constructed in polynomial time using a traceback from entries in the list for v. $\square$

We note that the above proof is readily modified to show that finding foremost, fastest, min-hop, and shortest no-wait paths in interval temporal graphs with an acyclic underlying static graph are NP-hard while these problems are polynomial for contact sequence temporal graphs whose underlying static graph is acyclic. The above proof also shows that finding no-wait walks (in a walk, a vertex may be visited more than once) in interval temporal graphs in NP-hard.[4] For contact sequence graphs, Bentert et al. (2020) have developed polynomial time algorithms to find optimal walks using various optimization functions. Further, the construction used in the proof is easily modified so that every edge has a λ value > 0.

Redundant intervals

It is possible for an interval (or portion of an interval) on an edge of an interval temporal graph to be redundant in that the departure times in this interval may never be used in an optimal path (e.g., a foremost path). For example, if the travel from u to v (on a connection (u, v) represented in an edge e) starts at a time t where $t \in [s_i, c_i]$, for some interval $i = (s_i, c_i, \lambda_i)$ on e, then the arrival time at v is $t + \lambda_i$. If $(c_i + \lambda_i) \geq (s_{i+1} + \lambda_{i+1})$ where $(i + 1) = (s_{i+1}, c_{i+1}, \lambda_{i+1})$ is the next interval on the edge e, then it is possible that departing from u to v at a time in the interval i reaches v after a departure that starts in the later interval $(i + 1)$.

For path problems such as finding foremost, min-hop, shortest, and fastest paths, the portion of the interval i that is beyond time t may be removed as an equal or better path can always be found by starting at the start time s_{i+1} of the next interval as opposed to starting at or after time t in the interval i. The portion of interval i that may be so removed is called *redundant*. All of the path problems studied in Wu et al. (2016a) and Bui-Xuan et al. (2003) never need to use these redundant intervals.

[4] Note that in the graph of Fig. 6, every walk is also a path.

An example of a graph with some redundant intervals is shown in Fig. 7. In this graph, a foremost path from S to C is, $P_f = (S, 0, A, 5, B, 8, C)$ arriving at node C at time 9. Notice that, even though we arrive at node A at time 3 using arc (S, A) and the first interval on the arc (A, B) is still open at time 3, there is no benefit of using this interval with the travel time of 6 as opposed to waiting at A for 2 time units and then using the next interval at time 5 with a travel time of 2, that would get us to B, 2 time units sooner at 7.

It is important to note that there are some path problems in which intervals defined to be redundant are actually useful and so should be retained when solving these path problems as above. For example, in Fig. 7, if the problem was to find a minimum wait path from S to C, the optimal path in this case would benefit by departing from the node A as soon as possible, instead of waiting for the next faster interval. In this case, the minimum wait path would be $P_{mw} = (S, 0, A, 3, B, 9, C)$ arriving at node C at time 10 with a wait time of 0.

For the path problems studied in this paper (min-hop paths and foremost paths) as well as those studied in Wu et al. (2016a) and Bui-Xuan et al. (2003) redundant intervals may be safely removed from the interval temporal graph representation. Therefore, we assume that our interval temporal graphs are free of redundant intervals.

The algorithm to remove redundant intervals from interval temporal graphs is described below.

```
• For every edge
    • Examine the intervals in descending order starting from interval$(n-1)$
    • For each interval$(i)$ that is examined
        * if $(c_i + \lambda_i > s_{i+1} + \lambda_{i+1}) \Rightarrow c'_i = s_{i+1} + \lambda_{i+1} - \lambda_i$
        * if $(c'_i \geq s_i) \Rightarrow interval(i) \leftarrow (s_i, c'_i)$
        * else eliminate $interval(i)$
```

Foremost paths in interval temporal graphs

Methodology

As noted earlier, Bui-Xuan et al. (2003) develop an algorithm to find foremost paths from a source vertex s to all remaining vertices in an interval temporal graph; the foremost paths are constrained to depart s at a time greater than or equal to t_{start}. This algorithm, however, assumes that all intervals of an edge have the same λ value (the λ may be different for different edges). A minor extension of their algorithm enables it to work for temporal interval graphs in which the λ value may change from interval of an edge to the next interval for the same edge.

Algorithmic strategy used to find foremost paths is similar to that used in Dijkstra's shortest path algorithm. We begin with knowledge of the foremost paths from start vertex to itself then generate foremost paths to remaining vertices in order of the arrival times of these foremost paths. When a new foremost path is found, its one edge extensions are considered much in the same way as in Dijkstra's algorithm and the projected arrival time of the foremost paths to neighbor vertices is updated as necessary. A min priority queue of projected foremost paths is maintained and in each round, the foremost path with least arrival time is considered.

Algorithm details

Algorithm 1 describes this extended algorithm that assumes the interval temporal graph has been pre-processed to remove redundant intervals.

Some of the key variables used in Algorithm 1 are:

1. t_{EAD}—array that stores the earliest known arrival time at every vertex u. When the algorithm terminates, every index of this array contains the foremost time of arrival at the corresponding vertex u.
2. *pred*—array that contains the predecessor node for every vertex u in the foremost path from the source vertex s. The *pred* array can be used to construct the foremost path from s to every vertex u.
3. PQ—priority queue whose elements are pairs of the form $(t_{EAD}[u], u)$ and the first element of each pair is the priority key. Our implementation uses a min-heap.

Algorithm 1 Foremost Path Algorithm

```
1:  Initialize $t_{EAD}[s] \leftarrow t_{start}$; $\forall v \in V, v \neq s, t_{EAD}[v] \leftarrow \infty$; $\forall v \in V$, mark $v$ as opened.
2:  $u \leftarrow s$
3:  do
4:      Mark $u$ as closed.
5:      for each open neighbor $v$ of $u$ do
6:          compute $(t, i) = f((u, v), t_{EAD}[u])$
7:          if $t + \lambda_i < t_{EAD}[v]$ then
8:              Update $t_{EAD}[v] \leftarrow t + \lambda_i$
9:              Update $pred[v] \leftarrow u$
10:             $PQ.insert(t_{EAD}[v], v)$
11:         end if
12:     end for
13:     do
14:         if $PQ.hasItems()$ then
15:             $(t_{EAD}[u], u) \leftarrow PQ.removeMin()$
16:         else
17:             EXIT
18:         end if
19:     while $u$ is closed
20: while true
```

Note that step 10 of our algorithm may result in the priority queue containing many entries for any given vertex v. However, from all the PQ entries for v, the one with the least arrival time at v will be the first one to be removed from PQ in step 15, and marked as closed in step 4. Therefore, when an element corresponding to a closed vertex is removed from the priority queue, it can simply be discarded as in the loop from step 13 to step 19. Experimentally we found it is more efficient to simply keep multiple elements in the priority queue for the same vertex than to update the arrival times of vertices already in the priority queue so as to ensure all elements correspond to different vertices.

