Minimum Spanning Trees in Temporal Graphs

Silu Huang
Chinese University of Hong Kong
slhuang@cse.cuhk.edu.hk

Ada Wai-Chee Fu
Chinese University of Hong Kong
adafu@cse.cuhk.edu.hk

Ruifeng Liu
Chinese University of Hong Kong
rfliu@cse.cuhk.edu.hk

ABSTRACT

The computation of Minimum Spanning Trees (MSTs) is a fundamental graph problem with important applications. However, there has been little study of MSTs for temporal graphs, which is becoming common as time information is collected for many existing networks. We define two types of MSTs for temporal graphs, $\mathcal{MST}_a$ and $\mathcal{MST}_w$, based on the optimization of time and cost, respectively. We propose efficient linear time algorithms for computing $\mathcal{MST}_a$. We show that computing $\mathcal{MST}_w$ is much harder. We design efficient approximation algorithms based on a transformation to the Directed Steiner Tree problem (DST). Our solution also solves the classical DST problem with a better time complexity and the same approximation factor compared to the state-of-the-art algorithm. Our experiments on real temporal networks further verify the effectiveness of our algorithms. For $\mathcal{MST}_w$, our solution is capable of shortening the runtime from 10 hours to 3 seconds.

1. INTRODUCTION

The increasing popularity of social networks and the emergence of other large scale networks have attracted much related research interest in recent years. While most works so far focus on static networks, many networks of interest carry time information, which can be highly useful and important for network analysis. Such networks are termed temporal networks [15]. An important category of temporal networks is that of communication, including phone calls, emails, messaging, and social contact networks, where each edge represents a correspondence between two parties or two individuals. Another example is a network of airports, with edges labeled by the times of flight departures and arrivals. Other temporal networks include neural and brain networks, ecological networks, etc. A comprehensive survey of temporal networks is given in [15].

Consider the call detail records (CDR) generated by telecom companies as a temporal graph example. CDR records the calling party, the receiving party, the start time, the call duration and billing information. We thus assume that we are given a directed graph $G = (V, E)$, where each edge e in E is of the form $(u, v, t_u, \hat{t}_v, w)$, e links source vertex u to destination vertex v with start time t_u and end time $\hat{t}_v$, and carries a weight (or cost) of w. A path in G is a sequence of edges $\langle e_1, e_2, ..., e_k \rangle$ for some $k \leq |V|$, where the destination vertex of e_i is the source vertex of e_{i+1}. The path is **time-respecting** if the start time of e_{i+1} is not earlier than the end time of e_i [20]. Information flows along a path P in the network only if P is time respecting. The spread of information, rumor, or disease (epidemics) can follow only time respecting paths. Therefore, in a temporal graph, we need to consider time-respecting paths. This necessity often makes temporal graphs much harder to handle. For example, computing strongly connected components takes linear time in a static graph, but is NP-complete in a temporal graph [4].

The computation of minimum spanning trees (MST) is a classical problem in graph theory. One major application of MST is network broadcasting. In a social network, this corresponds to *information dissemination*, which is important in social campaigns, viral marketing, study of rumor spreading, etc. In view of this application, we define two different optimization problems for spanning trees in a temporal graph. The first is a time based optimization, which is for the fastest spread from a source vertex to each reachable vertex. We refer to such an optimal spanning tree as a $\mathcal{MST}_a$. The second is a cost based optimization, assuming each edge in the graph carries some weight (cost), a spanning tree with a minimum total weight is a $\mathcal{MST}_w$.

Given a non-temporal (static) directed graph $G = (V, E)$, the MST problem can be solved in $O(|E| + |V| \log |V|)$ time [12]. However, the MST problems for temporal graphs behave very differently. Our study shows that a $\mathcal{MST}_a$ can be computed by efficient algorithms with linear time. However, the problem of computing a $\mathcal{MST}_w$ is MAX-SNP hard. Hence, there is no polynomial time approximation scheme for computing $\mathcal{MST}_w$ unless $P = NP$. We transform the $\mathcal{MST}_w$ problem on temporal graphs to the classical Directed Steiner Tree (DST) problem. The state-of-the-art approximation algorithm for DST requires $O(n^i k^{2i})$ computation time, where n is the number of vertices, k is the number of terminals, and i is the number of iterations in the algorithm [8]. We derive an improved algorithm that runs in $O(n^i k^i)$ time with the same approximation ratio.

Our main contributions are summarized as follows. (1) We consider MSTs for temporal graphs. Based on the application of information dissemination, we define two types of MST, namely, $\mathcal{MST}_a$ for optimizing time, and $\mathcal{MST}_w$ for optimizing cost. (2) For computing a $\mathcal{MST}_a$, we propose linear time algorithms which are optimal and improve on the results in [4]. (3) We show that computing a $\mathcal{MST}_w$ is MAX-SNP hard. We transform the $\mathcal{MST}_w$ problem to the minimum Directed Steiner Tree problem (DST) with a conversion of the temporal graph to a static graph. (4) DST is a classical problem of independent interest. Our result improves on the best-known approximation algorithm in [8], achieving a better time complexity while having the same approximation guarantee.

SIGMOD'15, May 31–June 4, 2015, Melbourne, Victoria, Australia.

http://dx.doi.org/10.1145/2723372.2723717.

(5) We evaluate our approaches with a comprehensive set of experiments on real temporal networks. We show that our DST algorithm is up to 5 orders of magnitude faster than the state-of-the-art algorithm.

This paper is organized as follows. Section 2 contains our problem definitions. Section 3 describes the problem of $\mathcal{MST}_a$ and our algorithms for this problem. Section 4 is about the $\mathcal{MST}_w$ problem and its solutions. Our empirical study is reported in Section 5, related work is discussed in Section 6. We conclude in Section 7.

2. PROBLEM DEFINITION

In this section we introduce some definitions and notations, and two problems of minimum spanning trees in a temporal graph.

2.1 Notations for Temporal Graphs

Let $G = (V, E)$ be a temporal graph, where V is the vertex set and E is the temporal edge set. Let $|V| = n$ and $|E| = M$. We also denote V by $V(G)$ and E by $E(G)$. Each temporal edge $e \in E$ is labeled with a starting point u, an end point v, a starting time t_u from the starting point, an arrival time $\hat{t}_v$ at the end point, and a weight (or cost) w, denoted as $e = (u, v, t_u, \hat{t}_v, w)$ where $u, v \in V$, and $t_u, \hat{t}_v, w$ are non-negative real numbers. The starting point, end point, starting time, arrival time, and weight of e are also denoted by $s(e)$, $a(e)$, $t_s(e)$, $t_a(e)$, and $w(e)$, respectively, i.e., $s(e) = u$, $a(e) = v$, $t_s(e) = t_u$, $t_a(e) = \hat{t}_v$, and $w(e) = w$. We assume that $t_a(e) \geq t_s(e)$. Also, let $d(e)$ be the duration on temporal edge e, i.e., $d(e) = \hat{t}_v - t_u$, $d(e) \geq 0$. We say that e is an **in temporal edge** incident to v, and it is an **out temporal edge** incident to u. Let $\mathcal{N}_o(u)$ and $\mathcal{N}_i(u)$ denote the set of out temporal edges and the set of in temporal edges incident to vertex u, respectively, i.e., $\mathcal{N}_o(u) = \{e|s(e) = u, e \in E\}$,$\mathcal{N}_i(u) = \{e|a(e) = u, e \in E\}$. Note that G can be a cyclic graph.

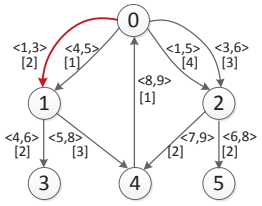
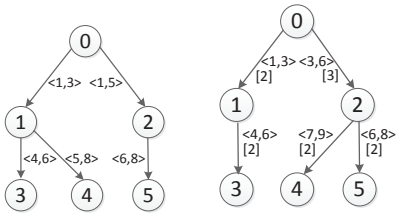


Figure 1: Temporal Graph G

EXAMPLE 1. *Figure 1 is a temporal graph. Each temporal edge is labeled with $\langle t_u, \hat{t}_v \rangle$ and $[w]$. In this example, we set the weight on each temporal edge as the duration,i.e., $\hat{t}_v - t_u$. However, note that in general, w can be set as any value. The graph can represent phone calls. Each vertex is an individual and an edge from u to v is a phone call from u to v. The temporal edge in red (bold) refers to $e = (0, 1, 1, 3, 2)$, meaning that there is a phone call from individual 0 to individual 1 starting from time 1, ending (arriving) at time 3 and the weight is 2. The graph may also represent flights, an edge from u to v is a flight from u to v.*

A temporal graph can be converted into a **static graph** by discarding the time information. Given a temporal graph $G = (V, E)$, let $G_S = (V_S, E_S)$ be the corresponding static graph. Then, $V_S = V$, $E_S = \{(u, v, w)|(u, v, t_u, \hat{t}_v, w) \in E\}$. $|V| = |V_S| = n$, let $|E_S| = m$.

$P = \{e_1, e_2, ..., e_k\}$ is called a **path** in a temporal graph if and only if $a(e_i) = s(e_{i+1})$ and $t_a(e_i) \leq t_s(e_{i+1})$, where $1 \leq i < k$. We say that P is a path from vertex $s(e_1)$ to $a(e_k)$. We call "$t_a(e_i) \leq t_s(e_{i+1})$" a time constraint. We call $t_a(e_k)$ the **arrival time** of P, and denote it by $\mathcal{A}(P)$. Given a time interval $[t_\alpha, t_\omega]$.



(a) $\mathcal{MST}_a$ (b) $\mathcal{MST}_w$

Figure 2: Minimum Spanning Tree of Figure 1

We say that a path P in G is **within** $[t_\alpha, t_\omega]$ if for each edge e_i in P, $t_\alpha \leq t_s(e_i)$ and $t_a(e_i) \leq t_\omega$. A vertex v is **reachable** from u in G in $[t_\alpha, t_\omega]$ if $v = u$ or there is a path from u to v within $[t_\alpha, t_\omega]$.

2.2 Two Types of Minimum Spanning Trees

Given a temporal graph $G = (V, E)$, a root $r \in V$, and a time interval $[t_\alpha, t_\omega]$. Let $\mathbb{P}(r, v)$ be the set of paths within $[t_\alpha, t_\omega]$ from r to v, if $P \in \mathbb{P}(r, v)$ has the earliest arrival time among all paths in $\mathbb{P}(r, v)$, we call the arrival time $\mathcal{A}(P)$ the **earliest arrival time** for v, and denote it by $\widetilde{\mathcal{A}}(v)$.

Let V_r be the set of vertices reachable from r in G in $[t_\alpha, t_\omega]$. A **spanning tree** $ST(r) = (V_{ST}, E_{ST})$ rooted at r is a subgraph of G with the following properties: (1) $V_{ST} = V_r$. (2) $\forall v \in V$ where $v \neq r$, there is exactly one in temporal edge incident to v; r has no in temporal edge, $|E_{ST}| = |V_r| - 1$. (3) $\forall v \in V$ where $v \neq r$, there exists a path from root r to v in $ST(r)$ within $[t_\alpha, t_\omega]$.

We define the following two types of minimum spanning trees in G and our problems are the computation of such MSTs.

- $\mathcal{MST}_a$ (Minimum spanning tree with earliest arrival times). A spanning tree $ST(r) = (V_{ST}, E_{ST})$ rooted at r is a $\mathcal{MST}_a$ if and only if $\forall v \in V_{ST}, v \neq r$, the path P from r to v in $ST(r)$ has the earliest arrival time for v, i.e. $\mathcal{A}(P) = \widetilde{\mathcal{A}}(v)$;
- $\mathcal{MST}_w$ (Minimum spanning tree with smallest total weight): A spanning tree $ST(r) = (V_{ST}, E_{ST})$ rooted at r is a $\mathcal{MST}_w$ if and only if $\sum_{e \in E_{ST}} w(e)$ is minimum among all spanning trees rooted at r. We refer to $\sum_{e \in E_{ST}} w(e)$ as the **weight** or **cost** of $ST(r)$ and also denote it by $\zeta(ST(r))$.

