# An Effective Data Structure for Contact Sequence Temporal Graphs

Sanaz Gheibi
*CISE department*
*University of Florida*
Gainesville, Florida, USA
sgheibi@ufl.edu

Tania Banerjee
*CISE department*
*University of Florida*
Gainesville, Florida, USA
tmishra@ufl.edu

Sanjay Ranka
*CISE department*
*University of Florida*
Gainesville, Florida, USA
sranka@ufl.edu

Sartaj Sahni
*CISE department*
*University of Florida*
Gainesville, Florida, USA
sahni@ufl.edu

2021 IEEE Symposium on Computers and Communications (ISCC) | 978-1-6654-2744-9/21/$31.00 ©2021 IEEE | DOI: 10.1109/ISCC53001.2021.9631469

***Abstract*—We propose a new time-respecting data structure (TRG) for contact sequence temporal graphs that is more memory efficient than previously proposed TRGs. Our new TRG alters the balance between TRG structures and the ordered sequence of edges (OSE) data structure. While TRG structures have an obvious performance advantage over OSE for problems that can be solved via a shallow neighborhood search, previous research has shown that single-source all-destinations problems are more effectively solved using OSE. The competitiveness of our TRG structure for this class of problems is demonstrated for the single-source all-destinations fastest paths and min-hop paths problems. Our TRG structure retains the advantage that other similar structures have over OSE for shallow neighborhood search problems.**

***Index Terms*—Contact sequence temporal graphs, data structures, fastest paths, min-hop paths.**

## I. Introduction

Temporal graphs are an extension of classical graphs in which edges and/or vertices have temporal information (i.e., timestamps) associated with them. In this paper, we shall limit ourselves to the case when temporal information is associated with only the edges. Temporal graphs have been used to model problems in communication networks, computational biology, transportation networks, spread of virus, social networks, and so on [1]–[3].

In a *contact sequence temporal graph* $G = (V, E)$, each edge $e \in E$ has the format $< u, v, t, w >$ where $u$ and $v$ are the source and target vertices, respectively, of the edge; $t$ is its time stamp; and $w$ ($w >= 0$) is the time it takes to get from $u$ to $v$ along this edge. The time stamp $t$ indicates the time at which one may begin to travel from $u$ along this edge; $t+w$ is the arrival time at $v$. Note that $G$ may have many edges from $u$ to $v$ each with a different time stamp and possibly different weights $w$. The different timestamps define the permissible departure times from $u$ to $v$ using a single edge. In an *interval temporal graph*, $G = (V, E)$ there is at most one edge from $u$ to $v$ for any pair of vertices $(u, v)$; this edge is labeled with a sequence of intervals of the form $(s, f, w)$. Each such interval describes a range of permissible start times from $u$ (i.e., one can begin to travel on this edge at any time $T$, $s <= T <= f$). For a permissible start time $T$, the arrival time at $u$ is $T+w$. It is easy to see that every contact sequence temporal graph may be represented by an equivalent interval temporal graph and that when time is discrete the reverse is also true. We note that some applications are naturally modeled as contact sequence temporal graphs and others as interval temporal graphs [3].

A *time-respecting path* (or simply path) from $u$ to $v$ has the property that the departure time from each vertex on the path is $>=$ the arrival time at that vertex. Different types of (time-respecting) paths have been defined and studied in the literature for temporal graphs (see, for example, [4] and [5]). Among them are "earliest arrival" (also known as "foremost") paths, "latest departure" paths, "fastest" paths, "shortest" paths and "min-hop" paths. Typically, in a path finding problem, we are given an interval $[t_{start}, t_{end}]$, where $t_{start}$ defines the earliest permissible start time from the source of the path and $t_{end}$ defines the latest permissible arrival time at the destination of the path. So, for example, if $u$ is the source vertex and $v$ the destination, the *feasible paths* from $u$ to $v$ are all time-respecting paths that depart $u$ at or after $t_{start}$ and arrive at $v$ at or before $t_{end}$. The earliest arrival or foremost path is a feasible path that arrives at $v$ at the earliest possible time; a feasible path with the latest departure from $u$ is the latest departure path; a feasible path for which the difference between the arrival time at $v$ and the departure time from $u$ is minimum is the fastest path; a shortest path minimizes the sum of the weights of the edges on the path; and a min-hop path minimizes the number of edges on the path.

Wu et al. [5] propose two data structures for contact sequence temporal graphs. The first is a time ordered sequence of edges (OSE) (i.e., non-decreasing order of timestamps) and the second is a graph in which all paths are time-respecting (TRG-Wu). They develop algorithms for the single source all destinations optimal paths problem (where the optimization metric is one of: earliest arrival, fastest, and shortest) as well as for the single destination all sources latest departure problem for both data structures and demonstrate that their contact sequence temporal graph algorithms are considerably faster than those in [4] for the interval temporal graph model. Further, their algorithms for the OSE data structure are faster than those for the TRG-Wu structure (an exception being the case of fastest paths where the OSE algorithm was faster than the TRG-Wu algorithm on some of their test data and slower on others). While OSE has been demonstrated to be faster than TRG-Wu on the stated path problems, it is clearly

978-1-6654-2744-9/21/$31.00 ©2021 IEEE

slower for other problems such as those that depend on only local properties. For example, is there a feasible one-hop or two-hop path from $u$ to $v$ (recall that feasible paths are time-respecting paths that start and end in the given time interval $[t_{start}, t_{end}]$)? Hence, there is interest in developing a more effective data structure than TRG-Wu; one that is competitive with or superior to OSE for single source all destinations path problems and competitive with TRG-Wu for problems on which it is superior to OSE. We note that Zschoche et al. [6] propose an alternative representation, TRG-Zchoche, for contact sequence temporal graphs in which all paths are time respecting. While this representation is more memory efficient than TRG-Wu, they develop only an algorithm for the single source all destinations shortest paths problem under the assumption that no edge has a weight that is 0.