The correctness proof for Algorithm 1 is the same as that given in Bui-Xuan et al. (2003).

Min-hop paths in interval temporal graphs

The algorithms of Bui-Xuan et al. (2003) and Wu et al. (2016a)

In the min-hop path problem, we are to find paths from a source vertex s to all remaining vertices in the temporal graph. Each such path must depart s at or after a specified time t_{start} and must use the smallest number of hops (edges) in getting to its destination from among all paths that start at or after t_{start}.

Bui-Xuan et al. (2003) use the term *shortest path* instead of min-hop path for this problem. The complexity of their algorithm[5] is $O(NM_{itg} \log \delta)$, where N is the number of vertices, M_{itg} is the number of edges, and δ is the maximum number of intervals on an edge.

While Wu et al. (2016a) do not consider the min-hop problem explicitly, they present algorithms to find the length of shortest paths from a source vertex to all other vertices in a contact sequence temporal graph. This algorithm is easily modified to compute the number of hops on min-hop paths from the source vertex by simply assuming that the distance accumulated on every edge traversal is just 1 instead of the λ for the edge. The complexity of Wu's algorithm that represents a contact sequence temporal graph as a time ordered sequence of edges is $O(N + M_{csg} \log d_{max})$, where d_{max} is the maximum in-degree of a vertex (Wu et al. 2016b). Note that the total number of edges M_{itg} in the interval temporal graph is different from M_{csg} in contact sequence temporal graph representation; often, M_{csg} is much larger than M_{itg}.

Our algorithm

Methodology

We now develop a faster min-hop algorithm for Interval temporal graphs than that of Bui-Xuan et al. (2003). Unlike the algorithm of Bui-Xuan et al. (2003) our algorithm does not require the λs for all intervals associated with an edge to be the same. It does, however, assume that redundant intervals have been removed. Our algorithm, Algorithm 2, is a greedy algorithm that first identifies all vertices that can be reached from s is one hop, then those that can be reached in 2 hops and so on. The algorithm terminates when one of the following conditions is met:

1. All V vertices have been reached.
2. No new vertices are discovered in a given hop or the earliest arrival time for none of the previously reached vertex decreases. This means that no new vertices will be discovered in future rounds.
3. The hop count is $V - 1$. Since the maximum hop-count for paths in a graph with V vertices is $V - 1$, hop counts larger than this need not be considered.

Algorithm details

Some of the data structures used in our algorithm are described below.

[5] There is a bug in the algorithm presented by Xuan et al. In Step 2.c of Algorithm 3 in their paper (Bui-Xuan et al. 2003), the algorithm can potentially overwrite a path to v from the previous round with a new path computed in the current round, before the path to v from the previous round has been extended in the current round.

1. *incSt* is a structure that keeps track of vertices discovered in every hop. The fields in this structure are as follows:

   (a) *curVtxId*—is the current vertex.
   (b) *arrTm*—is the time of arrival at the current vertex.
   (c) *refPrvIncSt*—is reference to previous *incSt* that stores similar information about previous vertex on this path

2. *allHopPaths*—array of lists that stores list of vertices discovered at every hop. This array has at most H lists, where H is the maximum number of hops in min-hop paths from source vertex, s to any of the vertices $v \in V$. Every element of the list is an instance of the structure *incSt*.
3. t_{EKA}—array that stores an earliest known arrival time to every vertex v.
4. *MHP*—array used to retrieve the min hop paths to every vertex $v \in V$. The elements of the array are a tuple h, *refIncSt*, where h is the number of hops in the min-hop path to the vertex v and *refIncSt* is the reference to an instance of *incSt* in the list stored at index h of *allHopPaths*. This instance of *incSt* is used to trace back the min-hop path to vertex v from the source vertex s.

Algorithm 2 Minimum Hop Path Algorithm

```
1: Create $startSt \leftarrow (s, t_{start}, null)$ as an instance of $incSt$.
2: Initialize $t_{EKA}[s] \leftarrow t_{start}; \forall v \in V, v \neq s, t_{EKA}[v] \leftarrow \infty$; Initialize $MHP[s] \leftarrow (0, startSt); \forall v \in V, v \neq s, MHP[v] \leftarrow null$; Initialize $allHopPaths[0] \leftarrow \{startSt\}$
3: $hopCnt = 0; newVsInHop = 1; totVsRchd = 1$
4: while $(hopCnt < V - 1)$ and $(newVsInHop \geq 0)$ and $(totVsRchd < V)$ do
5:     $hopCnt++$
6:     $newVsInHop \leftarrow 0$
7:     for each $(refIncSt \in alHopPths[hopCnt - 1])$ do
8:         $vert = refIncSt.curVtxId$
9:         $t_{vertArr} = refIncSt.arrTm$
10:        for each $(nbr \in V[vert].nbrs)$ do
11:            $(depTm, intvlId) = f((vert, nbr), t_{vertArr})$
12:            if $depTm \geq \infty$ then
13:                $continue$                                  ▷ start next loop iteration
14:            end if
15:            $newArr_{nbr} = depTm + \lambda_{intvlId}$
16:            if $(newArr_{nbr} < t_{EKA}[nbr])$ then
17:                Create an instance of $incStruct$ as $nis$
18:                $nis \leftarrow (nbr, newArr_{nbr}, refIncSt)$
19:                Append $nis$ to list $alHopPths[hopCnt]$
20:                $newVsInHop++$
21:                if $t_{EKA}[nbr] \geq \infty$ then
22:                    $totVsRchd++$
23:                    $MHP[nbr] \leftarrow (hopCnt, nis)$
24:                end if
25:                $t_{EKA}[nbr] = newArr_{nbr}$
26:            end if
27:        end for
28:    end for
29: end while
```