EXAMPLE 2 (MINIMUM SPANNING TREES). *Let us set the root r as vertex 0 in temporal graph G in Figure 1, we obtain $V_r = \{1, 2, 3, 4, 5\}$. Figure 2(a) is a $\mathcal{MST}_a$ for G, while Figure 2(b) is a $\mathcal{MST}_w$ for G, rooted at vertex 0. In Figure 2(a), the arrival times for vertices $1, 2, 3, 4, 5$ are $3, 5, 6, 8, 8$, respectively, which are the corresponding earliest arrival times in G. If the graph represents phone calls, these are the earliest times for a message to be transmitted from 0 to the other individuals. In Figure 2(b), the total weight of all edges is $2+3+2+2+2 = 11$, which is the smallest among all possible spanning trees of G rooted at 0. If the graph represents phone calls, then the minimum cost to transmit a message from 0 to all other individuals is 11. If the graph represents flights, then 11 is the minimum cost to distribute some goods from airport 0 to all other airports.*

2.3 Motivation for $\mathcal{MST}_a$ and $\mathcal{MST}_w$

Some discussion about our problem definitions is in order. For non-temporal graphs, a branching is a forest in which each edge is directed towards a different vertex. Given a directed graph $G = (V, E)$, a branching $G' = (V', E')$ in G is a spanning arborescence, or a directed spanning tree (DSPT), of G iff $|E'| = |V| - 1$.

A minimum DSPT (MDSPT), has the smallest total edge weight among all DSPTs [10, 11, 25]. Clearly, G contains a DSPT iff all vertices in G are reachable from some vertex r in G. Hence, previous studies on MDSPT assume that all vertices in V are reachable from some root vertex r in V. Then, either the root vertex is prescribed or it is not prescribed. In the early work by Edmonds, application of a least costly way to have a message communicated from the root to all other nodes is described, where the root is prescribed [11]. To our knowledge, no application is known for the case where the root is not prescribed. Thus, as in the best known algorithm for MDSPT in [12], we deal with a prescribed root r.

In real temporal graphs, different vertices may reach different sets of vertices, and a given vertex r typically cannot reach all vertices. Thus, we ensure the reachability condition by considering the subgraph induced by all vertices reachable from r.

Our work is motivated by a number of applications. Information flows follow time respecting paths in a network, and a $\mathcal{MST}_a$ from a source vertex s tells us the earliest time that a piece of information can be dispatched to other sites. $\mathcal{MST}_a$ is also useful for the study of epidemiology, the spread of infectious diseases (or computer virus), when the network is about individual contacts (or a computer network) [3, 20]. Another application is the schedule of transportation. $\mathcal{MST}_a$ gives a schedule of transports for distribution of goods from a source location with the earliest arrival time for each destination.

Information dissemination is the major motivating application for the study of temporal networks in [20]. An important application of $\mathcal{MST}_w$ is to minimize the cost of message communication or information propagation, where each edge represents a link of direct communication from one party to another, with a weight that indicates the cost of the communication, provided that the cost is additive. E.g. phone communication involves some cost for each call. Given a time window $[t_\alpha, t_\omega]$, $\mathcal{MST}_w$ gives us the minimum information dissemination cost from a person. As the time window slides forward, we can predict the minimum cost for the future. The transportation problem cited in [20] is another application. An example is a network of airports with edges corresponding to flights, and each flight involves some cost. A $\mathcal{MST}_w$ minimizes the total cost to transport some given resource from a given location r to all destinations. $\mathcal{MST}_w$ can also be useful for clustering [2, 33], which is related to community search in social networks.

$\mathcal{MST}_a$ bears some resemblance to a shortest path tree in a non-temporal multi-graph in that the path from the root to each vertex is optimized. The difference is that for a shortest path tree, the total length of edges in the path from r to each vertex v is minimized, while in a $\mathcal{MST}_a$, the arrival time at v is minimized.

$\mathcal{MST}_a$ also solves the problem of minimizing the maximum arrival time since the arrival time for each vertex is minimized.

3. TIME-MINIMUM SPANNING TREE

We consider the computation of a $\mathcal{MST}_a$ in a temporal graph. Given temporal graph G, an interval $[t_\alpha, t_\omega]$, and root r, let $\tilde{\mathcal{A}}(v)$ be the earliest arrival time from root r to reachable vertex v in V_r. We can show that there exists a $\mathcal{MST}_a$ rooted at r which is made up of paths from r to all reachable vertices with the earliest arrival times.

LEMMA 1. *Given a temporal graph $G = (V, E)$ and the root r, there must exist a minimum spanning tree ($\mathcal{MST}_a$) rooted at r.*

In the following we propose two efficient algorithms for $\mathcal{MST}_a$ with linear complexity. Let us first present the data format for the temporal graph. We assume that a temporal edge is of the form $e = (u, v, t_u, \hat{t}_v, w)$. We are given a list of M temporal edges, $\{e_1, e_2, ..., e_i, ..., e_M\}$, as the input raw data. Assume the edges are ordered by non-decreasing starting time $t_s(e_i)$. We call this raw data a *chronological edge list* input, denoted by $\mathcal{G} = [e_1, e_2, ..., e_M]$. The first algorithm is given in Algorithm 1.

Algorithm 1: $\mathcal{MST}_a(r, \mathcal{G}, t)$

Input : A temporal graph $\mathcal{G}$ (chronological edge list), root r, time interval $[t_\alpha, t_\omega]$;
Output : $\mathcal{MST}_a$ (given by $\mathcal{P}()$)

```
1 Initialize: A(u) = ∞, P(u) = r ∀u ≠ r, u ∈ V; A(r) = t_α;
2 for e = (u, v, t_u, t̂_v, w) ∈ G do
3     if (t_u ≥ A(u)) & (t̂_v < A(v)) & (t̂_v ≤ t_ω) then
4         A(v) ← t̂_v;
5         P(v) ← u;
```

EXAMPLE 3. *Consider the graph in Figure 1. We set the time interval of interest as $t_\alpha = 0$, $t_\omega = \infty$. The first edge in the input is $(0, 1, 1, 3, 2)$, we update $\mathcal{A}(1)$ to 3, and $\mathcal{P}(1)$ to 0. The next input is $(0, 2, 1, 5, 4)$, we set $\mathcal{A}(2)$ to 5, and $\mathcal{P}(2)$ to 0, according to Lines 4-5 in Algorithm 1. For the next inputs of $(0, 2, 3, 6, 3)$, $(0, 1, 4, 5, 1)$, the if condition at Line 3 is not met, no update is triggered. Similarly, we process the remaining inputs and the $\mathcal{MST}_a$ in Figure 2(a) is generated.*

THEOREM 1. *Algorithm 1 returns a $\mathcal{MST}_a$ in $O(M)$ time given $t_s(e) \neq t_a(e), \forall e \in E$. ($M = |E|$)*

PROOF. Let S_T be the set of earliest arrival time for vertices in V_r. Let $S_T = \{t_1, t_2, ...t_i, ...t_{|S_T|}\}$, where $t_j < t_{j+1}, 1 \leq j \leq |S_T|-1$. First we prove that $\mathcal{A}(u)$ in Algorithm 1 correctly records the earliest arrival time for each vertex $u \in V_r$ whose $\tilde{\mathcal{A}}(u) = t_1$. Vertex u must be one hop from root r. Suppose $\{e\}$ is the earliest arrival path from r to u, where $t_a(e) = t_1$, then $\mathcal{A}(u)$ must be set to t_1 when processing e in line 2 since $t_s(e) \geq t_\alpha$ and $\mathcal{A}(r)$ is set to be t_α initially, and note that t_α is a lower bound of starting times.

We prove by induction. Assume that the final $\mathcal{A}(v)$ in Algorithm 1 correctly records the earliest arrival time for any vertex v with $\tilde{\mathcal{A}}(v) < t_i$. We need to prove that $\mathcal{A}(v')$ correctly record the earliest arrival time for any vertex v' with $\tilde{\mathcal{A}}(v') = t_i$. Let $P = \{e_1, e_2, ..., e_k\}$ be a path from r to v' with earliest arrival time $\tilde{\mathcal{A}}(v') = t_i$; let $v_j = a(e_j)$, where $1 \leq j < k$. From inequality $t_a(e_{k-1}) \leq t_s(e_k) < t_a(e_k) = t_i$, we have $t_a(e_{k-1}) < t_i$. Also, $\tilde{A}(v_{k-1}) \leq t_a(e_{k-1})$, since $\tilde{\mathcal{A}}(v_{k-1})$ is the earliest arrival time for vertex $v_{k-1} = a(e_{k-1})$. Thus, $\tilde{\mathcal{A}}(v_{k-1}) \leq t_i$. By the induction hypothesis, the final $\mathcal{A}(v_{k-1}) = \tilde{\mathcal{A}}(v_{k-1})$. Note that when processing e_k at Line 2, $\mathcal{A}(v_{k-1})$ has already been set to $\tilde{\mathcal{A}}(v_{k-1})$ since $t_s(e_k) \geq t_a(e_{k-1}) > t_s(e_{k-1})$ and temporal edges are processed in ascending order of $t_s(e), e \in E$. According to Line 3, $\mathcal{A}(v')$ will be set to t_i when edge e_k is processed. Hence, Algorithm 1 correctly returns a $\mathcal{MST}_a$ as recorded in $\{\mathcal{P}(v) | v \in V_r\}$.

Finally, Algorithm 1 needs only one scan of the temporal edges, thus, the time complexity is $O(M)$. □

Algorithm 1 also applies when the input temporal graph consists of edges sorted in ascending order of the arrival time instead of the starting time. This can be proved by induction, similar to the proof of Theorem 1, as follows. Following the notations in the induction in the proof of Theorem 1, let $P = \{e_1, e_2, ..e_{k-1}, e_k\}$ be a path from r to v' with the earliest arrival time $\tilde{\mathcal{A}}(v') = t_i = t_a(e_k)$; let $v_j = a(e_j)$, where $1 \leq j < k$. According to the induction hypothesis, $\mathcal{A}(v_{k-1})$ is set to $\tilde{\mathcal{A}}(v_{k-1})$ correctly since

$t_a(e_{k-1}) < t_i$. Note that $\mathcal{A}(v_{k-1})$ must be set to $\tilde{\mathcal{A}}(v_{k-1})$ before the occurrence of e_k since $t_a(e_k) > t_a(e_{k-1}) \geq \tilde{A}(v_{k-1})$ and temporal edges are sorted in non-descending order of arrival time $t_a(e)$. Thus, after processing e_k in Line 2, $\mathcal{A}(v_k)$ equals $t_a(e_k)$, i.e., $\mathcal{A}(v') = \tilde{\mathcal{A}}(v')$. Hence the algorithm is correct.

However, in Theorem 1, we have the constraint that $t_s(e) \neq t_a(e), \forall e \in E$. That is, Algorithm 1 requires that the duration at each edges is non-zero. In some real temporal graphs, such as the graphs for publications, facebook wall posting, or email communication, the edge durations are given as zero, in such a case the algorithm may fail, as shown in the following example.

EXAMPLE 4. *The temporal graph G_0 in Figure 3 is an example where Algorithm 1 cannot correctly return a $\mathcal{MST}_a$. Suppose start time is 0, the edges of the input graph are sorted as follows:* $(0, 1, 1, 1, 0)$, $(2, 0, 2, 2, 0)$, $(3, 1, 2, 2, 0)$, $(1, 4, 3, 3, 0)$, $(3, 2, 4, 4, 0)$, $(4, 3, 4, 4, 0)$. *In this case, when* $(3, 2, 4, 4, 0)$ *is visited,* $\mathcal{A}(3) = \infty$, *the condition at Line 3 will fail and edge* $(3, 2)$ *will not be added to the MST. Hence, vertex 2 will be missing in the resulting tree.*

Hence, we need another algorithm to deal with the general case where an edge duration can be zero, i.e., $t_s(e) \leq t_a(e), \forall e \in E$. For this problem, we first transform the input graph G from the raw stream format into the following *sorted adjacency edge list format*. Each input line corresponds to the out temporal edges incident to one vertex $u \in V$, in the following form: $u, < v_1, t_1, \hat{t}_1, w_1 >$, ..., $< v_i, t_i, \hat{t}_i, w_i >$, ... In each line, edges $e \in \mathcal{N}_o(u)$ are sorted by the start times, $t_s(e)$, in non-increasing order, $\forall u \in V$. That is, $t_i \geq t_{i+1}$, where $1 \leq i < |\mathcal{N}_o(u)|$. We limit the edges so that the start time is earliest t_α and end time is latest t_ω, we call the resulting edge list input $G[t_\alpha, t_\omega]$. The transformation takes $O(M)$ time.

