Integer Programming Approach for Directed Minimum Spanning Tree Problem on Temporal Graphs

Takuto Ikuta
The University of Tokyo
Tokyo, Japan
tikuta@is.s.u-tokyo.ac.jp

Takuya Akiba
National Institute of Informatics
Tokyo, Japan
takiba@nii.ac.jp

ABSTRACT

Considerable effort has been devoted to establishing concepts and designing algorithms that are useful for graph data management. While most work so far has focused on static graphs, there are many networks with time information, i.e., *temporal graphs*, such as social network messages, phone calls, public transportation, and neural networks. Even the most fundamental problems for static graphs become nontrivial for temporal graphs. In this paper, we explore the *minimum-weight spanning tree problem* on temporal graphs, which was recently proposed by Huang *et al.* [SIGMOD 2015]. Even though this problem is proven to be NP-hard, we design practically efficient *exact* algorithms using *integer programming*. Experimental results confirm that the proposed algorithms can produce better solutions than a previously proposed approximation algorithm.

Categories and Subject Descriptors

G.2.2 [**Discrete Mathematics**]: Graph Theory—*Graph algorithms*

Keywords

Integer programming, Exact algorithm, Minimum spanning tree, Temporal graph

1. INTRODUCTION

Although there has been an extensive amount of research focusing on static networks, many real world networks, including social networks, co-authorship networks and transportation networks, hold temporal information. In the real world, this data can be represented using temporal graphs. While analyzing temporal graphs is, in general, more difficult than analyzing static graphs, many researchers have been paying attention to the topic due to its practical importance. For example, both the shortest path problem [24, 26] and core-decomposition problem [25] on the temporal graph have been studied.

NDA'16 , June 26-July 01 2016, San Francisco, CA, USA

DOI: http://dx.doi.org/10.1145/2980523.2980528

Huang, Fu, and Liu [13] proposed a way to apply the minimum directed spanning tree problem on temporal graphs. A spanning tree on a temporal graph is used to analyze how information spreads over a whole graph from a certain vertex. For static graphs, it is well known that the usual minimum spanning tree problem can be solved in polynomial time [9, 5, 10, 17, 22]. However, Huang *et al.* proved that the minimum spanning tree problem for temporal graphs is NP-hard. Thus, they proposed an approximation algorithm based on the known approximation algorithm for the Steiner tree problem [4]. The theoretical approximation ratio of their algorithm can be huge (more than 10000%) for large networks. However, they experimentally showed that, for small graphs (of roughly 100 vertices), whose optimal solutions are known, the approximation ratio of their algorithm is somewhat small (roughly 30%).

In this paper, we propose a method that computes the optimal solutions for many datasets of the minimum spanning tree problem on temporal graphs generated from real-world networks. Our method formulates the problem as an integer linear programming problem. We experimentally show that our method runs quickly for networks generated from real world data.

Based on experimental results, we believe that real-world networks may have certain properties (implicitly used by the integer linear programming solver) that make the problem easy. Some instances are solved quickly even if there are no known polynomial-time algorithms for the problem. In addition, we show that Huang *et al.*'s method has a good approximation ratio of roughly 30% to 40% even for *large* real-world networks.

1.1 Related Work

The minimum spanning tree problem for a static graph has been studied substantially. For undirected graphs, Prim's algorithm and Kruskal's algorithm [22, 17] solve the minimum spanning tree problem. These algorithms run in time $O(|E| \log |V|)$ for a graph $G = (V, E)$. For a directed graph, Chu-Liu/Edmonds' algorithm runs in time $O(|E||V|)$ [5, 9]. Gabow, Galil, Spencer and Tarjan improve their algorithm to $O(|V| \log |V| + |E|)$ [10].

Huang, Fu and Liu [13] were the first to study the minimum spanning tree problem for a temporal graph. In the paper by Huang *et al.*, they proposed two definitions for the minimum directed spanning tree problem on temporal graphs. One problem is minimizing the farthest distance between vertices and the root vertex. They proposed an algorithm that runs in time $O(|E| \log |V|)$ for this problem. Another problem is minimizing the total edge weight of a

spanning tree in a temporal graph. They prove that the second problem is NP-hard and proposed approximation algorithms for the problem.