Example

As an example, consider the interval temporal graph of Fig. 8. Let the source vertex be S and $t_{start} = 0$. In the first round ($hopCnt = 1$), the neighbors A, B, and C are identified

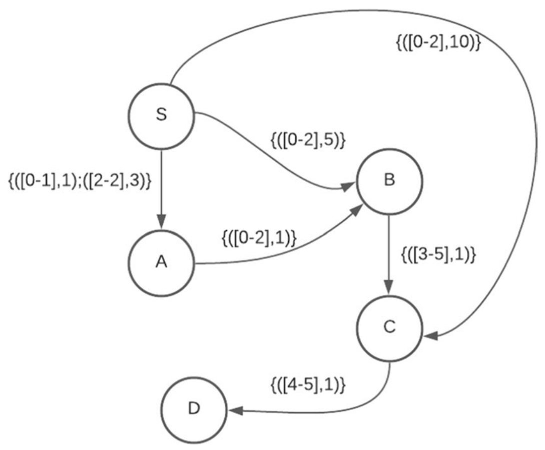

Fig. 8 Min hop paths in interval graph

as one-hop neighbors of S with one-hop path arrival times of 1, 5, and 10, respectively. In the next round ($hopCnt = 2$), these one-hop paths are expanded to two-hop paths to vertices B (S, A, B) and C (S, B, C). The arrival times of these paths are 2 and 6, respectively. Since these arrival times are earlier than the previous arrival times at B and C, these newly found two-hop paths may lead to new vertices. In the third round ($hopCnt = 3$), the earlier arriving 2-hop paths to B and C are expanded. While the 2-hop path to C cannot be expanded any further, the 2-hop path to B is expanded to get a 3-hop path to C that gets to C at 4. This path is expanded in the next round ($hopCnt = 4$) and the 4-hop path to D (S, A, B, C, D) is discovered. This path arrives at D at 5. The algorithm now terminates as $hopCnt = 4 = V - 1$ (note that coincidentally, $totVsRchd = V$ at this time in this example).

Correctness proof and complexity

Theorem 2 *Algorithm 2 finds min-hop paths from the source vertex, s to all other vertices $v \in V$ in the temporal graph $G = (V, E)$*

Proof We can prove this by induction. After k hops, assume:

1. We have found min-hop paths to all vertices reachable in ($\leq k$) hops.
2. We have found the k-hop foremost paths to all vertices reachable in k hops.
3. Any such foremost path that ends at time t at a vertex v, is also a minimum hop path from s to v arriving at or before time t.

For the next hop $k + 1$, any new paths that we discover are one-hop extensions of the foremost paths found in hop k, using the function *next* of “The function next” section. Such newly discovered paths are also the $k + 1$ hop foremost paths. This is because minimum extension of a foremost path is also a foremost path. Also notice that we find such paths to all the reachable vertices in $k + 1$ hops as we try to extend every foremost path that was reachable in k hops. Further, any vertex discovered for the first time in hop

$k + 1$ requires minimum $k + 1$ hops to get there, otherwise it should have been discovered in an earlier hop because of the hypothesis above.
It is easy to see that the base condition is true for hop 1, as starting from source vertex, s we find earliest arrival time to all its neighbors using the function $f((s, nbr), t_{start})$. Therefore, we find all vertices reachable in 1 hop and we also find the foremost time in which they can be reached, in 1 hop. □

The asymptotic complexity of Algorithm 2 is $O(NM \log \delta)$, which is the same as that of the min-hop algorithm of Bui-Xuan et al. (2003). Our algorithm is however simpler and works for interval temporal graphs in which the λs may be different for different intervals on the same edge while that of Bui-Xuan et al. (2003) requires all intervals on an edge to have the same λ. We demonstrate, in the next section, that despite the generality of our algorithm it is much faster than that of Bui-Xuan et al. (2003) even when all intervals on an edge have the same λ.

Experimental results

In this section, we assess the relative performance of our foremost path and min-hop path algorithms to that of the one-pass algorithms in Wu et al. (2016a) for these problems and the min-hop algorithm of Bui-Xuan et al. (2003). Our experimental platform was an Intel Core,i9-7900X CPU @ 3.30GHz processor with 64 GB RAM. The C++ codes for the one-pass contact sequence temporal graph algorithms was obtained from the authors of Wu et al. (2016a) and their code for the shortest paths problem modified to compute the number of hops in min-hop paths. All other algorithms were coded by us in C++. The codes were compiled using the $g++ver.7.5.0$ compiler with option O2. For test data, we used the datasets used in Wu et al. (2016a), which were also used in Bentert et al. (2020), as well as some synthetic datasets generated by us. As stated in Wu et al. (2016a), their algorithms do not work when some λs are 0 though they may be extended to work when this is the case. This limitation did not affect our experiments as none of the datasets used by us have a λ of 0.

The algorithm of Bui-Xuan et al. (2003) for the min-hop problem needed two changes so it could be compared with our algorithm and that of Wu et al. (2016a). The changes are as follows:

1. We fixed the bug in the min-hop algorithm of Bui-Xuan et al. (2003) that was mentioned earlier in "Min-hop paths in interval temporal graphs" section of this paper.
2. Algorithm 2 of Bui-Xuan et al. (2003) is repeatedly called by Algorithm 3 of Bui-Xuan et al. (2003), which is their min-hop algorithm. Algorithm 2 of Bui-Xuan et al. (2003) goes through all edges in the graph to find new minimum paths in a given hop and saves them in array variables e_{min} and t_{min} used by that algorithm. If in a given hop no new minimum paths are discovered, then they won't be discovered in any future hops either. Therefore, their min-hop algorithm can be terminated after the first such hop. Adding this early termination condition to their algorithm vastly improves their run time. For example, without the early termination condition, the min-hop algorithm of Bui-Xuan et al. (2003) took 943 s to find minimum hop paths in the *arxiv* dataset, but with the early termination condition added, it took only 0.17 s.