In this paper, we propose a new TRG data structure, TRG-Ours, that uses less memory than both TRG-Wu and TRG-Zschoche. We demonstrate its effectiveness relative to Wu-TRG using the single source all destinations fastest and the min-hop path problems as examples.

We review the related work in Section II. In Section III, we describe the 3 data structures TRG-Wu, TRG-Zschoche, and TRG-Ours and analyze the memory requirement of each. The fastest and min-hop paths algorithms for TRG-Ours are developed in Sections IV and V, respectively. Experimental results are presented in Section VI. We conclude in Section VII.

## II. Related Work

The work of Wu et al. [5] is closest to our work. We have already mentioned this work in section I and will further discuss their method in the sections that follow. Xuan et al. [4] develop single-source all-destinations algorithms for interval temporal graphs. The main difference between the model they use and the one used in this paper is that in our model, the edge traversals times are functions of edge timestamps while that is not the case in their model.

Another group of methods find optimal paths in the presence of constraints. Zhao et al. [7] propose methods to find approximate shortest paths in a graph such that the paths are subject to multiple constraints. As already mentioned, the paths they find are approximate not exact. Hassan et al. [8] also tackle the problem of path finding in constrained graphs where the constraints are imposed using edge labels. Their method works on graphs in which the edges are labeled. Examples of edge labels are family/fried relationships in social media. Himmel et al. [9] develop an algorithm to find optimal walks in contact sequence temporal graphs. Their model allows the specification of min and max wait times at each vertex and also optimizes a linear combination of criteria (e.g., fastest, shortest, foremost). Experimental results reported by them indicate that the median running time of their algorithm is comparable to that of the algorithms of [5] but that the average running time (averaged over all optimization criteria considered) is about 10 times that of the algorithms of [5]. They state that the relatively high average run time of their algorithm is a weakness.

Some of the existing methods form Shortest Path Trees (SPTs) rooting at each node and update the trees each time the graph is updated. In [10] the authors set an upper bound for the height of the Shortest Path Trees (SPTs). Using the statistical approaches, they find $(1 + e)$-approximation paths where $e \rightarrow 0$. Alshammari and Rezgui [11] update their SPT forest by working independently on each SPT. They first identify for each updated edge $< u, v >$ whether or not the distance of $v$ from the root node has been affected by the update. They then update its distance and use a modification of Dijkstra's algorithm to update any affected distances in the subtree rooted at $v$. [12] updates SPTs allowing for batch update of edges. The main difference of these methods from ours is that here the edge updates are not known in advance, but they are known in our model.

Another approach is to use a preprocessing step to store the distances from each node to a group of landmark nodes. These stored distances get updated after each graph update. Pairwise distances are computed by combining these partial distances. In [13] the landmark nodes are the hub nodes. For computing the distance from $u$ to $v$, they search for hub node $x$ such that $distance(u, x) + distance(x, v)$ is minimized. Hong et. al. [14] use this method for the large graphs that do not fit into the RAM. For each node $u$, they store its distance from nodes $v$ which are in a radius of $K$ from it. Tretyakov et. al. [15] use an assumption that the triangle inequality holds on their target graphs. As the landmark nodes, they use the nodes for which the triangle inequality turns into equality for a sample subset of paths. These methods are proper for the problems in which the edge updates are not known in advance.

Cvetkovski and Crovella [16] propose a greedy method for a temporal network model that is similar to wireless networks. Meaning that the nodes can appear and disappear and that the distance of the nodes is determined by their geographic location. [17] also focuses on temporal graphs in which the "nodes" are updated dynamically. Our model is more general in that a node can be removed when the weight of all its edges change to infinity. We also don't make assumptions about the node distances. Clementi et al. [18] put an upper bound on the flooding time in mobile ad-hoc networks. The flooding info can also be used to find the upper bound on the length of the fastest paths in such networks.

The specific requirements of different applications motivate different solution methods. Calle et al. [19] use a modified version of Sense of Smell Ant Colony Optimization (SoSACO) to find approximate shortest paths in temporal graphs. Their focus is on the graphs for which the query response time is more important than the path length itself. In the method proposed by Chen et. al. [20], the target networks are delay-tolerant. Their main concern is security.

The authors in [21] propose self-adapting methods that converge to optimal paths using differential equations. The source node is fixed in their method and the edge updates are not known ahead of time.

Brunelli et al. [22] tackle the problem of finding pareto-optimal paths in temporal graphs. Their formulation is for the

cases where the start time is fixed. Our method considers all the possible start times.

Distributed algorithms for finding shortest paths are developed in [23]–[25] and surveys of the general area of temporal graphs appear in [26], [27].

## III. TRG Data Structures

Figure 1 shows an example contact sequence temporal graph $G = (V, E)$. Each edge has a label $(t, w)$ that gives its time stamp and weight. For this graph, $|V| = 3$ and $|E| = 4$. If we remove the edge labels $(t, w)$ and coalesce the resulting identical edges (e.g., with the edge labels removed, the 2 edges from vertex $A$ to vertex $B$ in our example become identical and coalesce into a single edge), we get the corresponding *static graph*. Let $|E_s|$ be the number of edges in this static graph. For our example, $|E_s| = 3$. The ratio $|E|/|E_s|$ is the *edge activity*. The edge activity for our example graph is 1.33. Some paths in the graph of Figure 1(a) are time respecting (e.g., the $ABC$ path that used the edge $< A, B, 4, 1 >$) and others are not (e.g., the $ABC$ path that used the edge $< A, B, 3, 4 >$).