For NP-hard problems, the integer linear programming method is often used as a technique for finding the optimal solution of the problem in a reasonable time. The traveling salesman problem is one of the most famous problems that is studied when developing exact, approximation, and heuristic algorithms [21, 2, 6, 18]. There are some variants of the minimum spanning tree problem. Behle, Jünger, and Liers studied the degree constrained minimum spanning tree problem in 2007 [3]. The Steiner tree problem that is a generalized version of the minimum spanning tree problem has also been studied extensively. Koch and Martin solved some instances of the Steiner tree problem optimally in 1998 using integer linear programming and the cutting plane method that maintains the connectivity of a solution [16]. They also proposed preprocessing steps for the problem to reduce problem size before formulating the integer linear program. Ljubic, Weiskircher, Pferschy, Klau, Mutzel and Fischetti studied the prize-collecting Steiner tree problem [19]. The prize-collecting Steiner tree problem is a generalized version of the Steiner tree problem such that each vertex has a profit. Ljubic *et al.* solved the problem using a similar approach to the algorithm Koch *et al.* used to solve the Steiner tree problem. In this paper, we use the same cutting plane method that generates constraints to maintain the connectivity of the solution used in the above research [3, 16, 19].

Another topic related to temporal graphs is the shortest path. Wu, Cheng, Huang, Ke, Lu and Xu defined four types of shortest paths on temporal graphs [24]. Wang, Lin, Yang, Xiao and Zhou studied shortest path queries on temporal graphs using an indexing approach [23]. Wu, Cheng, Lu, Ke, Huang, Yan, and Wu studied core-decomposition for temporal graphs that had applications in visualization, community analysis, and so on [25].

1.2 Our Contribution

We propose an exact algorithm for the minimum directed spanning tree problem, while existing methods focus on obtaining an approximate solution. We implement an integer linear programming formulation for the minimum directed spanning tree on temporal graphs based on extensive research[3, 20, 16, 19, 15]. It is possible that the size of naïve constraints about the temporal property becomes quadratic with the size of the input. In order to obtain linear size constraints about the temporal property, we introduce auxiliary variables into the formulation. Even if we cannot obtain an exact solution, we can establish a lower and upper bound using our method. Thus, even if it is not necessary to obtain an exact solution, we can know if the solution is good.

We empirically evaluated two exact methods and Huang *et al.*'s method in our experiments using real-world networks. We compared the three methods in terms of the cost of the solutions and running times. Our experimental results show that the exact method gives an optimal solution for many datasets from real-world networks. In particular, our method solves large instances in less time than Huang *et al.*'s existing approximation algorithm.

2. PRELIMINARIES

2.1 Temporal Graph

The definition of a temporal graph that we use in this paper was proposed by Huang *et al.* [13]. A *temporal graph* $G = (V, E)$ consists of a vertex set V and a set E of edges with some additional information: for each edge $e \in E$, we use $h(e)$ as the head of edge e, $t(e)$ as the tail of edge e. An edge e connects vertices from $t(e)$ to $h(e)$. An edge e also has starting time $s(e)$ from $t(e)$, arrival time $a(e)$ at $h(e)$ and an edge weight $w(e)$. For example, this framework could be used for a situation in which a message is sent from vertex $t(e)$ in time $s(e)$ and arrives at $h(e)$ in time $a(e)$.

An important difference between a temporal graph and a static graph is the definition of a path. In a temporal graph, a sequence $P = (e_1, e_2, \cdots, e_k)$ of edges is said to be a *temporal path* if and only if $a(e_i) \leq s(e_{i+1})$ and $h(e_i) = t(e_{i+1})$ for any $i \in \{1, \ldots, k-1\}$. This means that an edge e_i cannot have an earlier start time than the arrival time of the previous edge on the temporal path and the head of e_i should be equal to the tail of e_{i+1}.

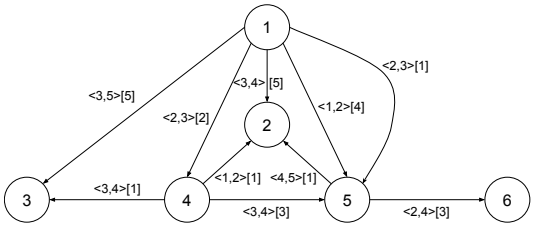

Figure 1: Example of a temporal graph.