Table 1 Koblenz collection graph statistics

Dataset	$\|V\|$	$\|E_s\|$	Wu-edges	Activity
Epin	131.8K	840.8K	841.3K	1
Elec	7119	103.6K	103.6K	1
Fb	63.7K	817K	817K	1
Flickr	2302.9K	33,140K	33,140K	1
Growth	1870.7K	39,953K	39,953K	1
Youtube	3223K	9375K	9375K	1
Digg	30.3K	85.2K	87.6K	1.02
Slash	51K	130.3K	140.7K	1.07
Conflict	118K	2027.8K	2917.7K	1.43
Arxiv	28K	3148K	4596K	1.45
Wiki-en-edit	42,640K	255,709K	572,591K	2.23
Enron	87,274	320.1K	1148K	3.58
Delicious	4512K	81,988K	301,186K	3.67

Datasets

The 14 datasets described in Wu et al. (2016a) and also in Bentert et al. (2020) were downloaded by us from the Koblenz network (Kunegis 2013). The description of the datasets in Wu et al. (2016a) closely matches the following:

1. arXiv hep-ph—http://konect.cc/networks/ca-cit-HepPh
2. dblp—http://konect.cc/networks/dblp_coauthor
3. delicious—http://konect.cc/networks/delicious-ut
4. digg—http://konect.cc/networks/munmun_digg_reply
5. wikipedia elections—http://konect.cc/networks/elec
6. enron—http://konect.cc/networks/enron
7. epinions trust—http://konect.cc/networks/epinions
8. facebook—http://konect.cc/networks/facebook-wosn-links
9. flickr—http://konect.cc/networks/flickr-growth
10. slashdot—http://konect.cc/networks/slashdot-threads
11. wikipedia conflict—http://konect.cc/networks/wikiconflict
12. wikipedia edits—http://konect.cc/networks/edit-enwiki
13. wikipedia growth—http://konect.cc/networks/wikipedia-growth
14. youTube—http://konect.cc/networks/youtube-u-growth.

The *dblp* dataset downloaded from the link mentioned above had a few negative timestamps, so we discarded this dataset. The statistics for the remaining 13 datasets are given in. Table 1. In this table, $|V|$ is the number of vertices, $|E_s|$ is the number of edges in the underlying static graph, Wu-edges is the number of edges in the contact sequence temporal graph, and activity is the ratio Wu-edges/$|E_s|$. Note that the number of edges in the interval temporal graph is also $|E_s|$.

Wu et al. (2016a) also experimented against *flow* network from yahoo, but we did not experiment against that network, because the network is too large to fit into memory of one computer. Parallel or distributed implementation of these Algorithms are out of scope of this paper. Statistics of some of the Koblenz networks are different from those described in Wu et al. (2016a). This may be because the datasets have evolved/changed since the time they were used by the authors of Wu et al. (2016a). The statistics of the downloaded datasets are described in Table 1. We implemented the Foremost paths algorithm, described in Algorithm 1 and the Min-Hop paths algorithm described in Algorithm 2 on all these datasets except the dblp network for the reasons described above. The datasets have a wide range of sizes in terms of the number of vertices and edges. The temporal activity defined as the ratio of Wu-edges to the static edges, on these datasets is very low ranging from 1 or no activity to a maximum value of 3.67 on the *delicious* dataset. The travel times on each edge is assumed to be a constant value of 1 as the datasets do not specify the travel time. Also, choice of this travel time is same as that used for experimental purposes in Wu et al. (2016a) and Bentert et al. (2020). Algorithms of Wu et al. (2016a) do not work with the travel time of 0, whereas our Algorithms work for any travel time ≥ 0. Wu et al. (2016a) also used an additional much larger dataset called *flow*. This dataset, however, was used only to benchmark their parallel algorithms as it was too large to fit in the memory of the computer used to benchmark their serial codes. We do not use this very large dataset either. The downloaded Koblenz datasets do not have λ values associated with the edges. All λ values were set to 1 by us. The authors of Wu et al. (2016a) and Bentert et al. (2020) confirmed that they did the same for their experiments.

The activity in the 13 datasets of Table 1, which ranges from a low of about 1 to a high of about 3.67 is rather low. Further, all λ values are 1. So, we generated datasets with higher activity and variable λs synthetically by starting with the social network graphs of *youtube, flickr, livejournal* available at http://socialnetworks.mpi-sws.org/data-imc2007.html shared by the authors of Mislove et al. (2007). These graphs represent user-to-user interactions. We synthesized temporal graphs from these large social graphs by randomly adding temporal intervals to each static edge. To generate the temporal intervals we used three random variables (I, D, T) for assigning random values to three temporal parameters, namely the number of Intervals on a given edge, the Duration of each interval and the Travel time (λ) on each of the intervals. We assigned values to each of these three random variables using the normal distribution around a fixed mean value for each of the variables. For example, in the first synthetic graph, using a static *youtube* graph with 1.15M vertices and 4.9M static edges as a base graph, we added a random number of temporal intervals on each static edge (random variable I), using a normal distribution with a mean value of 4. We assigned a random duration to each of these intervals (random variable D) using a normal distribution with a mean value of 5 time units. We also assigned a random value for the travel time on each of these intervals (random variable T) using a normal distribution with a mean value of 3 time units. We generated the synthetic graphs from the other two social networks as well using the random variables (I, D, T) with normal distribution around the same mean values. Finally, we converted these synthetic interval temporal graphs to contact sequence temporal graphs as described in "Data structures" section. Table 2 shows the statistics for the 5 synthetic temporal graphs generated by us.