The OSE data structure for our example is given in Figure 1(b). Each tuple in the OSE structure has the form $(u, v, t, w)$ and represents an edge of Figure 1(a) that is directed from vertex $u$ to vertex $v$, has the time stamp $t$, and weight $w$. To find a time-respecting path from $u$ to $v$ in the OSE structure one may need to examine all edges. Similarly, to find the one-hop neighbors of $u$ requires the examination of all edges. Wu et al. [5] have, however, developed simple and fast one-pass algorithms (the edges in OSE are examined in a single top-to-bottom pass) for several single source all destinations paths problems.

A time-respecting graph (TRG) has the property that all paths are time respecting. We first describe the TRGs of [5] and [6] and then describe our proposed TRG.

### A. TRG-Wu

Recall that the edge $(u, v, t, w)$ permits one to depart vertex $u$ at time $t$ and arrive at vertex $v$ at time $t+w$. Let $T_{out}(u)$ be the set of distinct timestamps on edges of the form $(u, *, t, w)$ in $G = (V, E)$; i.e., $T_{out}(u)$ is the set of distinct times at which one may depart from vertex $u$. Let $T_{in}(u)$ be the set of distinct times at which one may arrive at vertex $u$. For each vertex $u$ in $G$, the TRG, $G' = (V'E')$ of Wu et al. [5] has two sets of vertices

$$V_{in}(u) = \{u' =< u, t > | t \in T_{in}(u)\}$$

$$V_{out}(u) = \{u' =< u, t > | t \in T_{out}(u)\}$$

So, $V' = \bigcup_u \{V_{in}(u) \cup V_{out}(u)\}$. The edge set $E'$ is defined as below.

1) For each $u \in V$ link the vertices in $V_{in}(u)$ into a directed chain in ascending order of time (arrival time). Each edge on the chain has a weight of 0. Similarly, link the vertices in $V_{out}(u)$ into a directed chain in ascending order of out time (departure time). Again, the weight of each edge is 0.
2) For each $u \in V$ examine the vertices in $V_{in}(u)$ in decreasing order of time. When $< u, t_1 > \in V_{in}(u)$ is being examined, determine the least out time $t_2 >= t_1$ in $T_{out}(u)$. If an edge from a vertex in $V_{in}(u)$ to $< u, t_2 > \in V_{out}(u)$ hasn't already been added, add a directed edge from $< u, t_1 > \in V_{in}(u)$ to $< u, t_2 > \in V_{out}(u)$; the weight of this edge is 0.
3) For each edge $e = (u, v, t, w) \in E$, add a directed edge with weight $w$ from $< u, t > \in V_{out}(u)$ to $< v, t+w > \in V_{in}(v)$.

Figure 1(c) shows TRG-Wu for the example graph of Figure 1(a). It is easy to see that for every time-respecting path in $G$ there is a path in $G'$ and vice versa. Further, problems such as finding all vertices in $G$ reachable from $u$ (by a time-respecting path) are easily solved using classical graph algorithms such as depth- and breadth-first search on $G'$. Moreover, these problems can be solved faster using $TRG - Wu(G)$ than using $OSE(G)$ (especially when the number of reachable vertices is much less than $|V|$). From the described construction for $G'$, we see that

$$|V'| = \Sigma_{u \in V}(|T_{in}(u)| + |T_{out}(u)|) \leq 2|E| \tag{1}$$

$$\begin{aligned} |E'| &\leq \Sigma_{u \in V}(|T_{in}(u)| - 1) + \Sigma_{u \in V}(|T_{out}(u)| - 1) \\ &\quad + \Sigma_{u \in V} \min\{T_{in}(u), T_{out}(u)\} + |E| \\ &\leq \Sigma_{u \in V}(T_{in}(u) + T_{out}(u) + \min\{T_{in}(u), T_{out}u)\}) \\ &\quad - 2|V| + |E| \leq 4|E| - 2|V| \end{aligned} \tag{2}$$

### B. TRG-Zschoche

TRG-Zchoche [6] is similar to TSG-Wu. The main difference is that for each $u \in V$, the $V_{in}(u)$ and $V_{out}(u)$ vertices are put into a single directed chain in increasing order of time rather than into two separate chains as in TRG-Wu. It is assumed that all edge weights are integers and $> 0$. Let $G'' = (V'', E'')$ be the TRG-Zscoche for $G$. The following describes the construction of $G''$ graph from $G$.

1) Replace each edge $< u, v, t, w >$ with $w > 1$ by two edges $< u, v_{New}, t, 1 >$ and $< v_{New}, v, t + w - 1, 1 >$. Let's call the modified graph $G_1 = (V_1, E_1)$. This transformation ensures that every path in $G$ that uses the original $< u, v, t, w >$ gets to $v$ at time $t+w$ in $G_1$.
2) For each vertex $u \in V_1$, let $\Phi(u) = T_{in}(u) \cup T_{out}(u)$ and for each $t \in \Phi(u)$ add a vertex $< u, t >$ to $V''$.
3) For each $u \in V_1$, link the vertices $(u, *)$ into a directed chain in increasing order of time. The weight of each chain edge is 0.
4) For each edge $< u, v, t, w > \in E_1$ add a directed edge of weight = $w$ from $< u, t > \in V''$ to $< v, t+w > \in V''$.

Figure 1(d) shows TRG-Zschoche for the example graph of Figure 1(a). Nodes $< B_n, * >$ represent the $B_{new}$ nodes added to the graph according to step 1. One may verify that TRG-Zschoche has the same time-respecting properties as does TRG-Wu. Bounds on $|V''|$ and $|E''|$ are computed below.