In Figure 1, we show an example of a temporal graph. An edge from vertex 1 to vertex 2 with start time 3, arrival time 4 and edge weight 5 is shown by the label <3,4>[5].

2.2 Minimum Directed Spanning Tree in Temporal Graphs

In our research, we attempt to solve the minimum directed spanning tree problem on temporal graphs. This is a variant of the minimum spanning tree problem. The problem is defined by Huang *et al.* [13]. The input to this problem is a temporal graph $G = (V, E)$ and a root vertex $r \in V$. Every edge $e \in E$ has a weight $w(e)$. The goal is to extract a rooted tree T with the minimum total edge weight. We extract a tree T from the input. In the extracted tree T, there should be a temporal path from the root r to every vertex in the graph. In Figure 2, we show the solution for Figure 1 when the root is vertex 1.

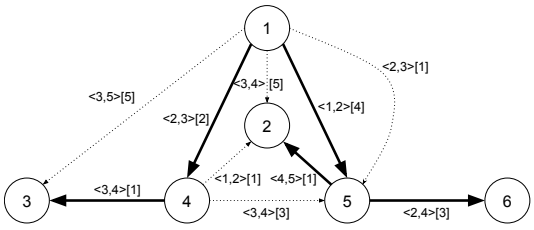

Figure 2: Example of a minimum directed spanning tree in a temporal graph.

The problem has been shown to be NP-hard by Huang *et al.* [13]. This was proved by constructing a reduction from the maximum leaf spanning tree problem, which is an NP-complete problem [12].

We note that it is easy to decide whether there exists a directed spanning tree in a temporal graph. A feasible directed spanning tree can be found by using Dijkstra's shortest path algorithm [7]. We will use this algorithm in order to generate a feasible solution.

2.3 Minimum Directed Steiner Tree

In Huang *et al.*'s method, an instance (*i.e.*, a temporal graph and a root) is first converted into a directed Steiner tree instance. The *minimum directed Steiner tree problem* is an optimization problem on a weighted directed graph. An instance of this problem consists of a weighted directed graph $G = (V, E)$, a root vertex $r \in V$ and a terminal vertex set $X \subset V$; the goal is to compute a subtree $T \in G$ with a minimum total edge weight such that every terminal vertex $x \in X$ is reachable from the root vertex r. This problem is known to be NP-hard [12].

2.4 Huang, Fu and Liu's Method

We review Huang, Fu and Liu's approximation algorithm. Their algorithm first converts an instance of the minimum spanning tree problem on temporal graphs into an instance of the directed Steiner tree problem.

The converted graph $\mathbb{G}$ is constructed as follows: We define the new vertex set $\mathbb{V}$ as $\mathbb{V} := \{ (h(e), a(e)) \mid e \in E \} \cup \{ (h(e), \infty) \mid e \in E \}$. That is, a vertex of the converted graph corresponds to a pair $(h(e), a(e))$ of the head vertex and the arrival time of an edge $e \in E$. For each vertex of the temporal graph G, we sort the vertex set of $\mathbb{G}$ having the same vertex as head of edges by the arrival time of edges. Let the sorted vertex set be $\mathbb{V}_v = \{ (h(e), a(e)) \mid h(e) = v \} = \{ v_1, v_2, \cdots, v_k \}$ for $v \in V$. We add zero cost edges between v_i and v_{i+1} for any $1 \leq i \leq k-1$. For each edge $e \in E$, we also add an edge from a vertex $u = (h(e'), a(e'))$ where

$$e' = \arg\max_{e' \in E} \{ a(e') \mid h(e') = t(e) \text{ and } a(e') \leq s(e) \}$$

to a vertex $v = (h(e'), a(e'))$ where

$$e' = \arg\min_{e' \in E} \{ a(e') \mid h(e) = h(e') \text{ and } a(e) \leq a(e') \}$$

with an edge weight $w(e)$. As a terminal vertex set for the directed Steiner tree problem, we assign