Table 3 gives the time (in seconds) required to read each dataset from disk as well as the disk memory required by each dataset. The columns labeled Wu are for the case when the dataset is stored as a contact sequence temporal graph and those labeled Ours are for the case when the interval temporal graph representation is

Table 2 Synthetic graphs statistics

Dataset	$\|V\|$	$\|E_s\|$	Wu-edges	Edge activity
Graphs with $\mu_I = 4, \mu_D = 5, \mu_T = 3$				
Youtube	1157.8K	4945K	105,039K	21.2
Flickr	1861K	22,613.9K	480,172K	21.24
Livejournal	5284K	77,402.6K	1,643,438K	21.3
Graphs with $\mu_I = 4, \mu_D = 8, \mu_T = 3$				
Youtube	1157.8K	4945K	159,103.7K	32.1
Flickr	1861K	22,613.9K	727,405.9K	32.1

Table 3 Reading times and sizes

Dataset	Reading time (in s)		Sizes in MBs	
	Wu	Ours	Wu	Ours
Koblenz collection				
Epin	0.20	0.35	19.6	29.8
Elec	0.041	0.060	2.3	3.6
Fb	0.18	0.30	15.4	21.7
Flickr	8.45	13.31	868	1299
Growth	9.53	16.38	1051	1571
Youtube	2.37	3.91	257	379
Digg	0.039	0.053	2	3.1
Slash	0.051	0.077	3.3	5
Conflict	0.67	1.05	74.1	103.3
Arxiv	0.94	1.32	83.8	103.6
Wiki-en-edit	143.87	169.72	15.52	17.81
Enron	0.28	0.30	25.8	29.7
Delicious	68.38	81.47	7346	8385
Synthetic Datasets with $\mu_I = 4, \mu_D = 5, \mu_T = 3$				
Youtube	20.94	3.59	1958	272.2
Flickr	95.26	16.36	9030	1248
Livejournal	344.13	56.16	33,838	4411.8
Synthetic Datasets with $\mu_I = 4, \mu_D = 8, \mu_T = 3$				
Youtube	32.58	3.53	2966	272.3
Flickr	148.03	16.19	13,682	1249

used. As expected, the reading time and the the disk space required when the contact sequence model is used are less than when the interval model is used when the edge activity is low; the reverse is the case when the edge activity is high. As an example, for the large synthetic flickr graph with an activity factor of 32.1, the reading time of the interval temporal graph is 16.19 s while that of the corresponding contact sequence graph is 148 s. The size of the interval temporal graph for the same dataset on disk is 1.24 GB as compared 13.6 GB for the corresponding contact sequence graph.

Table 4 Dataset run-times in seconds

Dataset	Foremost			Min-hop				
	Wu	Ours	Wu/ours	Wu	Xuan	Ours	Wu/ours	Xuan/ours
Koblenz run-times								
Epin	0.0012	0.0040	0.3	0.0039	0.052	0.0052	0.74	10
Elec	0.00020	0.00033	0.606	0.00052	0.0022	0.00033	1.55	6.66
Fb	0.0010	0.0011	0.9	0.0030	0.042	0.0011	2.65	38.18
Flickr	0.073	0.26	0.27	0.41	2.94	0.42	0.98	7.024
Growth	0.14	0.89	0.16	1.44	11.40	1.31	1.1	8.69
Youtube	0.03	0.018	1.64	0.097	0.34	0.013	7.38	25.8
Digg	0.0001	0.0001	1.04	0.0004	0.003	7.115e−05	5.7	50.5
Slash	0.0003	0.001	0.28	0.001	0.011	0.001	1	10.59
Conflict	0.004	5.4e−07	1.8e3	0.004	0.018	5.9e−07	6.7e3	31.4e3
Arxiv	0.006	0.006	0.99	0.016	0.17	0.009	1.8	19.44
Wiki-en-edit	2.3	1.16	1.97	6.22	7.77	1.15	5.3	6.74
Enron	0.001	0.001	1.37	0.004	0.014	0.0017	2.86	8.76
Delicious	0.51	0.12	4.3	1.81	3.69	0.24	7.3	14.91
Synthetic run-times with $\mu_I = 4, \mu_D = 5, \mu_T = 3$								
Youtube	0.31	0.21	1.46	4.11		0.32	12.7	
Flickr	1.35	0.47	2.84	19.2		1.08	17.6	
Livejournal	17.39	5.61	3.09	334.56		21.81	15.3	
Synthetic run-times with $\mu_I = 4, \mu_D = 8, \mu_T = 3$								
Youtube	0.44	0.20	2.15	6.72		0.33	20.1	
Flickr	1.79	0.47	3.77	23.13		1.01	22.7	

Run times

For the run times reported in this section, we assume that the graph is resident in memory (i.e., the read time from disk is not accounted for). This is consistent with the reporting in Wu et al. (2016a) and Bentert et al. (2020) and also practice where the temporal graph is input once and queried often for optimal paths. For the 13 Koblenz datasets, we report the average of the runtimes from 100 randomly selected source vertices and for the synthetic graphs the reported run times are the average for 5 randomly selected source vertices. The reduction from 100 to 5 was necessitated by the bigger size of the synthetic datasets that resulted in substantially larger run times.

The average run-times (in seconds) for the Koblenz and synthetic datasets are given in Table 4. The speedups (time taken by competing algorithms of Wu et al. (2016a) and Bui-Xuan et al. (2003)/time taken by our algorithm) is also shown visually in Figs. 9, 10, 11, 12 and 13. In the visual representation, the speedup for the Koblenz dataset "conflict" is shaded differently from the others as this speedup is too large to display. For the foremost paths problem, our algorithm outperforms that of Wu et al. (2016a) on 6 of the 13 Koblenz datasets and all 5 of the synthetic datasets. The speedups obtained by us over the algorithm of Wu et al. (2016a) range from 0.286 to 1800 for the Koblenz datasets and from 1.46 to 3.77 on the synthetic datasets. This is in contrast to the results in Wu et al. (2016a) where the one-pass foremost paths algorithm of Wu et al. (2016a) outperformed the algorithm of Bui-Xuan et al. (2003) that we enhanced, on all datasets and often by a factor more than 10!