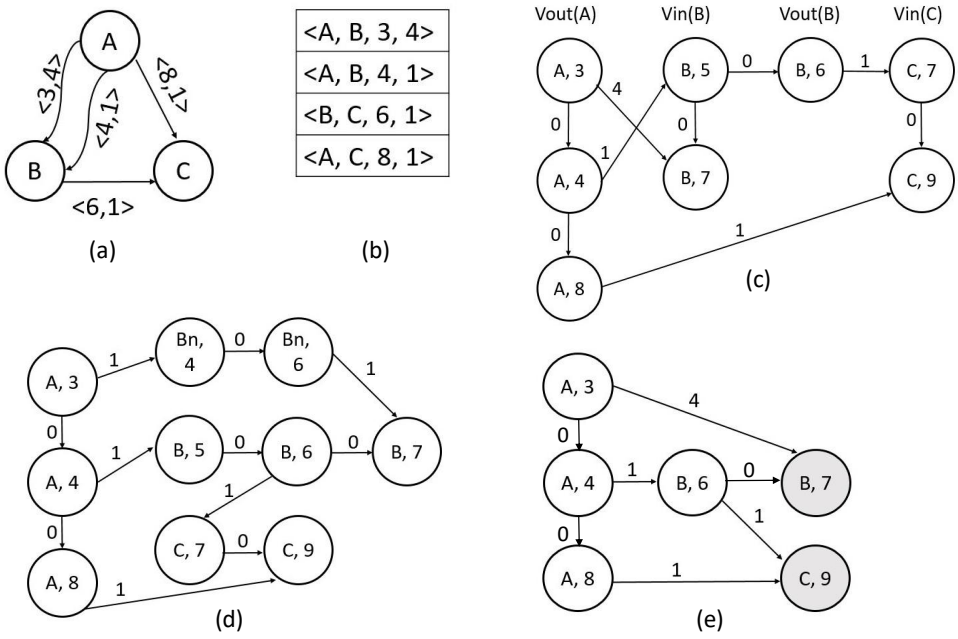


Fig. 1. (a) A temporal graph G, (b) OSE(G), (c) TRG-Wu(G), (d) TRG-Zschoche(G), and (e) TRG-Ours(G). For parts (d) and (e), as all the nodes $(u, *)$ belong to sets $\phi(u)$ and $V_{out}(u)$, respectively, we have not explicitly labeled them as we have done for (c).

1) $|V_1| = |V| + D$ and $|E_1| = |E| + D$ where $D <= |E|$ is the number of edges in $G$ with weight $> 1$.
2) $|V''| = \Sigma_{u \in V_1} |\Phi(u)| \leq \Sigma_{u \in V_1} (T_{in}(u) + T_{out}(u)) <= 2|E_1| = 2|E| + 2D \leq 4|E|$.
3) The number of chain edges is $\Sigma_{u \in V_1} (|\Phi(u)| - 1) \leq \Sigma_{u \in V_1} (|T_{in}(u)| + |T_{out}(u)| - 1) <= 2|E_1| - |V_1| = 2|E| + 2D - |V| - D \leq 3|E| - |V|$. The number of other edges in $E''$ is $|E_1| \leq 2|E|$. So, $|E''| \leq 5|E| - |V|$.

As can be seen the bounds for the number of vertices and edges are higher for TRG-Zschoche than for TRG-Wu. In TRG-Zschoche, all edge weights are either 0 or 1.

### C. TRG-Ours

Our TRG data structure uses fewer vertices and edges than does TRG-Wu. Let $G''' = (V''', E''')$ denote our TRG for $G = (V, E)$. Let $V_{out}(u)$ be as for TRG-Wu. We define $t_m = \max\{t : t \in T_{in}(u)\}$. Let $V'_{out}(u) = V_{out}(u) \cup \{< u, t_m >\}$ in case $T_{out}(u) = \emptyset$ or $t_m > \max\{t : t \in T_{out}(u)\}$. $V'_{out}(u)$ is equal to $V_{out}(u)$ otherwise. We call nodes $< u, t_m >$ "helper nodes" that are used to prevent erroneous elimination of nodes and edges in our transformation that only uses $V_{out}$ nodes.

$$V''' = \bigcup_{u \in V} V'_{out}(u)$$

For each edge $< u, v, t, w >$ in $E$, $E'''$ has a directed edge from $< u, t > \in V'''$ to $< v, t' > \in V'''$ such that $t'$ is the smallest time $>= t + w$ in $\{\bar{t} :< v, \bar{t} > \in V'_{out}(v)\}$. Additionally, for every vertex v, the vertices in $V'_{out}(v)$ are chained in ascending order of timestamp. Figure 1(e) shows TRG-Ours for the example graph of Figure 1(a). We have highlighted the "helper nodes". If we omit the helper nodes, then the algorithms on the resulting graph will work as if node $C \in V$ and both its incoming edges and edge $< A, B, 3, 4 >$ have been removed from the original graph $G =< V, E >$. Again, one may verify that TRG-Ours has the same time-respecting properties as does TRG-Wu. Bounds on $|V'''|$ and $|E'''|$ are computed below.

$$|V'''| \leq \Sigma_{u \in V} (|V_{out}(u)| + 1) \leq |E| + |V| \tag{3}$$

$$\begin{aligned} |E'''| &\leq |E| + \Sigma_{u \in V} |Tout(u)| \\ &\leq |E| + |E| = 2|E| \end{aligned} \tag{4}$$

We see that TRG-Ours will never have more vertices nor more edges than TRG-Wu or TRG-Zschoche.