$$\mathbb{X} = \{ (h(e), \infty) \mid e \in E \}.$$

After converting the graph, the directed Steiner tree instance is preprocessed to satisfy the triangle inequality. In other words, if there are vertices u and v on the graph such that there is a path from u to v shorter than the direct edge u to v, then the edge weight between u and v is overwritten by the distance of the shortest path. This preprocessing step runs in time $O(|V||E| \log |V|)$. After preprocessing, they construct a directed Steiner tree in time $O(|V|^i k^i)$ with approximation ratio $O(i^2(i-1)k^{1/i})$ where $i \geq 2$ is an optimization parameter and k is the number of terminal vertices. Huang *et al.* improved upon Charikar, Chekuri, Cheung, Dai, Goei, Guha and Li's result by reducing running time from $O(|V|^i k^{2i})$ to $O(|V|^i k^i)$ while keeping the same approximation ratio [4].

3. ALGORITHM DESCRIPTION

In this section, we describe two algorithms that use integer linear programming. First, we introduce our formulation of the integer linear programming method for the directed minimum spanning tree problem on temporal graphs. Next, in order to compare another formulation with our formulation, we describe an integer linear programming formulation for the directed Steiner tree problem based on previous work [16, 20].

3.1 Integer Linear Programming Formulation for Minimum Directed Spanning Tree

We explain the integer linear programming formulation for the directed minimum spanning tree problem on temporal graphs. Our integer programming formulation for the minimum directed spanning tree problem is based on Behle *et al.*'s formulation [3]. Using inequalities based on Behle *et al.*'s method, we force an extracted graph to be a minimum weighted spanning tree through formulation (1), (2) and (7). We add other inequalities (5), (6) as temporal constraints and introduce additional binary variables to achieve a $O(|E| + |V|)$ initial formulation size.

First, we introduce a binary variable $x_e \in \{0, 1\}$ for each edge $e \in E$ to represent whether the edge e is used in an optimal solution ($x_e = 1$) or not ($x_e = 0$). The objective function of this problem is represented as:

$$\text{minimize} \sum_{e \in E} w(e) x_e. \tag{1}$$

This objective function represents the sum of the weight of edges used in a solution.

In addition, we introduce constraints to ensure that the subgraph that is induced by x_e is a tree. In a rooted tree, the indegree of each vertex must be 1 except for the root vertex r, which can be represented as:

$$(\forall v \in V \setminus \{r\}), \qquad \sum \{ x_e \mid h(e) = v \} = 1. \tag{2}$$

We also introduce time constraints in order to prevent the appearance of an invalid solution. If we formulate time constraints only using the variables x_e, the formulation is as follows:

$$\forall e \in \{ e \mid t(e) \neq r \}.$$
$$x_e \leq \sum \{ x_{e'} \mid t(e) = h(e') \text{ and } a(e') \leq s(e) \} \tag{3}$$

Here, $\sum A$ for a set A means $\sum A = \sum_{x \in A} x$. This formulation indicates that each edge e can be used in a solution if there is an edge e' having faster arrival time $a(e')$ than the start time of edge e in a solution. The size of this naïve formulation can become quadratic to the size of an input graph. To reduce the quadratic size of the constraints to a linear size in asymptotic notation, we introduce another integer variable y_e for each edge $e \in E$. The bottleneck to the formulation size bottleneck of above constraint is based in the right side of the inequality. By replacing the right side of the constraint 3 with a simple expression, we reduce the size of the inequality corresponding to the time constraints. In our formulation focusing on the small formulation size, a variable y_e represents the number of edges e' whose head $h(e')$ is same as a head of e and having a faster or equal arrival time $a(e')$ compared to the arrival time of edge e as

follows,

$$\forall e \in E.y_e = \sum \{\, x_{e'} \mid h(e) = h(e') \text{ and } (a(e'), e') < (a(e), e) \,\}. \quad (4)$$