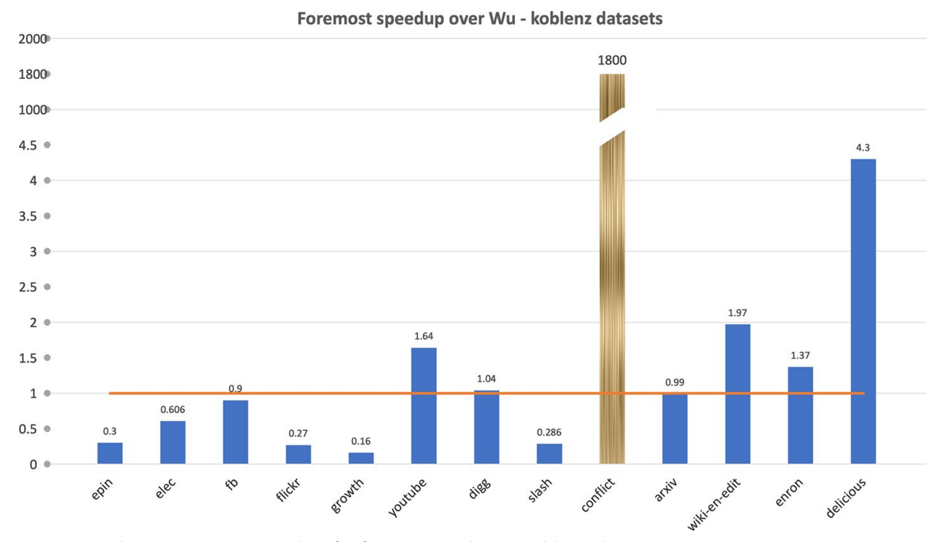

Fig. 9 Speedups over Wu's algorithm for foremost paths on Koblenz datasets

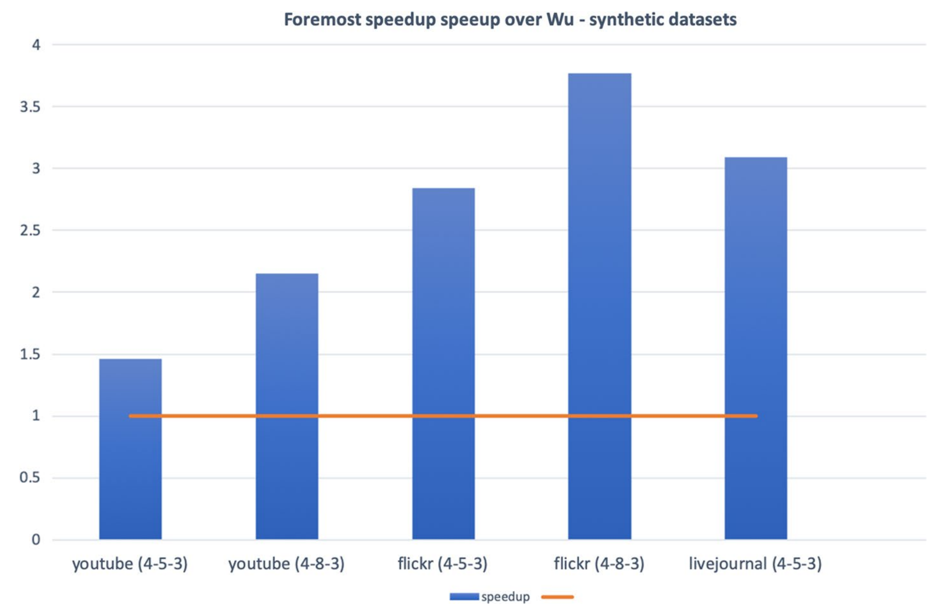

Fig. 10 Speedups over Wu's algorithm for foremost paths on synthetic datasets

For the min-hop problem, our algorithm outperforms that of Bui-Xuan et al. (2003) on all of the 13 Koblenz datasets and also that of Wu et al. (2016a) on 11 of the 13 Koblenz datasets. Our algorithm outperforms that of Wu et al. (2016a) on all 5 of the synthetic datasets. We do not compare against the algorithm of Bui-Xuan et al. (2003) on the synthetic datasets as that algorithm assumes the travel times to be the same across all intervals on a given edge; an assumption that is not valid for our synthetic datasets. The speedups obtained by us over the algorithm of Bui-Xuan et al. (2003) range from 6.66 to 31,000 on the Koblenz datasets. The speedups obtained by us over the algorithm of Wu et al. (2016a) range from 0.74 to 6700 for the Koblenz datasets and from 12.7 to 22.7 on the synthetic datasets. The reason for the unusually high speedup observed for