## IV. Fastest Paths Algorithm For TRG-Ours

Algorithm 1 gives our algorithm to find the fastest (feasible) paths from a given source vertex $s$ to all reachable destinations. While the algorithm as stated finds only the duration (arrival time at the destination - departure time from the source) of these paths, it is easily modified to construct the fastest paths. The input to the algorithm is a TRG-Ours graph $G$, the source vertex $s$ for the paths to be found, the earliest permissible

start/departure time from $s$ ($t_{start}$), and the latest permissible arrival time at any destination ($t_{end}$). In line 4, we sort $V_{out}(s)$ in decreasing order of the time. The loop in line 5 runs rounds of DFS (depth first search) where in each round, a DFS starts from a new node $\in V_{out}(s)$ and continues visiting the non-visited nodes. lines 6-8 mark the node as visited and check whether or not it is within the $[t_{start}, t_{end}]$ range. line 9 initializes the DFS stack and line 10 sets the start time for the paths to be traversed in this DFS round. The while loop in line 11 performs the DFS. Lines 12 and 13 pop the top of the stack and the for loop in line 14 iterates over all its neighbors. In lines 15-18 the enter time to the neighbor and the timespan of the trip to the neighbor are computed and if the timespan is less then the current distance of the neighbor, the distance gets updated. Lines 19-29 check if the neighbor$< u', t' >$ is not visited and is within the $[t_{start}, t_{end}]$ range. If so, they mark it as visited and push it to the stack. They do the same about all the chain neighbors of $u'$ with timestamps $> t'$.

The time complexity of $FastestPaths$ is $O(n+e)$, where $n$ and $e$ are, respectively, the number of vertices and number of edges in the input TRG-Ours graph.

## V. Min-Hop Algorithm for TRG-Ours

Our min-hop algorithm (Algorithm 2) is a modification of the standard BFS (breadth first search) algorithm. The inputs are the same as for our fastest paths algorithm. While the algorithm as stated finds only the number of hops on min-hop paths, it is easily modified to find the actual paths. Similar to BFS, we add $s$ to the queue in line 4. In line 6, $eout[i]$ keeps the earliest time our traversal algorithm has exited from node $i$. In the case of $s$, it is $t_0$ that we computed in line 5. $newEout[i]$ is used to keep the earliest exit time updates of node $i$ that happen in the for loop at line 10. If we update the earliest exit times in the same $eout[]$ vector, then the updated values may interfere with the check done in line 13 (the for loop). We keep an invariant that the vectors $eout$ and $newEout$ are the same at the beginning of each while loop. In line 7, the variables "hopCount" and "nodeCount" keep track of the BFS levels and the number of nodes for which the distance is computed respectively.

The while loop in line 9 iterates over different BFS levels. The for loop in line 10 takes care of one BFS level. After we remove a node $u$ from $Q$ in lines 11 and 12, in the for loop in line 13, we iterate over all its chain neighbors ($Vout$ nodes after the last exit from $u$ that have not been visited yet). For each of these chain neighbors, we check all their neighbors (for loop in line 15) to see if their distances can be updated. The variable "tOut" in line 16 keeps the exit time from the current node. We keep an invariant that all the $V_{out}(u')$ nodes with timestamps after $newEout[u']$ have already been visited. (In fact, we keep reducing $eout[u]/newEout[u]$ and visit all the chain neighbors of $u$ from the currently earliest exit time to the previous earliest exit time). If the condition at line 17 is met, we further process $u'$. lines 18-24 check whether or not node $u'$ has been reached for the first time (if not, $newEout[u'] <$ infinity). If so, its distance gets updated and if all distances have been finalized, the function returns. lines 25-29 update $Eout$ and add $u'$ to $Q$ if it is not visited in this level of BFS. The boolean vector $newLive[]$ keeps track of the nodes visited in this level of BFS. Finally, lines 34-38 update $eout$ and $hopCount$ and reset all values in $newLive[]$ to False.

The time complexity of $MinhopPaths$ is $O(n+e)$, where $n$ and $e$ are, respectively, the number of vertices and number of edges in the input TRG-Ours graph.

**Algorithm 1** Our fastest paths algorithm

```
1: function FastestPaths(G, s, t_start, t_end)                    ▷
   G = (V, E) is TRG-Ours graph and s is the source node
2:     return Array fastest[] of fastest path durations
3:     Initialize fastest[s] = 0, all other distances to infinity
4:     Sort V_out(s) in decreasing order of t
5:     for < s, t_i > ∈ V_out(s) in sorted order do
6:         Mark < s, t_i > as visited
7:         if t_i < t_start break
8:         if t_i > t_end continue
9:         push < s, t_i > to stack ST
10:        t_init ← t_i
11:        while ST is not empty do
12:            Let < u, t > be top of the stack
13:            pop < u, t > from the stack
14:            for < u', t' > ∈ neighbors(< u, t >) do
15:                Let w_link be the weight of the link from
   < u, t > to < u', t' >
16:                inTime ← t + w_link
17:                if inTime ≤ t_end AND inTime − t_init <
   fastest[u'] then fastest[u'] ← inTime − t_init
18:                end if
19:                if < u', t' > is not visited then
20:                    Mark < u', t' > as visited
21:                    if t' ≤ t_end then
22:                        Push < u', t' > into ST
23:                        for all non-visited < u', t'' > ∈
   V_out(u') with t'' > t' do
24:                            if t'' > t_end then break
25:                            Mark < u', t'' > as visited
26:                            Push < u', t'' > into ST
27:                        end for
28:                    end if
29:                end if
30:            end for
31:        end while
32:    end for
33: end function
```

## VI. Experimental Results

In this section, we compare the performance of our fastest paths and min-hop paths algorithms based on TRG-Ours to the algorithms given in [5] using OSE and TRG-Wu. We do not compare with algorithms for TRG-Zschosche because (a) Zschose et al. [6] do not give algorithms for either of these problems and (b) TRG-Zschoche has more vertices and edges