In the comparison of these tuples $(a(e'), e')$ and $(a(e), e)$, we first compare the arrival time $a(e')$ and $a(e)$. If $a(e')$ and $a(e)$ are different, the comparison is determined by the magnitude of the relation between $a(e')$ and $a(e)$. Otherwise, we compare the indices of edges e and e' and determine the magnitude of the relation between the tuples $(a(e'), e')$ and $(a(e), e)$. From constraints (2) and (4), the variables y_e are either become 0 or 1. Thus, we can treat y_e as a binary variable. It is still possible for the above constraint to be quadratic in size. Therefore, we reduce the size of the summation on the right side of the constraints (4) using y_e,

$$Q(e) = \{\, (a(e'), e') \mid h(e) = h(e') \text{ and } (a(e'), e') < (a(e), e) \,\}$$

$$\forall e \in E.y_e = \begin{cases} x_e & (\emptyset = Q(e)) \\ x_e + y_{e'} & ((a(e'), e') = \max Q(e)). \end{cases} \quad (5)$$

To construct this constraint quickly, we make edge sets S_v for each vertex v. Edge set S_v contains all the edges e with head v. In each edge set S_v, we sort edges in advance by arrival time. After sorting edges, we can construct constraint (5) in time $O(|E|)$.

Using the above constraints, we rewrite constraint 3 as follows,

$$\forall e \in \{\, e \mid t(e) \neq r \,\}. x_e \leq y_{e'}, \quad (6)$$

where e' is

$$\arg\max_{e'} \{\, (a(e'), e') \mid h(e') = t(e) \text{ and } a(e') \leq s(e) \,\}.$$

The initial formulations used in this method are the constraints (2), (5) and (6). In the constraint (2), the number of non-zero elements is $O(|E|)$. Every variable x_e appears in only this constraint. In constraint (5), the number of equations is $O(|E|)$. In each equation, a constant number of variables appear. Therefore, the number of non-zero elements in this constraint is bounded by $O(|E|)$. Constraint (6) contains only 2 variables in each equation, and the number of equations is $|E|$. As a result, the number of non-zero elements of constraint (6) is $O(|E|)$. Constraints (2), (5) and (6) contain $O(|E|)$ non-zero elements in a coefficient matrix in the integer linear programming formulation. The size of the initial formulation used in this problem is $O(|E|)$.

However, the above constraints are not sufficient to obtain an optimal solution. We need constraints for the connectivity of the solution tree. In a rooted spanning tree, there should be a temporal path from a root vertex r to every vertex $v \in V$. In order to satisfy this condition, we add the following constraints [3, 20, 16, 19, 15],

$$\forall S \subset V, r \notin S, \emptyset \neq S. \quad \sum \{\, x_e \mid h(e) \in S, t(e) \notin S \,\} \geq 1. \quad (7)$$

This constraint prevents the appearance of a circular in a solution and is used in many combinatorial optimization problems with graphs to guarantee the connectivity of a solution. This constraint guarantees that there will be a temporal path from the root r to every $v \in V$ in the solution.

However, this addition causes the number of constraints to become exponentially large. Thus, we do not add this constraint in the beginning. When we solve our integer linear programming formulation, we use a linear sized method in our initial formulation. As in existing methods, when we solve our linear relaxation problem for the formulation and find that constraint (7) is not satisfied by the solution we add it to the formulation as a lazy constraint cut [3, 20, 16, 19, 15]. In order to find the unsatisfied constraints (7), we solve the maximum flow problem. For the maximum flow problem, we make another graph from the solution of the linear relaxation problem $\hat{x_e}$. The graph G_f that we use in the maximum flow problem has a vertex set $V_f = V$ and an edge set $E_f = E$. Each edge $e_f \in E_f$ has a flow capacity $cap(e) = \hat{x_e}$. In this graph G_f, we find the minimum cut from the root vertex $r_f = r$ to every vertex $v_f \in V_f$. When a minimum cut has a capacity greater than or equal to 1, the cut satisfies the constraint (7). Otherwise, the cut does not satisfy the constraint (7). In this case, we add the constraints (7) to the formulation and solve the linear programming relaxation problem repeatedly. In order to find the minimum cut from the root vertex r to every vertex v_f, we use Dinic's algorithm, which runs in $O(|V|^2|E|)$ [8]. Dinic's algorithm runs quickly in practice.