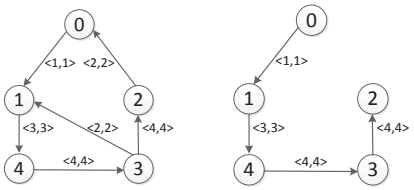


Figure 3: G_0 with zero durations and its $\mathcal{MST}_a$

We propose another efficient algorithm, Algorithm 2, with linear time complexity to solve the general $\mathcal{MST}_a$ problem. The input to the algorithm adopts the sorted adjacency edge list format above. In Algorithm 2, we assume that for each vertex $u \in V$, temporal edges incident to u are sorted in non-ascending order of $t_s(e)$ and stored in array $\mathbb{N}_o(u)$. We only need to scan $\mathbb{N}_o(u)$ once and $pos(u)$ is kept to indicate the current position in $\mathbb{N}_o(u)$. Only smaller arrival times are recorded as shown in Lines 4 and 5, and the time constraint for each path is maintained by Line 9. The complexity for Algorithm 2 is bounded by $O(M)$. Our empirical study also verifies the efficiency of this proposed algorithm.

THEOREM 2. *Algorithm 2 returns a $\mathcal{MST}_a$ in $O(M)$ time, where $M = |E|$.*

PROOF. We first prove the correctness. That is, given temporal graph G and root r, the path from r to v in $\mathcal{MST}_a$, $\forall v \in V, v \neq r$, results in earliest arrival time of v in G, where $\mathcal{MST}_a$ is the resulting minimum spanning tree from Algorithm 2. Let the path from r to v in $\mathcal{MST}_a$ be $P = \{e_1, e_2, ...e_k\}$, where $v = a(e_k)$. Note that $t_a(e_k)$ is the output $\mathcal{A}(v)$ of Algorithm 2. We say that vertex v is at level k of $\mathcal{MST}_a$. We prove by induction on k.

Algorithm 2: $\mathcal{MST}_a(r, G)$

Input : A temporal graph $G[t_\alpha, t_\omega]$ (in sorted adjacency edge list format), root r, time interval $[t_\alpha, t_\omega]$;
Output : $\mathcal{MST}_a$ (given by $\mathcal{P}()$)

```
1  Initialize: A(u) = ∞, pos(u) = 1, ∀u ∈ V; push < r, r, t_α > onto stack S;
2  while S ≠ ∅ do
3      < u, v, t̂_v > ← top(S), pop(S);
4      if t̂_v < A(v) then
5          A(v) ← t̂_v;
6          P(v) ← u;
7          if pos(v) ≤ |N_o(v)| then
8              < v, v', t_v, t̂_v', w' > ← N_o(v)[pos(v)];
9              while pos(v) ≤ |N_o(v)| and A(v) ≤ t_v do
10                 push < v, v', t̂_v' > onto stack S;
11                 pos(u) + +;
12                 < v, v', t_v, t̂_v', w' > ← N_o(v)[pos(v)];
```

When v is at level 1, i.e., $k = 1$ and $P = \{e_1\}$, then $t_a(e_1)$ must be the earliest arrival time for $a(e_1)$, i.e., $t_a(e_1) = \tilde{\mathcal{A}}(a(e_1))$. This is because all temporal edges meeting the time constraint will be visited as described in Line 9 of Algorithm 2 and we always keep the smallest arrival time for each vertex as described in Line 4 of Algorithm 2.

Assume that when $k < i$, $t_a(e_k)$ is the earliest arrival time for vertex v, i.e., $t_a(e_k) = \tilde{\mathcal{A}}(v)$, where v is at level k of $\mathcal{MST}_a$. We want to prove that $t_a(e_i)$ is the earliest arrival time for vertex $v' = a(e_i)$ when v' is at level i, i.e., when $k = i$. Suppose there exists a path from r to v' in G, $\mathbb{P} = \{e'_1, e'_2, ..., e'_l\}$, where $a(e'_l) = v'$ and $t_a(e'_l) < t_a(e_i)$. There must exist one vertex $\tilde{u} = s(e'_j)$ such that $\tilde{u}$ is at level h of $\mathcal{MST}_a$, where $h < i$. Then according to the hypothesis, $\mathcal{A}(\tilde{u})$ in Algorithm 2 is equal to $\tilde{\mathcal{A}}(\tilde{u})$. Thus, according to Line 9 in Algorithm 2, e'_j must be visited since $t_s(e'_j) \geq t_a(e'_{j-1}) \geq \tilde{\mathcal{A}}(\tilde{u})$, and $\mathcal{A}(a(e'_j))$ must be no more than $t_a(e'_j)$ according to Lines 4-5. Repeat the above analysis, we can conclude that $\mathcal{A}(a(e'_l))$, i.e., $\mathcal{A}(v')$ (or $t_a(e_i)$), must be no more than $t_a(e'_l)$, which contradicts with $t_a(e'_l) < t_a(e_i)$. Hence, $t_a(e_i)$ is the earliest arrival time for vertex $v' = a(e_i)$ when v' is at level i, i.e., $k = i$.

Next we analyze the time complexity for Algorithm 2. There are at most M outer while loops (Line 2), hence, the time required for Lines 3-8 is bounded by $O(M)$. The inner (Line 9) and outer while loops (Line 2) together take up one scan of the temporal graph. Hence, the total complexity is bounded by $O(M)$. □

4. WEIGHT-MINIMUM SPANNING TREES

In this section, we study the problem of computing a $\mathcal{MST}_w$, which turns out to be far more difficult than computing a $\mathcal{MST}_a$. In the following we prove the hardness of the $\mathcal{MST}_w$ problem even for approximation solutions. Then we show how we can transform the problem to the classical directed Steiner Tree problem (DST) on non-temporal graphs. While the transformed problem can be solved by a state-of-the-art approximation algorithm with some post-processing steps, scalability is a major problem, we propose an algorithm with better time complexity while preserving the approximation guarantee.

4.1 Hardness of the $\mathcal{MST}_w$ Problem

In this section, we discuss the hardness of computing a $\mathcal{MST}_w$. The problem of $\mathcal{MST}_w$ is: given a temporal graph $G = (V, E)$,

a root vertex r and an interval $[t_\alpha, t_\omega]$, compute a $\mathcal{MST}_w$. Our main results are that the problem is NP-hard and also MAX-SNP hard. A proof of the following is given in the Appendix.

THEOREM 3. *Computing a MST_w is NP-hard.*

A polynomial time approximation scheme (PTAS) is an algorithm which takes an instance of an optimization problem and a parameter $\epsilon > 0$ and, in polynomial time, produces a solution that is within a factor $1+\epsilon$ of being optimal for a minimization problem (or $1-\epsilon$ for maximization problems). A MAX-SNP hard problem does not have a PTAS unless $P = NP$ [1].

THEOREM 4. *Computing a $\mathcal{MST}_w$ is MAX-SNP hard.*

Given a (f, g, α, β) L-reduction from problem P to problem Q[1], and a $(1 \pm \varepsilon)$-approximation algorithm for Q, we obtain a polynomial-time $(1 \pm \delta)$-approximation algorithm for P where $\delta = \alpha\beta\varepsilon$ [23, 18]. The proof of Theorem 4 first converts the transformation in the proof of Theorem 3 into an L-reduction $\mathbb{X}$ from the maximum leaf spanning tree problem to the $\mathcal{MST}_w$ problem. Next we adopt the L-reduction $\mathbb{Y}$ in [13], which reduces the MAX-SNP complete problem of Min Dominating set-B to the problem of maximum leaf spanning tree. We form an L-reduction by a composition of the functions in $\mathbb{Y}$ and the corresponding functions in $\mathbb{X}$. Thus, we obtain an L-reduction from the minimum dominating set problem to the $\mathcal{MST}_w$ problem. From Theorem 4, there is no polynomial time approximation scheme for computing a $\mathcal{MST}_w$ in a temporal graph, unless $P = NP$.

4.2 Problem Transformation

We have shown that computing $\mathcal{MST}_w$ is a hard problem and remains hard for approximation algorithms. Here, we aim for an approximation algorithm with good scalability. In this section, we show that this problem can be transformed to the minimum directed Steiner tree problem (DST) on a static graph. Hence, we can apply the best known DST approximation algorithm for solving the $\mathcal{MST}_w$ problem. We first introduce the graph transformation. The directed Steiner tree problem is defined as follows.

Minimum Directed Steiner Tree (DST) : Let $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ be a directed static graph, where $\mathbb{V}$ is the vertex set and $\mathbb{E}$ is the edge set with weights (or costs) on edges. Given a set of terminals $X \subset \mathbb{V}$ and a root vertex r, the minimum directed Steiner tree problem (DST) asks for a minimum-cost tree $T = (V_T, E_T)$ which contains a path from root r to each terminal $x \in X$, such that $\sum_{e \in E_T} w(e)$ is minimum among all such trees. We refer to $\sum_{e \in E_T} w(e)$ as the **cost of** T and also denote it by $\zeta(T)$.

In the following, we present a graph transformation algorithm. For clarity, in the remaining discussions about $\mathcal{MST}_w$, we assume that $[t_\alpha, t_\omega] = [0, \infty]$ and $V_r = V$. It is straightforward to extend the discussions to the general case.

Graph Transformation: Given a temporal graph $G = (V, E)$, let $E_{in}(v)$ be v's incoming edge set for each vertex $v \in V$. If $v \neq r$, let $\mathcal{T}(v)$ be the set of v's arrival time instances, i.e., $\mathcal{T}(v) = \{\hat{t}_v | e = (u, v, t_u, \hat{t}_v, w) \in E_{in}(v)\} = \{\hat{t}_v^1, \hat{t}_v^2, ..\}$. For root r, let $\mathcal{T}(r) = \{0\}$, and $\hat{t}_r^1 = 0$. The following steps transform the temporal graph $G = (V, E)$ into a directed weighted graph $\mathbb{G} = (V_\mathbb{G}, E_\mathbb{G})$, where an edge e from vertex u to v with a weight of w is given by $e = (u, v, w)$.

[1] f, g are functions such that if p is an instance of P, then $f(p)$ is an instance of Q, and if q is a solution to $f(p)$, then $g(q)$ is a solution to P.

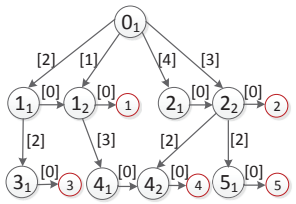


Figure 4: The Transformed Graph $\mathbb{G}$ of Figure 1

Step 1 Construct $V_\mathbb{G}$ in $\mathbb{G}$. For each vertex $v \in V$,

(a) Create $|\mathcal{T}(v)|$ **virtual vertices** in $\mathbb{G}$, i.e., $\{v_1, v_2, ...v_{|\mathcal{T}(v)|}\}$ with non-decreasing time instances: $\hat{t}_v^1 \leq \hat{t}_v^2 ... \leq \hat{t}_v^{|\mathcal{T}(v)|}$.

(b) If $v \neq r$, create a dummy vertex v in $\mathbb{G}$. Set $\hat{t}_v^{|\mathcal{T}(v)|+1} = \infty$

Step 2 Construct $E_\mathbb{G}$ in $\mathbb{G}$.

(a) For each vertex $v \in V$, create $|\mathcal{T}(v)|$ **virtual edges** $(v_1, v_2, 0)$, $(v_2, v_3, 0)$ ... $(v_{|\mathcal{T}(v)|-1}, v_{|\mathcal{T}(v)|}, 0)$ and $(v_{|\mathcal{T}(v)|}, v, 0)$ in $\mathbb{G}$.