**Algorithm 2** Our minhop paths algorithm

**function** MINHOPPATHS($G$, $s$, $t_{start}$, $t_{end}$) ▷ $G =< V, E >$ is TRG-Ours graph and $s$ is the source node
  **return** Array $minhop[]$ of number of hops in min-hop paths
  Initialize $minhop[s] = 0$, all other distances to infinity
  Push $s$ into the queue $Q$
  Let $t_0$ be the smallest time in $T_{out}(s)$ such that $t_0 \geq t_{start}$
  $eout[s] \leftarrow t_0$, $newEout[s] \leftarrow t_0$
  $hopCount \leftarrow 0$ , $nodeCount \leftarrow 0$
  Initialize $newLive[]$ with all $False$
  **while** $Q$ is not empty **do**
    **for** $i = 1...Q.size()$ **do**
      Let $u \in V$ be the top of $Q$
      Remove $u$ from $Q$
      **for** all non-visited $< u, t > \in V_{out}(u)$ such that $t \geq eout[u]$ AND $t \leq t_{end}$ **do**
        Mark $< u, t >$ as visited
        **for** $< u', t' > \in neighbors(u)$ **do**
          $tOut \leftarrow t'$
          **if** $tOut < newEout[u']$ **then**
            **if** $newEout[u']$ equals infinity **then** ▷ $< u', t' >$ is newly reached
              $minhop[u'] \leftarrow hopCount$
              $nodeCount \leftarrow nodeCount + 1$
              **if** $nodeCount$ equals $V$ **then**
                Return
              **end if**
            **end if**
            $newEout[u'] = t'$
            **if** $newLive[u']$ equals $False$ **then**
              Add $u'$ to $Q$
              $newLive[u'] \leftarrow True$
            **end if**
          **end if**
        **end for**
      **end for**
    **end for**
    **for** $j \in 1...V$ **do**
      $eout[j] \leftarrow newEout[j]$
    **end for**
    $hopCount \leftarrow hopCount + 1$
    Clear $newLive[]$ ▷ set all values to $False$
  **end while**
**end function**

than does TRG-Wu and so is expected to perform slower. Although [5] does not give an algorithm for the min-hop paths problem, their algorithm for shortest paths is easily modified to compute min-hop paths.

### A. Setup

The computational platform we use for our experiments is an Intel Knights Landing with maximum CPU clock cycle of 1.7 GHz and up to 384 GB RAM memory. All the codes are in C++. We used the g++ compiler with the following flags: $-std = c++11$ and $-O3$. The codes for the OSE data structure were obtained from the authors of [5] and used with no modification. The remaining algorithms were coded by us.

We use 11 of the datasets used by Wu et al. [5] in addition to an airline dataset. The airline dataset is taken from the Bureau of Transportation Statistics website [28]. The Wu datasets are taken from The KONECT Project website [29]. The airline dataset contains the information about flights in the US since 1987. For our experiments, we have used the data for January 2020. Self-loops were removed from all datasets in a preprocessing step and undirected graphs were converted to directed ones by replacing each undirected edge with 2 directed ones. In all datasets (except the airline dataset) the weight $w$ of every edge was set to 1 as was done in [5]).

Table I summarizes the properties of the datasets we used (the $|E|/|E_s|$ values are rounded to 2 digits):

TABLE I
DESCRIPTION OF THE DATASETS USED IN THIS PAPER

| dataset | $\|V\|$ | $\|E\|$ | $\|E_s\|$ | $\|E\|/\|E_s\|$ |
|---|---|---|---|---|
| airline | 351 | 599,183 | 5,704 | 105.046 |
| arxiv | 28,093 | 9,193,606 | 6,296,894 | 1.46 |
| conflict | 116,836 | 5,835,570 | 4,055,742 | 1.44 |
| digg | 30,360 | 86,203 | 85,247 | 1.01 |
| elec | 7,115 | 103,617 | 103,617 | 1 |
| enron | 86,978 | 2,269,980 | 594,912 | 3.82 |
| epin | 131,580 | 840,799 | 840,799 | 1 |
| fb | 63,731 | 1,634,070 | 1,634,070 | 1 |
| flicker | 2,302,925 | 33,140,017 | 33,140,017 | 1 |
| growth | 1,870,709 | 39,953,145 | 39,953,145 | 1 |
| slash | 51,083 | 139,789 | 130370 | 1.07 |
| youtube | 3,223,585 | 18,750,748 | 18,750,748 | 1 |

There are small differences from the statistics reported in [5]. This is possibly because the datasets have been updated since the work of [5].

### B. Complexity Comparison

In this section, we perform a complexity comparison between OSE, TRG_Wu and TRG_Ours. Time complexities for OSE and TRG_Wu are taken from [5]. Table II contains the complexities.Notations used here are the same as the ones used in section III. Here, $d_{max}$ means the maximum in-degree among all the nodes in the original graph $G$.

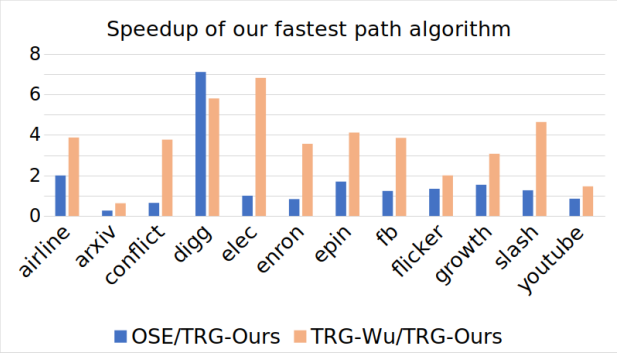


Fig. 2. Speedup obtained by our fastest path algorithm.