Algorithm 1 Calculate connected constraint

```
1: procedure CUT-GENERATION(G_f)
2:     for all e ∈ E_f do
3:         if x̂_e = 1 then
4:             union(h(e), t(e))
5:     for all v ∈ V_f do
6:         while find(r) ≠ find(v) do
7:             f ← maxflow(G_f, r, v)
8:             maxflow(G_f, v, r)
9:             if f < 1 then
10:                add cut C to formulation that corresponds
   to constraint 7.
11:                for all e ∈ C do
12:                    x̂_e ← 1
13:                    union(h(e), t(e))
14:            else
15:                break
```

Pseudo code for the cutting plane method used in our experiments is shown in Algorithm 1. After we run the maximum flow method in line 7, instead of recreating the whole graph G_f to reset the status of the graph, we calculate a reverse maximum flow in line 8 from the vertex v to a root r in order to speed up our cut generation method. In line 10, we add the cut found from the maximum flow problem. In line 11 to line 13, we increase the capacity of edges corresponding to the constraint. After increasing the edge capacity, we attempt to find multiple cuts. Another optimization is used in our implementation. If there is a temporal path from a root r to a vertex v in the solution $\hat{x_e}$, we can check the connectivity between the vertices r and v using a disjoint-set data structure [11]. In line 4, we merge a set including vertex $h(e)$ and a set including $t(e)$ if an edge e is used in the current linear programming problem solution. In line 6, we check whether there is a temporal path from the root r to a vertex v or not by using the disjoint-set data structure. If we find that there is a temporal path, we can skip forward

to find a cut using the maximum flow method because there is a flow with capacity larger than or equal to 1. Union and find operations for a disjoint-set data structure run in amortized $O(\alpha(n))$ where α is the inverse of the Ackermann function and n is the number of elements managed by the data structure. If we do not find any cut in Algorithm 1, a variable branch occurs. In a variable branch, the value of a variable that is not fixed is assigned and relaxation of the integer linear programming problem is solved recursively. In assignment, the variable is assigned a value of 0 or 1 and the formulation is solved recursively.

Finally, we give our integer linear programming formulation an initial solution to obtain a good upper bound. To calculate the initial solution, we run the modified version of Dijkstra's shortest path algorithm with a high priority for the arrival time of edges instead of the total path weights [7]. This type of path is called the earliest-arrival path by Wu *et al.* [24].

3.2 Directed Steiner Tree Based Exact Algorithm

In this section, we present another exact method for solving the minimum directed spanning tree problem on temporal graphs. We will solve the directed Steiner tree problem using integer programming. The formulation itself is based on the Steiner tree problem formulation by Koch and Martin [16] and the prize collecting Steiner tree problem formulation by Ljubic, Weiskircher, Pferschy, Klau, Mutzel and Fischetti [20]. In the directed Steiner tree problem, we use a variable x_e for each edge e. The objective function of this problem is the same as the objective function of the minimum spanning tree problem on temporal graphs: that is

$$\text{minimize} \sum_{e \in E} w(e) x_e. \tag{8}$$

We then add the indegree constraints. In the directed Steiner tree problem, we know that only the terminal vertices have exactly one indegree in the optimal solution.

$$\forall v \in X. \qquad \sum \{ x_e \mid h(e) = v \} = 1. \tag{9}$$

In the directed Steiner tree problem, we can add outdegree constraints. If there is an in-edge to a non-terminal vertex v, the vertex v must have at least one out-edge. In a directed Steiner tree problem, leaf vertices of an optimal solution do not include non-terminal vertices because if there are non-terminal vertices in the leaf nodes of the solution, we decrease the total edge cost by deleting such vertices and edges connected to the vertices from the solution. The corresponding constraints for this property are:

$$\forall v \in V \setminus X, v \neq r \qquad \sum \{ x_e \mid h(e) = v \} \leq \sum \{ x_e \mid t(e) = v \}. \tag{10}$$

This constraint (10) means that if the non-terminal vertex v is used in a solution, the outdegree of the vertex v is at least 1 in a directed Steiner tree. For the root vertex r, the outdegree of the vertex is at least 1 when there is a non-root terminal vertex x in a graph [20, 16],

$$1 \leq \sum \{ x_e \mid t(e) = r \}. \tag{11}$$

In order to make the formulation stronger, we add another constraint to the directed Steiner tree problem. When there is an edge e used in a solution, the tail vertex $t(e)$ of the edge e should have an in-edge when the tail vertex $t(e)$ is not a root vertex r [20, 16].

$$\forall e \in E, t(e) \neq r. \qquad \sum \{ x_{e'} \mid h(e') = t(e) \} \geq x_e \tag{12}$$

From the above constraint, the sum of variables corresponding to in edges should be greater than or equal to any variable x_e corresponding to edge e coming from the vertex $t(e)$.