(b) For each edge $e = (u, v, t_u, \hat{t}_v, w) \in E$

– find virtual vertex $u_i \in V_\mathbb{G}$ with time instance $\hat{t}_u^i \leq t_u$ and $\hat{t}_u^{i+1} > t_u$; find virtual vertex $v_j \in V_\mathbb{G}$ with time instance $\hat{t}_v^j = \hat{t}_v$. Create **solid edge** (u_i, v_j, w) in $\mathbb{G}$.

EXAMPLE 5 (GRAPH TRANSFORMATION). *According to the 2 steps above, temporal graph G in Figure 1 is transformed into static graph $\mathbb{G}$ in Figure 4. [Step 1]: take vertex 1 in G as an example, vertices 1_1 and 1_2 are created in Figure 4 corresponding to arrival times of 3 and 5, respectively, according to Step $1(a)$; at the same time dummy vertex 1 is created by Step $1(b)$. [Step 2]: take vertex 1 in G as an example, virtual edges $(1_1, 1_2, 0)$ and $(1_2, 1, 0)$ are created in Figure 4, as described in Step $2(a)$; meanwhile take temporal edge $(1, 3, 4, 6, 2)$ in G as an example, according to Step $2(b)$ we find the starting point 1_1 in $\mathbb{G}$ (since 1_1 corresponds to time 3, 1_2 corresponds to time 5 and $3 < 4 < 5$); then find the end point 3_1 (since 3_1 corresponds to time 6); we connect 1_1 to 3_1 by a solid edge with weight 2.*

LEMMA 2. *Given a temporal graph $G = (V, E)$, the transformed graph $\mathbb{G}$ contains $O(|E|)$ vertices and $O(|E|)$ edges.*

The following theorem says that after the graph transformation, we can apply a solution for the minimum DST problem and then we can obtain the $\mathcal{MST}_w$ result from the DST result.

THEOREM 5 (PROBLEM TRANSFORMATION). *A $\mathcal{MST}_w$ with root r in a temporal graph G can be derived in linear time from a minimum Directed Steiner Tree (DST) rooted at r in G's corresponding transformed graph $\mathbb{G}$, where the terminal set is the dummy vertex set S created in $\mathbb{G}$, i.e., $X = S$.*

In the next subsection, we describe the best-known approximation algorithm for DST and how to use it for the $\mathcal{MST}_w$ problem.

4.3 Approximation Algorithm on the Transformed Graph

The best known approximation algorithm for the minimum directed Steiner tree problem in static graph is shown in Algorithm 3 [8], which greedily picks the lowest density subtrees in the computation. Note that the input graph for Algorithm 3 is a **transitive closure**. That is, given a directed static graph $\mathbb{G} = (\mathbb{V}, \mathbb{E})$, we first need preprocessing to get the transitive closure $\mathbb{G}_t$ so that there is

Algorithm 3: $A^i(k, r, X)$

Input : Terminal set X, number of terminals k to be covered, root r, level number i, transitive closure $\mathbb{G}_t$;
Output : A tree T with height i rooted at r covering k terminals in X

1 Initialize: $T \leftarrow \emptyset$;
2 **if** $i = 1$ **then**
3 **while** $k > 0$ **do**
4 $(r, v) \leftarrow arg_{(r,v)} \min cost(r, v), \forall v \in X$;
5 $T \leftarrow T \cup (r, v)$; $k \leftarrow k - 1$; $X \leftarrow X - \{v\}$;
6 **else**
7 **while** $k > 0$ **do**
8 $T_{BEST} \leftarrow \emptyset$; $den(T_{BEST}) \leftarrow \infty$;
9 **for** *each vertex* $v \in \mathbb{V}$ *and each* k', $1 \leq k' \leq k$ **do**
10 $T' \leftarrow A^{i-1}(k', v, X) \cup (r, v)$;
11 **if** $den(T_{BEST}) > den(T')$ **then**
12 $T_{BEST} \leftarrow T'$;
13 $T \leftarrow T \cup T_{BEST}$; $k \leftarrow k - |X| \cap V(T_{BEST})$;
 $X \leftarrow X - V(T_{BEST})$;
14 return T;

an edge (u, v) whenever v is reachable from u, and the weight of (u, v) is the weight of the shortest path from u to v.

Algorithm 3 is a recursive function on i, where i is the level number, r is the root, X refers to the terminal set and k denotes how many terminals need to be covered in this function. The approximate result for DST is derived from $A^i(|X|, r, X)$. When $i = 1$, k edges from root r to terminals are selected with smallest weights. Since the input graph is a transitive closure $\mathbb{G}_t$, the selected edges correspond to k shortest paths from the root to k terminals in $\mathbb{G}$. When $i \neq 1$, Algorithm 3 recursively calls $A^{i-1}(k', v, X)$ for different v and k' as shown in Lines 9-10, picking the v and k' whose resulting T' is of the best density. The density is given by $den(T') = cost(T')/k(T')$, where $cost(T')$ is the total weight of tree T' and $k(T')$ is the number of covered terminals in T'. As proved in [8], for $i > 1$ iterations, the approximation ratio of $A^i(k, r, X)$ is bounded by $i^2(i-1)k^{1/i}$ and the time complexity is $O(|\mathbb{V}|^i k^{2i})$, which is still very costly.[2] In the next 2 subsections, we introduce strategies that lead to a much faster algorithm.

After getting the approximation spanning tree from Algorithm 3, we need the following two **postprocessing** steps to get the corresponding spanning tree in the original temporal graph G to approximate $\mathcal{MST}_w$. We should point out that the first step is also required for the minimum DST problem in $\mathbb{G}$.

Step 1 Get corresponding spanning tree $\mathbb{T}$ in transformed graph $\mathbb{G}$:

(a) Replace edges in the resulting spanning tree of Algorithm 3 with shortest paths in $\mathbb{G}$.

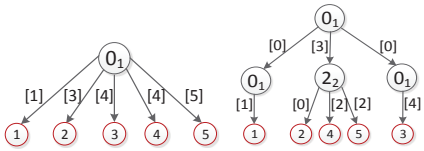
(b) Each vertex keeps only one incoming edge. If there are multiple incoming edges, choose one with the smallest weight.

Step 2 Get corresponding spanning tree in temporal graph G:

(a) For $\mathbb{T}$ from Step 1, delete virtual edges by merging all virtual and dummy vertices for each vertex in G into one vertex. According to the transformation scheme in Section 4.2, restore the corresponding temporal edges and vertices in G.

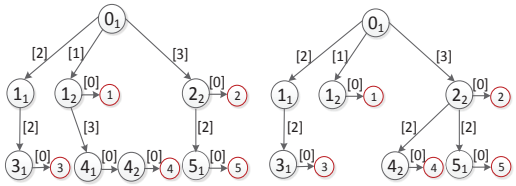
(b) Each vertex keeps only one incoming temporal edge, the one with the smallest arrival time among such edges.

[2]In [8], when $i = \log k$, an approximation ratio of $O(\log^2 k)$ is derived based on an erroneous result in [32], with a corrected result in [14], the ratio is given by $O(\log^3 k)$.



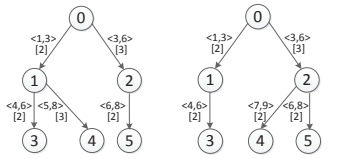
(a) $i = 1$ (b) $i = 2$

Figure 5: Resulting Approximation Tree in Algorithm 3



(a) $i = 1$ (b) $i = 2$

Figure 6: Postprocess Step 1



(a) $i = 1$ (b) $i = 2$

Figure 7: Postprocess Step 2

EXAMPLE 6 (ALGORITHM 3). *Take the graph in Figure 4 as the given static graph $\mathbb{G}$ before preprocessing for Algorithm 3 with $X = \{1, 2, 3, 4, 5\}$, $k = 5$ and $r = 0$. Figure 5(a) is the resulting spanning tree for $i = 1$ with total weight 17. That is, pick the shortest path from root 0 to each terminal vertex in $\mathbb{G}$. Figure 5(b) is the resulting spanning tree for $i = 2$ with total weight 12. For each $1 \leq k' \leq 5$ and each vertex in Figure 4, we calculate the density for T' in Lines 9-12, and pick $v = 0, k' = 1$ whose resulting T' has the smallest density $den(T') = \frac{1}{1} = 1$. Then, update T, k and X according to Line 13 and go to the next while iteration, where $v = 2_2, k' = 3$ are chosen with density $den(T') = \frac{3+2+2}{3}$. Finally $v = 0, k' = 1$ are chosen with density $den(T') = \frac{4}{1} = 4$.*

Postprocess: *Figure 6 is the resulting spanning tree after Step 1 in $\mathbb{G}$ (Figure 4). Figure 7 is the resulting temporal spanning tree after Step 2 in G (Figure 1). Take vertex 1 in G as an example, there are two incoming temporal edges after Step2(a) and temporal edge $(0, 1, 1, 3, 2)$ is chosen according to Step 2(b).*

Though we have a guarantee on the approximation for the DST problem, it does not translate to the $\mathcal{MST}_w$ problem directly since the postprocessing changes the cost or weight of the final spanning tree. The following theorem guarantees that the approximation ratio still holds.

THEOREM 6. *Given a temporal graph $G = (V, E)$, the DST-based method with Algorithm 4 provides an $i^2(i-1)k^{1/i}$ approximation to the $\mathcal{MST}_w$ problem in $O(|E|^i k^i)$ time, where $k < |V|$.*

Clearly the postprocessing steps can be executed in linear time. With the graph transformation steps in Section 4.2, the DST algorithm and the postprocessing steps above, we can solve the $\mathcal{MST}_w$ problem. The time complexity is dominated by Algorithm 3, which is still high and severely limits the number of levels it can process. We are interested in reducing the runtime while preserving the answer and hence the approximation guarantee. This problem will be studied in the next subsection.

4.4 An Improved DST Algorithm

We begin with some observations about Algorithm 3. Consider $A^i(k, r, X)$, in order to find T_{BEST} in Lines $7-10$, we need to process each $A^{i-1}(k', v, X) \cup (r, v)$, $\forall v \in \mathbb{V}, 1 \leq k' \leq k$. Some further analysis can help to greatly reduce this complexity. First we need some definitions.

DEFINITION 1. *The **process tree** of Algorithm 3 is defined as follows. For each node v in the tree, v is the root of the tree returned by a recursive call with $r = v$, the children are the roots of the returned trees for the recursive calls at the next level, arranged in the order of the function calls. For each subtree T_v rooted at v, we call an edge from v to a child w a **branch** of T_v. We also refer to the subtree rooted at the child as a branch of T_v.*

*In Algorithm 3, let us call an execution of the while loop at Lines 3-5 or Lines 7-13 a **w-iteration**. In the following, let $T^{i-1}(k')$ be the tree returned by $A^{i-1}(k', v, X)$, for simplicity, we may also overload the term $A^{i-1}(k', v, X)$ to refer to $T^{i-1}(k')$.*

LEMMA 3 (PROPERTY 1). *Let l be the smaller number of w-iterations in $A^{i-1}(k', v, X)$ and $A^{i-1}(j', v, X)$. $A^{i-1}(j', v, X)$ and $A^{i-1}(k', v, X)$ must share the same outcomes for the first $(l-1)$ w-iterations. That is, $T^{i-1}(k')$ and $T^{i-1}(j')$ have the same first $(l-1)$ branches in the process tree.*

DEFINITION 2 $(\mathbb{L}, p_l^{i-1})$. *Denote by $\mathbb{L}$ the total number of w-iterations in $A^{i-1}(k, v, X)$. Let p_l^{i-1} be the number of covered terminals (terminals in T) after l w-iterations in $A^{i-1}(k, v, X)$, where $0 \leq l \leq \mathbb{L}$. $p_0^{i-1} = 0, p_{\mathbb{L}}^{i-1} = k$.*

LEMMA 4 (PROPERTY 2). *For $v \in \mathbb{V}$, $A^{i-1}(p_l^{i-1}, v, X) \cup (r, v)$ has the smallest density among all $A^{i-1}(k', v, X) \cup (r, v)$, $\forall k' \in (p_{l-1}^{i-1}, p_l^{i-1}]$, where $1 \leq l \leq \mathbb{L}$.*

PROOF. Let $T^{i-1}(k')$ be the returned tree of $A^{i-1}(k', v, X)$ for a given k' and vertex v. Note that when $k' \geq p_{l-1}^{i-1}$, there are at least l subtrees in $T^{i-1}(k')$ and from Lemma 3, the first $l-1$ branches in $T^{i-1}(k')$ are the same for all such k'. For $k' \geq p_{l-1}^{i-1}$, let $T_l^{i-1}(k')$ be the l^{th} branch in $T^{i-1}(k')$. $T_l^{i-1}(k')$ is obtained in the l^{th} w-iteration (in the function call at Line 10) in $A^{i-1}(k', v, X)$.