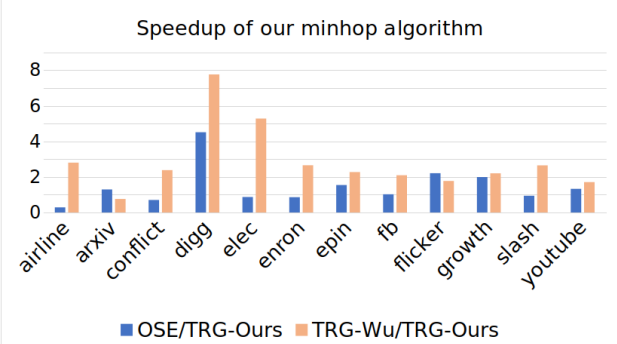


Fig. 3. Speedup obtained by our minhop algorithm.

TABLE II
COMPLEXITY COMPARISON BETWEEN OSE, TRG_WU AND TRG_OURS.

| | fastest | minhop |
|---|---|---|
| OSE | $O(\|T_{out}(s)\|(\|V\|+\|E\|))$ | $O(\|V\|+\|E\|log(d_{max}))$ |
| TRG_Wu | $O(\|V'\|+\|E'\|)$ | $O(\|V'\|+\|E'\|log\|V'\|)$ |
| TRG_Ours | $O(\|V'''\|+\|E'''\|)$ | $O(\|V'''\|+\|E'''\|)$ |

Based on table II, we can expect TRG_Ours to have better performance than TRG_Wu. Especially because based on our computations in section III we expect the upper bounds on $|V'|$ and $|E'|$ to be larger than the upper bounds on $|V'''|$ and $|E'''|$ in most cases.

Regarding the comparison between OSE and TRG_Ours, we replace $|V'''|$ with its upper bound derived in inequality 3 and $|E'''|$ with $2|E|$ as per inequality 4. The resulting complexities of TRG_Ours will be $O(V+3|E|)$ which simplifies to $O(V+|E|)$ for both fastest and minhop algorithms. Whether or not TRG_Our performs better than OSE depends on $|T_{out}(s)|$ and $d_{max}$.

It should be noted that, in the model we use, the edge updates are known in advance. This model is not proper for real time applications; but can be used in applications such as buying transportation tickets or analysing social networks.

### C. Results

For each algorithm, we measured the average time using 100 different randomly selected start vertices as done in the experiments of [5]. The speedup ((time taken by OSE or TRG-Wu)/time taken by TRG-Ours) obtained by our algorithms is shown visually in Figures 2 and 3.

As can be seen, TRG-Ours is faster than TRG-Wu on 11 of the 12 datasets for both the fastest paths and the min-hop problems. The speedup obtained by TRG-Ours on the fastest paths problem range from 0.63 to 6.81 and the speedup obtained by our minhop algorithm ranges from 0.76 to 7.76. On average, the speedup obtained by TRG_Ours over TRG_Wu is 3.63 on the fastest paths problem and 2.86 on the minhop problem across the 12 datasets. With respect to OSE, TRG-Ours is faster on 8 of the 12 datasets for the fastest paths problem with speedup ranging from 0.27 to 7.10. On the min-hop problem, TRG-Ours is faster than OSE on 6 of the 12 datasets used by us and the speedup obtained ranges from 0.29 to 4.51. On average, the speedup obtained by TRG_Ours over OSE is 1.65 on the fastest paths problem and 1.46 on the minhop problem across the 12 datasets. It should be noted that the OSE algorithms of [5] are limited to contact sequence temporal graphs in which no edge has a weight of 0; the TRG algorithms do not have this limitation. Zero weights can appear in many applications. As an example, we can consider a person who considers the least expensive way of getting to his destination using public transportation. He can use some discount on buses so some legs of travel can be free for him. Other similar applications can be considered. While Wu et al. [5] have stated that their OSE algorithms could be extended to handle the case when edges have a weight of 0, such an extension would, likely, increase the run time of the OSE algorithms.

## VII. CONCLUSION

We have developed a new time-respecting graph data structure TRG-Ours for contact sequence temporal graphs. This data structure has fewer vertices and edges than previously proposed TRG structures TRG-Wu and TRG-Zschoche. Benchmark experiments conducted for the fastest paths and min-hop paths problems indicates that TRG-Ours is superior to TRG-Wu as well as to the ordered sequence of edges structure OSE used by Wu et al. [5]. In fact, TRG-Ours outperformed TRG-Wu on all but one of our datasets on the fastest paths problem and for the min-hop problem. It also outperformed OSE on 8 of the 12 datasets on the fastest paths problem and 6 of 12 datasets on the non-hop problem. We did not experiment with TRG-Zschoche as this data structure uses more vertices and edges than does TRG-Wu and so it is expected to give inferior performance than TRG-Wu. We have also pointed out that TRG structures are superior to OSE on problems requiring only local information (shallow neighborhood search problems) such as one-hop and 2-hop neighbors. So, while Wu et al. [5] conclude that OSE is the data structure of choice for single-source all-destinations problems, our work indicates that TRG structures can be highly competitive with OSE while providing distinct advantages for shallow neighborhood search problems.

## REFERENCES


[1] C. Scheideler, "Models and techniques for communication in dynamic networks," in *Annual Symposium on Theoretical Aspects of Computer Science*. Springer, 2002, pp. 27–49.

[2] I. Stojmenovic, "Location updates for efficient routing in ad hoc networks," *Handbook of wireless networks and mobile computing*, vol. 8, pp. 451–471, 2002.