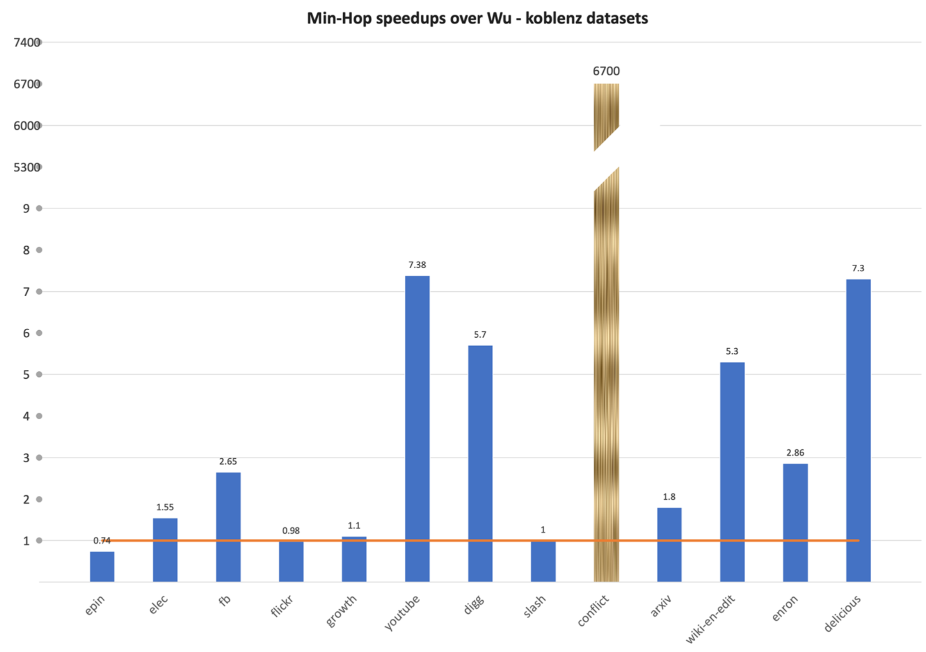

Fig. 11 Speedups over Wu's algorithm for min-hop paths on Koblenz datasets

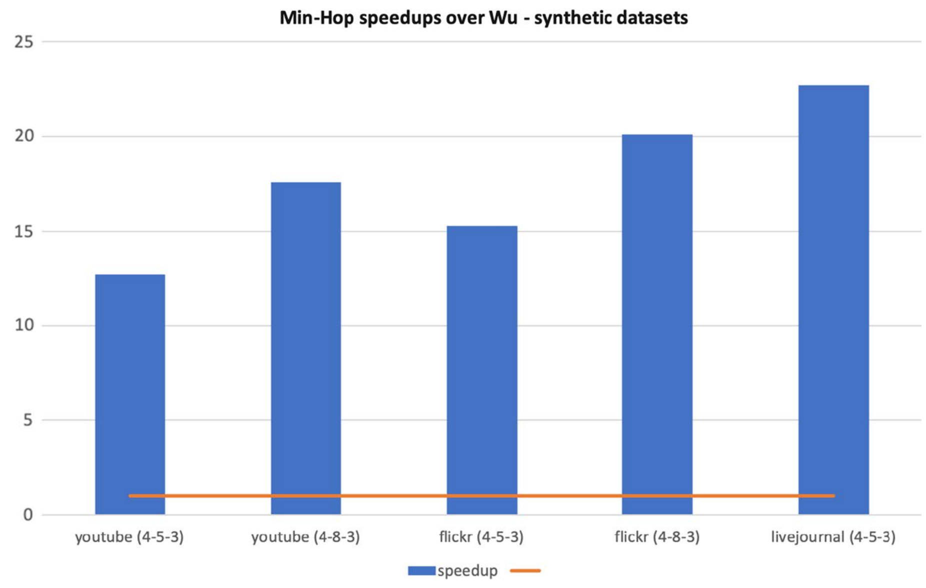

Fig. 12 Speedups over Wu's algorithm for min-hop paths on synthetic datasets

the Koblenz dataset "conflict" is that this dataset has very low temporal connectivity and our foremost and min-hop algorithms are able to detect this and thus terminate very quickly without examining all edges while the algorithms of Wu et al. (2016a) and Bui-Xuan et al. (2003) necessarily examine all edges.

Algorithm of Wu et al. (2016a) outperforms our algorithm on some of the Koblenz datasets. Our algorithm works on interval temporal graph model whereas the algorithm of Wu et al. (2016a) works on the contact sequence graph model. Interval temporal

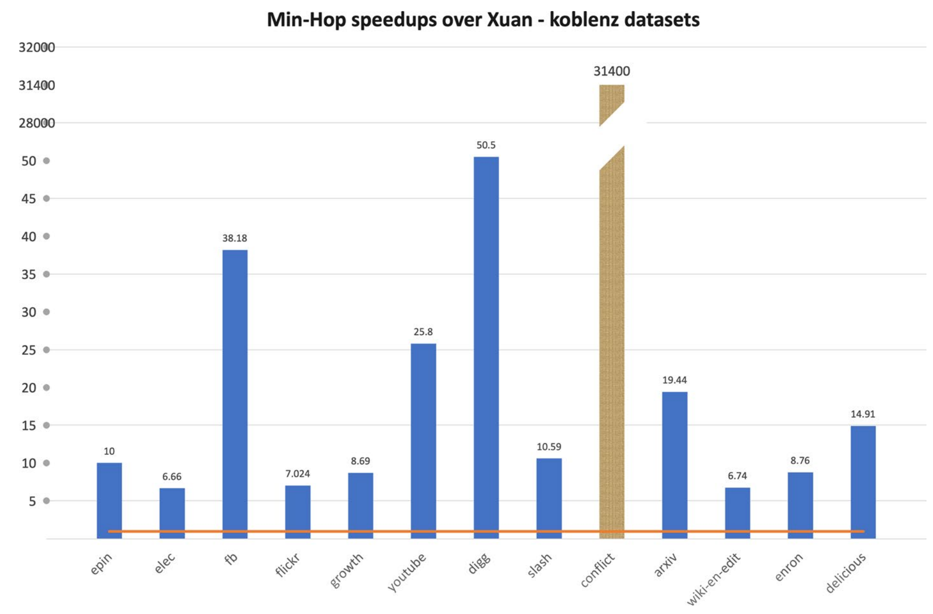

Fig. 13 Speedups over Xuan's algorithm for min-hop paths on Koblenz datasets

graphs are a superset of contact sequence graphs as described in "Introduction" section. Koblenz datasets are represented as contact sequence model that need to be transformed to the interval model for experiments with our algorithm. For graphs that are small in size and have low temporal activity, contact sequence model is a more efficient representation and Wu et al's algorithms can outperform our algorithms for such graphs. As these graphs get larger in size and temporal activity, our algorithms outperform that of Wu et al. (2016a) as illustrated in Table 4

The synthetic datasets described in Table 2, have time intervals of edge connectivity that translate to contact sequence edges at each time instance in such time intervals. These graphs have much higher temporal activity as described in Table 2. Our algorithm outperforms that of Wu et al. (2016a) on each of these graph instances.

Conclusion

We have demonstrated path problems that are NP-hard in the interval temporal graph model but solvable in polynomial time in the contact sequence temporal graph model. Additionally, we have extended the foremost path algorithm of Bui-Xuan et al. (2003) to the case when the time to traverse an edge may vary from one time interval to the next and developed a greedy algorithm to find min-hop paths in interval temporal graphs that is substantially faster than the min-hop algorithm of Bui-Xuan et al. (2003). Our new algorithms were benchmarked against the fastest algorithms in Wu et al. (2016a) using both the datasets used in Wu et al. (2016a) and synthetic datasets. Our algorithm for foremost paths outperformed the algorithm in Wu et al. (2016a) on 6 of the 13 Koblenz datasets used in Wu et al. (2016a) and all 5 of the synthetic datasets. The speedups obtained by us range from 0.286 to 1800 for the Koblenz datasets and from 1.46 to 3.77 on the synthetic datasets. For the min-hop problem, our algorithm outperforms that of Bui-Xuan et al. (2003) on all 13 Koblenz datasets and the speedups range from 6.6 to

31000. Our algorithm also outperforms that of Wu et al. (2016a) on 11 of the 13 Koblenz datasets and all 5 of the synthetic datasets. The speedups obtained over Wu et al. (2016a) range from 0.74 to 6700 for the Koblenz datasets and from 12.7 to 22.7 on the synthetic datasets. Going forward we would like to extend our work to compute paths and walks with more than one optimization criteria. For example, when computing foremost walks and paths in a temporal graph, it would be interesting to examine problems that require finding foremost walk with minimum waiting time along the walk. In other words we would like to explore problems like min-wait foremost walks, min-hop foremost walks, min-cost foremost walks etc. We would also like to study the problem of optimizing linear combination of multiple optimization criteria for interval temporal graphs which is studied by Bentert et al. (2020) for contact sequence graphs.