First we show that $den(T_l^{i-1}(k')) \geq den(T_l^{i-1}(p_l^{i-1}))$ $\forall k' \in (p_{l-1}^{i-1}, p_l^{i-1}]$. If there exists a k'' such that $den(T_l^{i-1}(k'')) < den(T_l^{i-1}(p_l^{i-1}))$, then k'' should be the number of covered terminals after l w-iterations in $A^{i-1}(k, v, X)$, instead of p_l^{i-1}. From Lemma 3, the first $l-1$ branches in $T^{i-1}(k')$ are exactly the same $\forall k' \in (p_{l-1}^{i-1}, p_l^{i-1}]$. Note that $T^{i-1}(k')$, where $p_{l-1}^{i-1} < k' < p_l^{i-1}$, may have more than l branches while $T^{i-1}(p_l^{i-1})$ has exact l branches. According to the greedy selection in Algorithm 3, the density of the ℓ^{th} branch should be greater than that of the l^{th} branch in $T^{i-1}(k')$ when $\ell > l$. Hence $den(T^{i-1}(p_l^{i-1})) \leq den(T^{i-1}(k'))$.

Let α be the cost of (r, v). The density of $A^{i-1}(k', v, X) \cup (r, v)$ is $\alpha/k' + den(T^{i-1}(k'))$, $\forall k' \in (p_{l-1}^{i-1}, p_l^{i-1}]$. Since $\alpha/p_l^{i-1} \leq \alpha/k'$ and $den(T^{i-1}(p_l^{i-1})) \leq den(T^{i-1}(k'))$, we can conclude that the density of $A^{i-1}(p_l^{i-1}, v, X) \cup (r, v)$ is the smallest among all $A^{i-1}(k', v, X) \cup (r, v)$, $\forall k' \in (p_{l-1}^{i-1}, p_l^{i-1}]$ and a specific v. □

Property 1 indicates repeated computations in trying different $k', 1 \leq k' \leq k$, for each $v \in \mathbb{V}$. Property 2 further confirms the possible saving by skipping certain k' values for a specific v. It reveals that one only need to consider $A^{i-1}(p_l^{i-1}, v, X)$, where $1 \leq l \leq \mathbb{L}$. What is more, according to Property 1, we only need to try $A^{i-1}(k, v, X)$ since each of $A^{i-1}(p_l^{i-1}, v, X)$ $(1 \leq l < \mathbb{L})$ is already covered in the process of $A^{i-1}(k, v, X)$. It remains to work out the design of an improved algorithm that can make use of these two properties. For our design, another crucial observation is the following. Let α be the cost of (r, v). As l increases, p_l^{i-1} increases, and the covered terminals increases, α/p_l^{i-1} decreases, while $den(T^{i-1}(p_l^{i-1}))$ increases. Hence the density $\alpha/p_l^{i-1} + den(T^{i-1}(p_l^{i-1}))$ can increase or decrease as l increases. This means that we need to keep track of the smallest density with increasing l.

Algorithm 4: $\widetilde{A}^i(k, r, X)$

```
Input  : Terminal set X, covered number of terminals k, root r, level
         number i, transitive closure G_t;
Output : A tree T with height i rooted at r covering k terminals in X
1  Initialize: T ← ∅;
2  if i = 1 then
3      while k > 0 do
4          (r, v) ← arg_(r,v) min cost(r, v), ∀v ∈ X;
5          T ← T ∪ (r, v); k ← k − 1; X ← X − {v};
6  else
7      while k > 0 do
8          T_BEST ← ∅; den(T_BEST) ← ∞;
9          for each vertex v ∈ V do
10             T' ← B^{i−1}(k, v, X, (r, v)) ∪ (r, v);
11             if den(T_BEST) > den(T') then
12                 T_BEST ← T';
13         T ← T ∪ T_BEST; k ← k − |X| ∩ V(T_BEST);
           X ← X − V(T_BEST);
14 return T;
```

Algorithm 4 is our improved algorithm. Within Algorithm 4, Algorithm 5 is called at Line 10. The main improvement in Algorithm 4 is that for each v, recursive call (Algorithm 5) is triggered only once instead of k times when compared to Line 9 in Algorithm 3. Algorithm 5 replaces the checking of different k' in Line 9 in Algorithm 3 according to the analysis above and we will show that it returns the $T^{i-1}(p_l^{i-1})$ where the density (denoted by $den()$) of $T^{i-1}(p_l^{i-1}) \cup (r, v)$ is the smallest (the best) for $\forall 1 \leq l \leq \mathbb{L}$. Note that the number of covered terminals in the returned tree of $B^i(k, r, X, e)$ is not necessarily k, instead it should be p_l^{i-1} for some l so that the density of $T^{i-1}(p_l^{i-1}) \cup (r, v)$ is the smallest. From Lines 16-17, the best density $T^{i-1}(p_l^{i-1}) \cup (r, v)$ is always tracked in Algorithm 5. Before we prove the correctness, let us first compare our algorithm with Algorithm 3 with an example.

EXAMPLE 7 (ALGORITHM 4). *Compared to Algorithm 3, Algorithm 4 only calls function $B^{i-1}(k, v, X, (r, v))$ once for each v while Algorithm 3 calls function $A^{i-1}(k', v, X)$ k times for each v. Given the graph in Figure 4. Compare $\widetilde{A}^2(5, 0, X)$ with $A^2(5, 0, X)$ where $X = \{1, 2, 3, 4, 5\}$, taking vertex 0 as r. $B^1(5, 0, X, (0, 0))$ is called in $\widetilde{A}^2(5, 0, X)$ and $T = \{(0, 3)\}$ is found, where $T \cup \{(0, 0)\}$ is of the best density. While in $A^2(5, 0, X)$, $A^2(5, k', X)$ is called for different k' values, so that the resulting $T \cup \{(0, 0)\}$ is of the best density. Here $i = 2$, the cases of $i \geq 3$ work in a similar way according to Lines 9-17 instead of Lines 3-7 in Algorithm 5.*

For our proof of correctness, let us also call an execution of the while iteration at Lines 3-5 or Lines 7-13 of Algorithm 4; or at Lines 3-5, or Lines 9-17 of Algorithm 5 a **w-iteration**.

THEOREM 7. *[Correctness] Given k, r and X, Algorithm 4 ($\widetilde{A}^i(k, r, X)$) and Algorithm 3 ($A^i(k, r, X)$) return the same tree.*

Algorithm 5: $B^i(k, r, X, e)$

Input : Terminal set X, maximum number of available terminals k, root r, level number i, the incoming edge e of r, $\mathbb{G}_t$;
Output : A tree T with height i rooted at r covering at most k terminals in X so that the density of $T \cup e$ is the smallest

1 Initialize: $T \leftarrow \emptyset$; $T_c \leftarrow \emptyset$, where $den(T \cup e) = den(T_c \cup e) = \infty$;
2 **if** $i = 1$ **then**
3 **while** $k > 0$ **do**
4 $(r, v) \leftarrow arg_{(r,v)}$ min $cost(r, v)$,$\forall v \in X$;
5 $T_c \leftarrow T_c \cup (r, v)$; $k \leftarrow k - 1$; $X \leftarrow X - \{v\}$;
6 **if** $den(T \cup e) > den(T_c \cup e)$ **then**
7 $T = T_c$;
8 **else**
9 **while** $k > 0$ **do**
10 $T_{BEST} \leftarrow \emptyset$; $den(T_{BEST}) \leftarrow \infty$;
11 **for** *each vertex* $v \in \mathbb{V}$ **do**
12 $T' \leftarrow B^{i-1}(k, v, X, (r, v)) \cup (r, v)$;
13 **if** $den(T_{BEST}) > den(T')$ **then**
14 $T_{BEST} \leftarrow T'$;
15 $T_c \leftarrow T_c \cup T_{BEST}$; $k \leftarrow k - |X| \cap V(T_{BEST})$;
 $X \leftarrow X - V(T_{BEST})$;
16 **if** $den(T \cup e) > den(T_c \cup e)$ **then**
17 $T = T_c$;
18 return T;

PROOF. Algorithm 4 differs from Algorithm 3 only at Lines 9-10. Hence, to prove that $\widetilde{A}^i(k, r, X)$ equals $A^i(k, r, X)$, it suffices to prove that T' returned by $B^{i-1}(k, v, X, (r, v)) \cup (r, v)$ (Line 10) in Algorithm 4 has the smallest density among all T' returned by $A^{i-1}(k', v, X) \cup (r, v)$ for different $1 \leq k' \leq k$ (Lines 9-10) in Algorithm 3. According to Lemma 4, we only need to consider the density of $A^{i-1}(k', v, X) \cup (r, v)$, where $k' = p_l^{i-1}, 1 \leq l \leq \mathbb{L}$. Thus, the problem is reduced to proving that the tree T' returned by $B^{i-1}(k, v, X, (r, v)) \cup (r, v)$ in Algorithm 4 has the best density among all T' returned by $A^{i-1}(k', v, X) \cup (r, v)$ for different $k' = p_l^{i-1}, 1 \leq l \leq \mathbb{L}$ in Algorithm 3. Hence, it remains to prove the following:

For a given k and a vertex $u \in \mathbb{V}$, T_c in the l^{th} w-iteration of $B^{i-1}(k, u, X, (r, u))$ is equal to T returned by $A^{i-1}(p_l^{i-1}, u, X)$, where $1 \leq l \leq \mathbb{L}$. Recall that p_l^{i-1} refers to the number of covered terminals (in T) after l w-iterations in $A^{i-1}(k, v, X)$.

We prove by induction on i. For the base case where $i = 2$, consider Lines 3-5 in Algorithm 3 and Algorithm 5, where one terminal is chosen in each while iteration. Hence $p_l^{i-1} = l$ and $\mathbb{L} = k$. In the l^{th} w-iteration of $B^{i-1}(k, u, X, (r, u))$, T_c consists of the shortest paths to l closest terminals. While in $A^{i-1}(p_l^{i-1}, u, X)$, the resulting T also consists of the shortest paths to l closest terminals.

The induction hypothesis states that given k and $v \in \mathbb{V}$, when $i \leq j$, T_c in the l^{th} w-iteration of $B^{i-1}(k, v, X, (r, v))$ is equivalent to T returned by $A^{i-1}(p_l^{i-1}, v, X)$, where $1 \leq l \leq \mathbb{L}$.

Consider $i = j + 1$. We first note that the tree T' returned by $B^{j-1}(k, v, X, (r, v)) \cup (r, v)$ (Line 12) $\forall v \in \mathbb{V}$ in $B^j(k, r, X, e)$ has the smallest density among all trees T' that are returned by $A^{j-1}(p_l^{j-1}, v, X) \cup (r, v)$ (Line 10), $\forall v \in \mathbb{V}$ and $1 \leq l \leq \mathbb{L}$ in $A^j(k, r, X)$, due to the induction hypothesis and Lines 16-17 in Algorithm 5. According to Lemma 4, for each $v \in \mathbb{V}$, the tree T' returned by $B^{j-1}(k, v, X, (r, v)) \cup (r, v)$ in $B^j(k, r, X, e)$ has the best density among all T' returned by $A^{j-1}(k', v, X) \cup (r, v)$ in $A^j(k, r, X)$, where $1 \leq k' \leq k$. Hence, T_{BEST} in the first w-iteration of $B^j(k, r, X, e)$ is the same as that in the first w-iteration of $A^j(k, r, X)$. That is, T_c in the first w-iteration of $B^j(k, r, X, e)$ is the same as T in the first w-iteration of $A^j(k, r, X)$. In addition, the tree returned by $A^j(p_1^j, r, X)$ is the T in the first w-iteration of $A^j(k, r, X)$. Thus, T_c in the first iteration of $B^j(k, r, X, e)$ is the same as T in $A^j(p_1^j, r, X)$.