[3] P. Holme and J. Saramäki, "Temporal networks," *Physics reports*, vol. 519, no. 3, pp. 97–125, 2012.

[4] B.-M. Bui-Xuan, A. Ferreira, and A. Jarry, "Evolving graphs and least cost journeys in dynamic networks," in *WiOpt'03: Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks*, 2003, pp. 10–pages.

[5] H. Wu, J. Cheng, Y. Ke, S. Huang, Y. Huang, and H. Wu, "Efficient algorithms for temporal path computation," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 11, pp. 2927–2942, 2016.

[6] P. Zschoche, T. Fluschnik, H. Molter, and R. Niedermeier, "The complexity of finding small separators in temporal graphs," *Journal of Computer and System Sciences*, vol. 107, pp. 72–92, 2020.

[7] A. Zhao, G. Liu, B. Zheng, Y. Zhao, and K. Zheng, "Temporal paths discovery with multiple constraints in attributed dynamic graphs," *World Wide Web*, vol. 23, no. 1, pp. 313–336, 2020.

[8] M. S. Hassan, W. G. Aref, and A. M. Aly, "Graph indexing for shortest-path finding over dynamic sub-graphs," in *Proceedings of the 2016 International Conference on Management of Data*, 2016, pp. 1183–1197.

[9] A.-S. Himmel, M. Bentert, A. Nichterlein, and R. Niedermeier, "Efficient computation of optimal temporal walks under waiting-time constraints," in *International Conference on Complex Networks and Their Applications*. Springer, 2019, pp. 494–506.

[10] L. Roditty and U. Zwick, "Dynamic approximate all-pairs shortest paths in undirected graphs," *SIAM Journal on Computing*, vol. 41, no. 3, pp. 670–683, 2012.

[11] M. Alshammari and A. Rezgui, "An all pairs shortest path algorithm for dynamic graphs," *Comput. Sci*, vol. 15, no. 1, pp. 347–365, 2020.

[12] E. P. Chan and Y. Yang, "Shortest path tree computation in dynamic graphs," *IEEE Transactions on Computers*, vol. 58, no. 4, pp. 541–557, 2008.

[13] S. Cicerone, M. D'Emidio, and D. Frigioni, "On mining distances in large-scale dynamic graphs." in *ICTCS*, 2018, pp. 77–81.

[14] J. Hong, K. Park, Y. Han, M. K. Rasel, D. Vonvou, and Y.-K. Lee, "Disk-based shortest path discovery using distance index over large dynamic graphs," *Information Sciences*, vol. 382, pp. 201–215, 2017.

[15] K. Tretyakov, A. Armas-Cervantes, L. García-Bañuelos, J. Vilo, and M. Dumas, "Fast fully dynamic landmark-based estimation of shortest path distances in very large graphs," in *Proceedings of the 20th ACM international conference on Information and knowledge management*, 2011, pp. 1785–1794.

[16] A. Cvetkovski and M. Crovella, "Hyperbolic embedding and routing for dynamic graphs," in *IEEE INFOCOM 2009*. IEEE, 2009, pp. 1647–1655.

[17] C. Demetrescu and G. F. Italiano, "A new approach to dynamic all pairs shortest paths," *Journal of the ACM (JACM)*, vol. 51, no. 6, pp. 968–992, 2004.

[18] A. Clementi, R. Silvestri, and L. Trevisan, "Information spreading in dynamic graphs," *Distributed Computing*, vol. 28, no. 1, pp. 55–73, 2015.

[19] J. Calle, J. Rivero, D. Cuadra, and P. Isasi, "Extending aco for fast path search in huge graphs and social networks," *Expert Systems with Applications*, vol. 86, pp. 292–306, 2017.

[20] D. Chen, G. Navarro-Arribas, and J. Borrell, "On the applicability of onion routing on predictable delay-tolerant networks," in *2017 IEEE 42nd conference on local computer networks (LCN)*. IEEE, 2017, pp. 575–578.

[21] X. Zhang, F. T. Chan, H. Yang, and Y. Deng, "An adaptive amoeba algorithm for shortest path tree computation in dynamic graphs," *Information Sciences*, vol. 405, pp. 123–140, 2017.

[22] F. Brunelli, P. Crescenzi, and L. Viennot, "On computing pareto optimal paths in weighted time-dependent networks," *Information Processing Letters*, p. 106086, 2021.

[23] S. Riazi, S. Srinivasan, S. K. Das, S. Bhowmick, and B. Norris, "Single-source shortest path tree for big dynamic graphs," in *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 2018, pp. 4054–4062.

[24] P. Ni, M. Hanai, W. J. Tan, C. Wang, and W. Cai, "Parallel algorithm for single-source earliest-arrival problem in temporal graphs," in *2017 46th International Conference on Parallel Processing (ICPP)*. IEEE, 2017, pp. 493–502.

[25] Z. Ning, G. Dai, Y. Liu, Y. Ge, and J. Wu, "An improved index based on mapreduce for path queries in public transportation networks," in *2018 14th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)*. IEEE, 2018, pp. 919–926.

[26] D. Ferone, P. Festa, A. Napoletano, and T. Pastore, "Shortest paths on dynamic graphs: a survey," *Pesquisa Operacional*, vol. 37, no. 3, pp. 487–508, 2017.

[27] G. Nannicini and L. Liberti, "Shortest paths on dynamic graphs," *International Transactions in Operational Research*, vol. 15, no. 5, pp. 551–563, 2008.

[28] Bureau of transportation statistics. [Online]. Available: $https://www.transtats.bts.gov/OT\_Delay/OT\_DelayCause1.asp$

[29] The konect project. [Online]. Available: $http://konect.cc/$