Abbreviations

NAPP	No-wait acyclic path problem
V	Number of vertices in a graph
N	Used interchangeably with V
E_s	Number of edges in underlying static graph
M_{itg}	Number of edges in interval temporal graph (same as E_s)
M_{csg}	Number of contact sequence edges
λ	Travel duration on an edge at a given departure time
δ	Maximum number of departure intervals on an edge

Acknowledgements
We acknowledge Wu et al. (2016a) and Bentert et al. (2020) for sharing their implementation with us to benchmark against.

Author contributions
Both authors contributed equally to the paper. Both authors have read and approved the final manuscript.

Funding
N/A

Availability of data and materials
The koblenz datasets used for benchmarking are available in the KONECT graphs (Kunegis 2013). The social network graphs of ***youtube, flickr, livejournal*** are available at ***http://socialnetworks.mpi-sws.org/data-imc2007.html***. This was shared by the authors of Mislove et al. (2007)

Declarations

Competing interests
The authors declare that they have no competing interests.

Received: 2 October 2021 Accepted: 11 August 2022
Published online: 29 August 2022

References

Bentert M, Himmel A-S, Nichterlein A, Niedermeier R (2020) Efficient computation of optimal temporal walks under waiting-time constraints. Appl Netw Sci 5(1):73. https://doi.org/10.1007/s41109-020-00311-0
Bhadra S, Ferreira A (2003) Complexity of connected components in evolving graphs and the computation of multicast trees in dynamic networks. In: Pierre S, Barbeau M, Kranakis E (eds) Ad-hoc, mobile, and wireless networks. Springer, Berlin, Heidelberg, pp 259–270
Bhadra S, Ferreira A (2012) Computing multicast trees in dynamic networks and the complexity of connected components in evolving graphs. J Internet Serv Appl 3(3):269–275. https://doi.org/10.1007/s13174-012-0073-z
Bui-Xuan B-M, Ferreira A, Jarry A (2003) Evolving graphs and least cost journeys in dynamic networks. In: WiOpt'03: modeling and optimization in mobile, ad hoc and wireless networks, Sophia Antipolis, France, p 10. https://hal.inria.fr/inria-00466676
Casteigts A, Himmel A, Molter H, Zschoche P (2019) The computational complexity of finding temporal paths under waiting time constraints. arXiv:1909.06437
Casteigts A, Raskin M, Renken M, Zamaraev V (2020) Sharp thresholds in random simple temporal graphs. arXiv:2011.03738
Erlebach T, Hoffmann M, Kammer F (2021) On temporal graph exploration. J Comput Syst Sci 119:1–18. https://doi.org/10.1016/j.jcss.2021.01.005

Guo F, Zhang D, Dong Y, Guo Z (2019) Urban link travel speed dataset from a megacity road network. Sci Data 6(1):61. https://doi.org/10.1038/s41597-019-0060-3

Holme P, Saramäki J (2012) Temporal networks. Phys Rep 519(3):97–125. https://doi.org/10.1016/j.physrep.2012.03.001

Kuhn F, Oshman R (2011) Dynamic networks: models and algorithms. SIGACT News 42(1):82–96. https://doi.org/10.1145/1959045.1959064

Kunegis J (2013) Konect: The koblenz network collection. In: Proceedings of the 22nd international conference on world wide web. WWW '13 companion. Association for Computing Machinery, New York, NY, USA, pp 1343–1350. https://doi.org/10.1145/2487788.2488173

Lightenberg W, Pei Y, Fletcher G, Pechenizkiy M (2018) Tink: a temporal graph analytics library for apache flink. In: Companion proceedings of the the web conference 2018. WWW '18. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, pp 71–72. https://doi.org/10.1145/3184558.3186934

Michail O (2015) An introduction to temporal graphs: an algorithmic perspective. arXiv:1503.00278

Michail O, Spirakis PG (2016) Traveling salesman problems in temporal graphs. Theor Comput Sci 634:1–23. https://doi.org/10.1016/j.tcs.2016.04.006

Mislove A, Marcon M, Gummadi KP, Druschel P, Bhattacharjee B (2007) Measurement and analysis of online social networks. In: Proceedings of the 5th ACM/Usenix internet measurement conference (IMC'07), San Diego, CA

Sahni S (2004) Data structures, algorithms, and applications in C++, 2nd edn. Silicon Press, Summit

Santoro N, Quattrociocchi W, Flocchini P, Casteigts A, Amblard F (2011) Time-varying graphs and social network analysis: temporal indicators and metrics. arXiv:1102:0629

Scheideler C (2002) Models and techniques for communication in dynamic networks. In: Proceedings of the 19th annual symposium on theoretical aspects of computer science. STACS '02. Springer, Berlin, Heidelberg, pp 27–49

Stojmenović I (2002) Location updates for efficient routing in ad hoc networks. Wiley, Hoboken, pp 451–471. https://doi.org/10.1002/0471224561.ch21

Wu H, Cheng J, Ke Y, Huang S, Huang Y, Wu H (2016a) Efficient algorithms for temporal path computation. IEEE Trans Knowl Data Eng 28(11):2927–2942. https://doi.org/10.1109/TKDE.2016.2594065

Wu H, Cheng J, Ke Y, Huang S, Huang Y, Wu H (2016b) Appendix-h of efficient algorithms for temporal path computation. IEEE Trans Knowl Data Eng 28(11):2927–2942. https://doi.org/10.1109/TKDE.2016.2594065

Zschoche P, Fluschnik T, Molter H, Niedermeier R (2018) The complexity of finding small separators in temporal graphs. arXiv:1711:00963

Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Submit your manuscript to a SpringerOpen journal and benefit from:

► Convenient online submission
► Rigorous peer review
► Open access: articles freely available online
► High visibility within the field
► Retaining the copyright to your article

Submit your next manuscript at ► springeropen.com