In the first w-iteration of $A^j(k, r, X)$ and $B^j(k, r, X, e)$, k and X are updated according to Line 13 in Algorithm 3 and Line 15 Algorithm 5, respectively. The updates are the same since the T_{BEST}'s are the same. Let k_l and X_l be the updated k and X values, respectively, at the end of the l^{th} w-iteration in $A^j(k, r, X)$ and $B^j(k, r, X, e)$. In the second w-iteration of $A^j(k, r, X)$ and $B^j(k, r, X, e)$, the above analysis on $B^{i-1}(k_1, v, X_1, (r, v))$ and $A^{i-1}(k_1, v, X_1)$ also applies. That is, the T_{BEST} trees in the second w-iteration of $B^j(k, r, X, e)$ and that of $A^j(k, r, X)$ are the same. Hence, T_c in this w-iteration of $B^j(k, r, X, e)$ is the same as T returned by $A^j(p_2^j, r, X)$. Similarly, the argument applies to the remaining w-iterations. Thus, the tree T_c in the l^{th} w-iteration of $B^j(k, r, X, e)$ is equivalent to the tree T returned by $A^j(p_l^j, r, X)$. This completes the proof by induction.

With the above correspondence of T_c and T, and according to Lines 16-17, $B^{i-1}(k, v, X, (r, v))$ returns T such that $T \cup (r, v)$ has the best density among all w-iterations, we conclude that the density of $B^{i-1}(k, v, X, (r, v)) \cup (r, v)$ (Line 10) in Algorithm 4 is the best among all $A^{i-1}(k', v, X) \cup (r, v)$, where $k' = p_l^{i-1}, 1 \leq l \leq \mathbb{L}$ (Lines 9-10) in Algorithm 3.

Thus, given k, r and X, Algorithm 4 ($\widetilde{A}^i(k, r, X)$) and Algorithm 3 ($A^i(k, r, X)$) return the same tree. □

THEOREM 8. *Given a static graph $\mathbb{G} = (\mathbb{V}, \mathbb{E})$, Algorithm 4 provides an $i^2(i-1)k^{1/i}$ approximation to the optimal DST in time $O(|\mathbb{V}|^i k^i)$.*

PROOF. For the approximation bound, the analysis is the same as Theorem 3.1 in [8], except that the corrected result in [14] replaces that of Lemma 2.2 in [8]. Next consider the time complexity analysis. First, $\widetilde{A}^i(k, r, X, e)$ invokes at most nk $B^{i-1}(k, r, X, e)$ and each $B^{i-1}(k, r, X, e)$ invokes at most nk $B^{i-2}(k, r, X, e)$. Recursively, we have that the time complexity for $\widetilde{A}^i(k, r, X, e)$ is bounded by $O(|\mathbb{V}|^i k^i)$. □

From Theorem 8, if we choose $i = \log n$, then Algorithm 4 attains an $O(\log^3 n)$ approximation for the optimal directed Steiner Tree problem in $O(n^{\log n} k^{\log n})$ time. Algorithm 3 achieves the same approximation guarantee but requires $O(n^{\log n} k^{2 \log n})$ time. From our experimental study, we show that the improvement in time complexity translates into significantly faster runtime.

4.5 Enhancement by Density Based Ordering

In Section 4.4, we discuss the pruning of different k' ($1 \leq k' < k$), but still we need to process all different $v \in \mathbb{V}$ at Line 9 of Algorithm 3. In this section, we introduce a pruning strategy so that we may explore only a subset of $\mathbb{V}$ instead. This pruning strategy is based on vertex ordering.

Consider the improved $A^i(k, r, X)$ in Algorithm 4, the function $B^{i-1}(k, v, X, e)$ is called for each vertex $v \in \mathbb{V}$ in the first while iteration. Let $\tau(v)$ be the density of $B^{i-1}(k, v, X, (r, v)) \cup (r, v)$, $\forall v \in \mathbb{V}$. That is, $\tau(v)$ records the best density of all possible $T^{i-1}(k')$, $1 \leq k' \leq k$, for $\forall v \in \mathbb{V}$. Let us sort the vertices in non-descending order of $\tau()$. In the next while iteration, we call $B^{i-1}(k_r, v, X_r, e)$ according to this vertex order. We keep updating $\tau(v)$ and record the best τ_{best} so far in this while iteration. We can substantially reduce the computation of $B^{i-1}(k_r, v, X_r, e)$ for some v, whose $\tau(v)$ is bigger than the current best τ_{best} as shown in Line 3* of Algorithm 6. This pruning strategy is also applied to $B^i(k, r, X, e)$.

Algorithm 6: $FinalA^i(k, r, X)$

1* Same as Algorithm 4 except for replacing Lines 9-12 by the following:
2* **for** *each vertex* $v \in \mathbb{V}$ *(in sorted order)* **do**
3* **if** $\tau(v) < \tau_{best}$ **then**
4* $T' \leftarrow FinalB^{i-1}(k, v, X, (r, v)) \cup (r, v)$;
5* **if** $den(T_{BEST}) > den(T')$ **then**
6* $T_{BEST} \leftarrow T'$;
7* $\tau(v) \leftarrow den(T')$;
8* **if** $\tau(v) < \tau_{best}$ **then**
9* $\tau_{best} \leftarrow \tau(v)$;
10* **else**
11* sort $\mathbb{V}$ according to $\tau(v), \forall v \in \mathbb{V}$;
12* break;
13* /* $FinalB^{i-1}(k, v, X, (r, v))$ is the same as Algorithm 5 except for replacing Lines 11-14 by the same for-loop replacement (Lines 2*-12*) as in the above. */

THEOREM 9. *Given k, r and X, Algorithm 6 ($FinalA^i(k, r, X)$) and Algorithm 3 ($A^i(k, r, X)$) return the same tree.*

PROOF. (Sketch) Line 3* in Algorithm 6 is where pruning occurs. Assume we are now starting the for loop execution for a specific vertex v in the l^{th} w-iteration (Line 2* in Algorithm 6), $\tau(v)$ in Line 3* records the best density for v in the $(l-1)^{th}$ w-iteration. The best density for v in the l^{th} iteration cannot exceed that in the $(l-1)^{th}$ w-iteration, due to less available terminals in X. Hence we can use $\tau(v)$ in the previous iteration for possible pruning of vertices in the for loop. □

Though Algorithm 6 has the same time complexity as Algorithm 4, since in the worst case there may be no pruning, in our experiments, pruning by density ordering is shown to be highly effective, and the runtime is improved by more than an order of magnitude in most cases.

5. EXPERIMENTAL RESULTS

In this section, we evaluate the performance of our proposed algorithms for computing $\mathcal{MST}_a$ and $\mathcal{MST}_w$. We also evaluate the quality of the results of our approximation algorithm on DST for datasets with known solutions. We run all experiments on a machine with a 3.4Ghz Intel Core i7-4770 CPU and 16 GB RAM, running Ubuntu 12.04 LTS Linux OS. All algorithms were implemented in C++ and compiled with GNU c++ compiler.

5.1 Datasets

We tested on datasets downloaded from the Koblenz Large Network Collection (http://konect.uni-koblenz.de/). *Slashdot* is a network of user replies collected from technology website Slashdot. *Epinions* records online product ratings, where edges are trust or distrust links between users. *Facebook* is a graph representing posts to other users' walls on Facebook. *Enron* records emails sent among employees of Enron between 1999 and 2003. arXiv *HepPh* is a collaboration network of authors of scientific papers from High Energy Physics, in which vertices are authors and edges are common publications between two authors. *DBLP* is a graph of coauthors from DBLP computer science bibliography. *Phone* is based on anonymized records of Orange phone calls and SMS exchanges in Ivory Coast for five months from Dec, 2011 [5].

All datasets are temporal graphs, each time stamp is either a physical time (for Phone) or a Unix time. For the Phone dataset, each edge has a weight given by the duration_voice_calls attribute.

	$\|V\|$	$\|E\|$	$\|E_s\|$	deg	deg_s	π	$\|\Gamma_G\|$
Slashdot	51K	140K	130K	313	249	17	89862
Epinions	114K	717K	717K	2070	2070	1	939
Facebook	46K	855K	264K	1430	157	742	846858
Enron	87K	1135K	320K	32552	1566	1074	213218
HepPh	28K	9193K	6291K	11134	4909	262	2337
DBLP	1101K	11957K	8451K	1433	1189	38	70
Phone	1192	10766K	512K	9031	429	317	2991

Table 1: Datasets: sizes of temporal graph $G = (V, E)$ and static graph $G_s = (V_s, E_s)$, distinct time instances (Γ_G), and π

The other networks are unweighted, we assign weights to the edges based on the weight cascade model in [19], in which the propagation probability for an edge (u, v) is given by $pp(u, v) = 1/d(v)$, where $d(v)$ is the in-degree of v. We use the out-degree instead since it represents better the graph properties. As noted in [9], if we convert this to a weight of $-\log pp(u, v)$ on (u, v), then we have a maximum influence path when the total edge weight is minimum. Hence we adopt $-\log pp(u, v)$ as the weight (or cost) for each edge (u, v).

Next, we describe how to set the window of $[t_\alpha, t_\omega]$, the time span of interest. For each of the six datasets, we compute the total time range $[t_A, t_\Omega]$ from the time units attached to all edges. Then we set the range $[t_\alpha, t_\omega]$ so that it covers the middle one tenth of the total range, i.e. $(t_\omega - t_\alpha) \approx 0.1(t_\Omega - t_A)$. We extract a subgraph G' of the given graph G by limiting edges to within $[t_\alpha, t_\omega]$. For the root vertex for the MST, we require that it can reach at least one tenth of the set of vertices in G'. We simply scan the vertices until one such vertex is found.

We show some statistics of the datasets in Table 1. In Table 1, π is the maximum number of temporal edges from u to v, for any u and v in $V(G)$. Note that for $\pi = 1$, G can still be very different from G_s since paths must be time respecting.

5.2 Efficiency of the Algorithms for $\mathcal{MST}_a$

Here we evaluate the performance of our two MST_a algorithms, Algorithm 1, denoted by Alg1, and Algorithm 2, denoted by Alg2. We compare with the existing algorithm in [4], denoted by Bhadra, which utilizes a modified Prim-Dijkstra algorithm. The time complexity of Bhadra given in [4] is $O(n^2 D \log \mathcal{T})$, where n is the number of vertices, D is the maximum out degree of a vertex and $\mathcal{T}$ is the number of time points in G. With more careful analysis, the time can be bounded by $O(m \log n + m \log \pi)$, where m is the number of edges in the corresponding static graph $G_S = (V_S, E_S)$, and π is the maximum number of time instances incident to each static edge $e_s \in E_S$, which is bounded by $\mathcal{T}$.

For the window of $[t_\alpha, t_\omega]$, since the computation time is very short for $\mathcal{MST}_a$, other than the default range giving rise to G' as described above, we also consider the range of $[0, \infty]$, where $\mathcal{MST}_a$ spans $V_r(G)$. Our first set of experiment ensures that the durations in the edges are non-zero in the given temporal graphs. As in [27], the durations are set to 1. The results are shown in Table 2. Alg1 outperforms Bhadra by a large margin in all cases.

Our next set of experiment is to compare Alg2 with Bhadra, for networks that may contain zero edge durations. Since Alg1 cannot handle zero duration, it is not included. Following the given datasets, the edge durations are set as zero. The results are shown in Table 3. Alg2 outperforms Bhadra in almost all cases. The root in Table 2 and 3 is set as the first vertex in the input order, which can reach more than one tenth of the total vertex size. Compared with Table 2, the numbers of reachable vertices are similar, except for DBLP. DBLP is about collaboration of coauthors, there exist paths from an author to other authors in the same year, and not across dif-

	Slashdot	Epinions	Facebook	Enron	HepPh	DBLP
$\|V_r(G)\|$	8601	13563	32593	43642	27756	791009
Bhadra	6.97	24.86	16.75	20.57	176.60	666.61
Alg2	0.95	4.00	18.82	15.84	94.45	336.25
Alg1	0.48	1.32	2.74	2.95	13.68	64.67
$\|V_r(G')\|$	898	795	1430	722	283	–
Bhadra	0.467	0.468	0.682	0.264	0.272	–
Alg2	0.036	0.088	0.110	0.059	0.036	–
Alg1	0.032	0.042	0.125	0.024	0.022	–

Table 2: Number of reachable vertices, $|V_r(G)|$, and runtime (in ms) for $\mathcal{MST}_a$ with non-zero edge durations

	Slashdot	Epinions	Facebook	Enron	HepPh	DBLP
$\|V_r(G)\|$	8607	55117	32593	43642	27914	849944
Bhadra	5.95	32.61	16.97	20.82	176.95	707.76
Alg2	1.08	17.29	19.08	15.69	123.50	461.72
$\|V_r(G')\|$	898	863	1430	722	295	4412
Bhadra	0.441	0.473	0.665	0.215	0.360	4.897
Alg2	0.036	0.169	0.103	0.056	0.066	0.603

Table 3: $|V_r(G)|$ and runtime (in ms) for $\mathcal{MST}_a$ with zero edge duration

ferent years. If duration is set to non-zero, co-authors in the same paper cannot reach each other. Hence, the number of reachable vertices is much smaller than that when duration is zero. " $-$ " in Table 2 indicates that no vertex in DBLP can reach more than one tenth of the total vertex set.

5.3 Efficiency of the Algorithms for $\mathcal{MST}_w$

For $\mathcal{MST}_w$, we evaluate our DST-based mechanisms. The preprocessing time includes the time for extracting the graph G' within $[t_\alpha, t_\omega]$ from the given temporal graph G, transforming the temporal graph G' into a static graph $\mathbb{G}$ according to Section 4.2, and computing the transitive closure $\mathbb{G}_T$. Table 4 shows the sizes of G' and $\mathbb{G}$, and the preprocessing time. Note that $|V_r(G')|$ is the number of terminals for the DST problem on $\mathbb{G}$. Since the size of $\mathbb{G}_T$ is $|V(\mathbb{G})|^2$, the preprocessing time is mostly dominated by its computation and grows quickly with $|V(\mathbb{G})|$.

In Table 5, we compare the runtime for the 3 algorithms: the algorithm proposed by Charikar et al. in [8] (Charik), our proposed Algorithm 4 (Alg4), and Algorithm 6 (Alg6). Here, '–' indicates a runtime of over 3 days. The results confirm that Charik is not scalable when the number of iterations is two or more. Our first proposed algorithm, Alg4, improves the runtime by up to 4 orders of magnitude. The second proposed algorithm (Alg6) further improves the runtime by up to 5 orders of magnitude speedup. Note that the time taken for one iteration ($i = 1$) for all algorithms is the same since all algorithms involve finding the edges from the root to all vertices in the transitive closure.

5.4 Quality of the DST Results

The algorithms of Charik, Alg4, and Alg6 for the DST problem are significant because they have the best known approximation guarantee, and they typically exhibit much better quality than the guarantee in actual results. We illustrate the latter point here with some datasets available at http://steinlib.zib/de/showset.php, which are prepared by Zentrum fur Informationstechnik Berlin (ZIB). The datasets consist of randomly generated sparse graphs with edge weights between 1 and 10. The DST problems in these datasets are solved and we can determine the accuracy of our algorithms by comparing with the optimum solution.

	$\|V(G')\|$	$\|E(G')\|$	$\|V_r\|$	$\|V(\mathbb{G})\|$	$\|E(\mathbb{G})\|$	$Tprep$
Slashdot	8,336	15,374	898	3815	338K	29.0s
Epinions	6,127	15,859	1,901	11K	22,721K	684.5s
Facebook	7,294	42,675	1,430	15,949	10,508K	2,095.1s
Enron	2,053	10,000	722	5,642	4,182K	96.4s
HepPh	873	19,532	295	1,848	986K	4.2s
DBLP	43,176	152,736	4,412	13,030	51,097K	1,184.0s
Phone	916	284,596	903	1,829	253K	31s

Table 4: Datasets: sizes of extracted graph G', transformed graph $\mathbb{G}$, and V_r, and preprocessing time ($Tprep$)

Alg-i	Slashdot	Epinions	Facebook	Enron	HepPh	DBLP	Phone
Charik-1	0.3	2.2	5.4	0.7	0.2	9.4	0.3
Charik-2	4468.6	238401	38396.4	4516.7	64.2	–	10469
Charik-3	–	–	–	–	–	–	–
Alg4-1	0.3	2.3	5.4	0.7	0.2	9.4	0.3
Alg4-2	7.7	227.2	47.1	11.6	0.6	966.5	40.4
Alg4-3	6585.1	–	116780	64720	404.5	–	–
Alg6-1	0.3	2.2	5.4	0.7	0.2	9.4	0.3
Alg6-2	0.3	5.0	3.3	1.5	0.1	11.0	0.5
Alg6-3	40.1	21237.2	436.2	1028.1	7.8	85720	88.8

Table 5: Runtime (in sec) for Algorithm 3, Algorithm 4, and Algorithm 6, i is the number of iterations.

i	Slashdot	Epinions	Facebook	Enron	HepPh	DBLP	Phone
1	2002.05	6148.46	3948.07	3644.21	972.22	7584.11	1397.00
2	1999.78	6079.16	3916.39	3453.88	864.13	7384.38	1202.00
3	1974.14	5926.04	3908.11	3433.87	828.50	7206.16	1193.00

Table 6: Weights of $\mathcal{MST}_w$ solutions for i iterations

G	$\|V\|$	$\|E\|$	$\|X\|$	opt	Charik-3	Alg6-3	Alg6-4	Al6-5
b01	50	63	9	82	2.2	0.018	1.40	102.0
b03	50	63	25	138	54.8	0.059	5.92	545.8
b05	50	100	13	61	6.5	0.028	2.34	186.1
b07	75	94	13	111	15.6	0.049	5.84	668.8
b09	75	94	38	220	1061.0	0.217	31.2	4626.8
b11	75	150	19	88	41.3	0.064	9.32	1110.2
b13	100	125	17	165	80.6	0.140	23.45	3620.5
b15	100	125	50	318	4567.5	0.408	96.21	18846.6
b17	100	200	25	131	283.1	0.228	43.73	9227.5

Table 7: Runtime (in sec) for small datasets with known results.

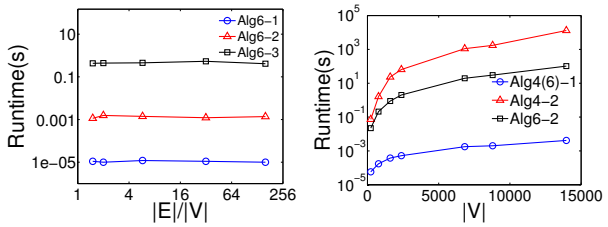
	b01	b03	b05	b07	b09	b11	b13	b15	b17
i=1	0.02	0.07	0.05	0.17	0.13	0.44	0.24	0.08	0.31
i=2	0.00	0.00	0.05	0.16	0.01	0.21	0.09	0.02	0.09
i=3	0.00	0.00	0.05	0.02	0.00	0.14	0.09	0.02	0.03
i=4	0.00	0.00	0.05	0.01	0.00	0.11	0.05	0.02	0.03
i=5	0.00	0.00	0.05	0.01	0.00	0.11	0.05	0.02	0.02

Table 8: Result Quality: relative error in total weight

Table 7 shows the graph sizes, the number of terminals ($|X|$), the cost (total weight) of the optimum solution (Opt), and the runtime (in sec) for the algorithm Charik for $i = 3$ and Alg6 for $i = 3, 4, 5$. As in our previous results, the difference in runtime shows up to 4 orders of magnitude improvement of Alg6 over Charik. For the approximation accuracy, we show the relative error in Table 8. That is, we show the values of $(Approx - Opt)/Opt$, where $Approx$ is the total weight of the DST tree returned by our approximation algorithm, and Opt is the weight of an optimum DST. The results show that the accuracy of our algorithms is much better than the theoretical approximation bound in all the test cases, which indicates that the bound may not be tight. The approximation results are very close to the optimum results when $i = 3$.

5.5 Impact of Graph Sizes and Other Studies

Figure 8 shows the runtime of our algorithms for datasets with different $|V|$ and $|E|/|V|$ values. Datasets from the Steinlib collection at http://steinlib.zib.de/showset.php?I320 and http://steinlib.zib



(a) $|V| = 320, k = 256$ (b) $|E|/|V| = 1.1, k/|V| = 0.8$

Figure 8: Runtime with varying $|E|/|V|$ and $|V|$

.de/showset.php?WRP4 are used in this experiment. Figure 8(a) illustrates the runtime for Alg-6 when $|V|$ is fixed and $|E|/|V|$ is varied. As $|E|/|V|$ increases, the execution time remains stable. The reason is that the input to Alg-6 is a transitive closure we get after preprocessing, thus, the average degree of the given graph has no effect on the runtime. Figure 8(b) shows the runtime of Alg-4 and Alg-6 for different $|V|$ values when $|E|/|V|$ and $k/|V|$ are fixed. The experiment results reflect the complexity bound of $O(|V|^i k^i)$ for Alg-4 and Alg-6.

We have conducted some further empirical studies which show that graph sizes do not affect the quality of results, and varying the edge weights also has no impact on both the runtime and quality. From Tables 6 and 8, the quality of the results is mainly affected by the number of iterations of the algorithms.

6. RELATED WORK

Given a non-temporal undirected weighted graph $G = (V, E)$, the MST problem can be solved in $O(|E| \log |V|)$ time by a greedy algorithm [24, 22]. Given a non-temporal (static) directed weighted graph $G = (V, E)$, a MST can be computed in $O(|E||V|)$ time using the Chu -Liu / Edmonds algorithm [10, 11], a faster implementation of which takes $O(|E| \log |V|)$ time for sparse graphs and $O(|V|^2)$ for dense graphs [25, 6]. The implementation in [12] further improves the runtime to $O(|E| + |V| \log |V|)$.

Bhadra et al. in [4] introduces a similar concept as $\mathcal{MST}_a$, computing directed MSTs in a strongly connected evolving digraph and utilizing a modification of the Prim-Dijkstra algorithm. $\mathcal{MST}_a$ and $\mathcal{MST}_w$ are related to paths from the root to other vertices. An early work on temporal graphs computes disjoint paths between any two vertices [20]. Different kinds of paths in a temporal graph are defined in [29], among them paths with earliest arrival times are called *foremost* paths. The first linear time algorithms for computing different kinds of paths in temporal graphs are proposed in [16]. The idea of time ordered edges in [16] is adopted in [27] for a different input format. Note that the algorithm for earliest-arrival paths in [27] has a similar problem as Algorithm 1 with zero durations, and our technique in Algorithm 2 can provide a solution. Some recent studies have explored the use of temporal information in different applications. In [21], temporal paths are studied for analyzing the information latency among vertices in a social network. In [28], long term and short term preferences of users over time are modeled based on the temporal graph and temporal information for recommendation purpose. Surveys of research on temporal graphs are found in [7] and [15].

Other than information dissemination, MSTs are also useful in the problem of clustering [31, 17, 33, 30, 2]. MSTs play a significant role in other graph problems. They are used in solving the traveling salesman problem, the multi-terminal minimum cut problem, single terminal maximum flow problem, and the minimum cost weighted perfect matching problem.

Steiner tree problems have been found useful in different applications, see [26] for a general survey. The DST problem is also interesting from a theoretical perspective since a variety of problems can be reduced to it while preserving the approximation [8]. The DST algorithm in [8] is based on a result in [32].

7. CONCLUSION

We study the problem of minimum spanning trees in temporal networks, for which we have proposed two meaningful definitions and derived efficient exact and approximation algorithms, respectively. For future work we plan to extend our results to the problem of minimum directed Steiner tree in a temporal graph. This will be useful for targeted information dissemination such as content delivery networks for delivering web-based contents to target sites.

ACKNOWLEDGEMENTS: We thank the anonymous reviewers for their very helpful comments and suggestions. This research was supported by the RGC GRF research grant 412313 Proj_id 2150758 of Hong Kong.

8. REFERENCES

[1] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and hardness of approximation problems. In *FOCS*, pages 14–23, 1992.

[2] T. Asano, B. Bhattacharya, M. Keil, and F. Yao. Clustering algorithms based on minimum and maximum spanning trees. In *Symposium on Computational Geometry*, pages 252–257, 1988.

[3] N. Bailey. *The Mathematical Theory of Infectious Diseases and its Applications*. Hafner Press, 1975.

[4] S. Bhadra and A. Ferreira. Complexity of connected components in evolving graphs and the computation of multicast trees in dynamic networks. In *Ad-Hoc, Mobile, and Wireless Networks*, pages 259–270. Springer, 2003.

[5] V. D. Blondel, M. Esch, C. Chan, F. Clerot, P. Deville, E. Huens, F. Morlot, Z. Smoreda, and C. Ziemlicki. Data for development: the d4d challenge on mobile phone data. In *arXiv:1210.0137*, Jan 2013.

[6] P. Camerini, L. Fratta, and F. Maffioli. A note on finding optimum branchings. *Networks*, 9:309–312, 1979.

[7] A. Casteigts, P. Flocchini, W. Quattrociocchi, and N. Santoro. Time-varying graphs and dynamic networks. *International Journal of Parallel, Emergent and Distributed Systems*, 27(5):387–408, 2012.

[8] M. Charikar, C. Chekuri, T.-y. Cheung, Z. Dai, A. Goel, S. Guha, and M. Li. Approximation algorithms for directed steiner problems. In *SODA*, pages 192–200, 1998.

[9] W. Chen, C. Wang, and Y. Wang. Scalable influence maximization for prevalent viral marketing in large-scale social networks. In *16th ACM SIGKDD*, pages 1029–1038, 2010.

[10] Y. J. Chu and T. H. Liu. On the shortest arborescence of a directed graph. *Science Sinica*, 14:1396–1400, 1965.

[11] J. Edmonds. Optimum branchings. *J. Res. Nat. Bur. Standards*, 71B:233–240, 1967.

[12] H. N. Gabow, Z. Galil, T. Spencer, and R. E. Tarjan. Efficient algorithms for finding minimum spanning trees in undirected and directed graphs. *Combinatorica*, 6:109–122, 1986.

[13] G. Galbiati, F. Maffioli, and A. Morzenti. A short note on the approximability of the maximum leaves spanning tree problem. *Information Processing Letters*, 52:45–49, 1994.

[14] C. S. Helvig, G. Robins, and A. Zelikovsky. An improved approximation scheme for the group steiner problem. *Networks*, 37(1):8–20, 2001.

[15] P. Holme and J. Saramäki. Temporal networks. *CoRR*, abs/1108.1780, 2011.

[16] S. Huang, J. Cheng, and H. Wu. Temporal graph traversals: Definitions, algorithms, and applications. In *arXiv:1401.1919*, Jan 2014.

[17] J.Shi and J.Malik. Normalized cuts and images egmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22:888–905, 2000.

[18] V. Kann. *On the approximability of NP-complete optimization problems*. PhD thesis, Royal Institute of Technology Stockholm, 1992.
[19] D. Kempe, J. Kleinberg, and É. Tardos. Maximizing the spread of influence through a social network. In *ACM SIGKDD*, pages 137–146, 2003.
[20] D. Kempe, J. M. Kleinberg, and A. Kumar. Connectivity and inference problems for temporal networks. In *STOC*, pages 504–513, 2000.
[21] G. Kossinets, J. M. Kleinberg, and D. J. Watts. The structure of information pathways in a social communication network. In *KDD*, pages 435–443, 2008.
[22] J. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7:48–50, 1956.
[23] C. H. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. In *20th ACM STOC*, 1988.
[24] R. Prim. Shortest connection networks and some generalizations. *Bell System Technical Journal*, 36:1389–1401, 1957.
[25] R. E. Tarjan. Finding optimum branchings. *Networks*, 7:25–35, 1977.
[26] P. Winter. Steiner problem in networks: a survey. *Networks*, 17:129–167, 1987.
[27] H. Wu, J. Cheng, S. Huang, Y. Ke, Y. Lu, and Y. Xu. Path problems in temporal graphs. In *VLDB*, 2014.
[28] L. Xiang, Q. Yuan, S. Zhao, L. Chen, X. Zhang, Q. Yang, and J. Sun. Temporal recommendation on graphs via long and short term preference fusion. In *ACM SIGKDD*, pages 723–732, 2010.
[29] B.-M. B. Xuan, A. Ferreira, and A. Jarry. Computing shortest, fastest, and foremost journeys in dynamic networks. *Int. J. Found. Comput. Sci.*, 14(2):267–285, 2003.
[30] Y.Xu, V.Olman, and D.Xu. Clustering gene expression data using a graph-theoretic approach: an application of minimum spanning tree. *Bioinformatics*, 18:536–545, 2002.
[31] C. Zahn. Graph-theoretical methods for detecting and describing gestalt clusters. *IEEE Transactions on Computers*, C-20:68–86, 1971.
[32] A. Zelikovsky. A series of approximation algorithms for the acyclic directed steiner tree problem. *Algorithmica*, 18:99–110, 1997.
[33] C. Zhong, D. Miao, and R. Wang. A graph-theoretical clustering method based on two rounds of minimum spanning trees. *Pattern Recognition*, 43:752–766, 2010.

9. APPENDIX

This appendix contains our proofs for some theorems.

9.1 Proof of Theorem 3 for hardness of $\mathcal{MST}_w$

PROOF. We prove that computing $\mathcal{MST}_w$ in a temporal graph is NP-hard by a reduction from the maximum leaf spanning tree problem in static graphs, which is known to be NP-complete. The decision problem for *maximum leaf spanning tree* is defined as follows: Given an undirected static graph $G_s = (V_s, E_s)$; $|V_s| = n, |E_s| = m$; is there a spanning tree T_s of G_s with k or more leaves (vertices with degree 1)? The corresponding decision problem for $\mathcal{MST}_w$ is: given a temporal graph G, is there a spanning tree with a total weight of no more than $2(n+1)-k$?

We transform the given undirected static graph G_s into a temporal directed graph G in the following way:

For each edge (u, v) in the given static graph G_s, (1) create $2n$ temporal edges of the form $(u, v, 2i, 2i+2, 2)$ and $(v, u, 2i, 2i+2, 2)$, for $0 \leq i < n$; (2) create 2 temporal edges of the form $(u, v, 2n+1, 2n+2, 1)$ and $(v, u, 2n+1, 2n+2, 1)$.

That is, for each edge (u, v) in the static graph, we have $n+1$ edges from u to v in the temporal graph, with starting times of $0, 2, 4, ..., 2n-2, 2n+1$. We prove that finding a spanning tree T_s in G_s with k or more leaves is equivalent to the corresponding decision problem for $\mathcal{MST}_w$ in the constructed temporal graph G.

Firstly, given spanning tree T_s in G_s, we can always construct a corresponding temporal spanning tree T in G as follows: Let $l(u)$ be the distance (level number) of vertex u from root r in T_s, where $l(r) = 0$. Create an empty tree T. For each edge (u, v) in T_s, if v is an leaf node, then add temporal edge $(u, v, 2n+1, 2n+2, 1)$ to T; otherwise, add edge $(u, v, 2 * l(u), 2 * l(u) + 2, 2)$ to T. We observe that if there are k leaves in T_s, the total weight in T is $2(n-1)-k$.

Conversely, given a MST_w, T for G, if there are k leaves in T, then the weight of T is $2(n-1)-k$, since the smallest weight of an edge from a vertex u to a leaf v in T is 1 (by picking the edge from u to v with start time $2n+1$), while the remaining edges have weights of 2. Obviously, if we ignore the time information, T is also a spanning tree for G_s with k leaves. Hence, if there exists a spanning tree for G with weight $2(n-1)-k$, there exists a spanning tree for G_s with k leaves. □

9.2 Proof of Theorem 5 for problem transformation from $\mathcal{MST}_w$ to DST

PROOF. Given a temporal graph $G = (V, E)$ and root r, let $\mathbb{G}$ be the corresponding transformed graph and X be the terminal set.

Firstly, we show how a minimum directed Steiner tree $T_{\mathbb{G}}$ rooted at r with terminal set X in $\mathbb{G}$ corresponds to a spanning tree T with root r in G. *For each dummy vertex $v \in X$, there is only one in-neighbor, $v_{|\mathcal{T}(v)|}$, in $\mathbb{G}$. Hence, for each $v \in X$, $v_{|\mathcal{T}(v)|} \in V_{\mathbb{G}}$ must be in the Steiner tree $T_{\mathbb{G}}$. Suppose multiple copies of v, namely, $\{v_i, .., v_j\}$, are covered in $T_{\mathbb{G}}$, where $i < j$. Then, there exists a minimum DST where the incoming edges for $\{v_i, .., v_j\}$ must be (u_k, v_i, w), $(v_i, v_{i+1}, 0)$, ..., $(v_{j-1}, v_j, 0)$, respectively, where u_k is a copy of $u \in V$. This can be ensured as follows. Without loss of generality, we consider the case where the in-neighbors of v_i and v_j's are u_k and x_l, respectively, where $i < j$. Then w in (x_l, v_j, w) must be 0, for otherwise, we can replace (x_l, v_j, w) by $(v_{j-1}, v_j, 0)$ and get a smaller cost. Since $w = 0$, we can preserve the cost of $\zeta(T_{\mathbb{G}})$ by replacing edge $(x_l, v_j, 0)$ by $(v_{j-1}, v_j, 0)$. In this way we can form a DST where for each $v \in V$ in G, there is a single incoming solid edge in $\mathbb{G}$ which corresponds to an edge in G. Thus, a minimum DST $T_{\mathbb{G}}$ in $\mathbb{G}$ corresponds to a spanning tree T rooted at r in G, and $\zeta(T) = \zeta(T_{\mathbb{G}})$.*

Secondly, we show that T in the above is a $\mathcal{MST}_w$ with root r in G. *We prove by contradiction. Suppose there exists a spanning tree T' in G with $\zeta(T') < \zeta(T)$. Then we can construct a DST $T'_{\mathbb{G}}$ with $\zeta(T'_{\mathbb{G}}) < \zeta(T_{\mathbb{G}})$ as follows. For each vertex $v \in V$ and $v \neq r$, there is one incoming edge e in T'. Include in $T'_{\mathbb{G}}$ the solid edge created for e. Let v_i be the corresponding vertex for end vertex v in $\mathbb{G}$ in step 2(b). Include the edges $(v_i, v_{i+1}, 0), .., (v_{|\mathcal{T}(v)|}, v, 0)$ in $T'_{\mathbb{G}}$. Obviously, $\zeta(T'_{\mathbb{G}}) < \zeta(T_{\mathbb{G}})$, and we arrive at a contradiction.*

From the above, a $\mathcal{MST}_w$ in G can be derived in linear time from a minimum DST in the transformed graph $\mathbb{G}$. □

9.3 Proof of Theorem 6 for the guarantees of Algorithm 4 for $\mathcal{MST}_w$

PROOF. From the proof of Theorem 5, the minimum directed Steiner tree $T_{\mathbb{G}}$ in $\mathbb{G}$ corresponds to a spanning tree T rooted at r in G, and $\zeta(T) = \zeta(T_{\mathbb{G}})$. For the DST problem, the Postprocessing Step 1 in Section 4.3 gives the approximation solution. Thus, if we show that Postprocessing Step 2 in Section 4.3 does not increase the cost or weight, then the theorem is proved. This is trivial since in Step 2, we only merge multiple solid edges from u to v, if any. The merging can only reduce the total weight so that the final spanning tree has no more weight than that of the tree from Step 1. The time complexity follows from Lemma 2 since $|\mathbb{V}| = O(|E|)$. □