Initially, we use only the above constraints. These constraints are not sufficient to guarantee the connectivity of the solution of directed Steiner tree. We use the following lazy constraint to guarantee the connectivity of the directed Steiner tree, similar to what was used in the minimum spanning tree problem for the temporal graph formulation [20, 16].

$$\forall S \subset V, r \notin S, \emptyset \neq S \cap X. \qquad \sum \{ x_e \mid h(e) \in S, t(e) \notin S \} \geq 1. \tag{13}$$

In the directed Steiner tree problem, there should be a path from the root vertex r to every terminal vertex $v \in X$. We add these constraints if there is a constraint that is not satisfied in the solution to the integer linear programming relaxation problem. In order to find the unsatisfied constraint, we use the maximum flow problem in the same way as used above for the minimum spanning tree problem in temporal graphs. For the maximum flow problem, we create a graph in which every edge has a flow capacity $cap(x_e)$ equal to the solution $\hat{x_e}$ of the linear relaxation problem.

Algorithm 2 Calculate connected constraint for the directed Steiner tree problem

```
1: procedure CUT-GENERATION(G_f)
2:     for all e ∈ E_f do
3:         if x̂_e = 1 then
4:             union(h(e), t(e))
5:     for all v ∈ X do
6:         while find(r) ≠ find(v) do
7:             f ← maxflow(G_f, r, v)
8:             maxflow(G_f, v, r)
9:             if f < 1 then
10:                add cut C to formulation corresponds to
   constraint 13.
11:                for all e ∈ C do
12:                    x̂_e ← 1
13:                    union(h(e), t(e))
14:            else
15:                break
```

One difference between Algorithm 1 and Algorithm 2 is in a line 5. In a directed Steiner tree problem, we check only connectivity between a root vertex r and the terminal vertices $v \in X$. Therefore, we check only the maximum flow from the root vertex r to every terminal vertex $v \in X$.

Finally, we give an initial solution for the integer linear formulation of the directed Steiner tree problem. In order to calculate an initial solution, we construct a shortest path tree from the root vertex r in the converted directed Steiner tree graph. After constructing a shortest-path tree, we repeatedly remove the leaf vertices that are not terminal ver-

tices from the shortest-path tree until all the leaf vertices become terminal vertices. Therefore, the initial solution for the directed Steiner tree problem has only terminal leaf vertices. This tree is a feasible solution to the directed Steiner tree problem.

4. EXPERIMENTS

4.1 Environment

In our experiments, all programs are run on Cent OS 5.10 with an Intel Xeon X5675 3.07 GHz CPU, and 283 GB memory. All programs are implemented in C++11 and compiled by GNU Compiler 5.2.0 with the -O3 option. We also use ILOG CPLEX 12.6 developed by IBM in order to solve the integer linear programming problem. The CPLEX parameter for the number of threads was set to 1.

4.2 Datasets

In our experiments, we use datasets from real-world networks. The graph type, vertex size, and edge size of the subgraphs are listed in Table 1. All datasets were downloaded from the Koblenz Large Network Collection[1]. *HepPh* is the co-authorship network in arXiv's High Energy Physics Phenomenology section. *DBLP* (Digital Bibliography & Library Project) is a co-authorship network in the DBLP computer science bibliography. *Enron* is an email communication network of the company Enron. *Epinions* is a trust and distrust network in an online product rating site. *Facebook* is a communication network in one of the most famous social network services. *Slashdot* is a communication network in a news website focusing on technology and science.

Table 1: Dataset information.

Dataset	$\|V\|$	$\|E\|$	type
HepPh	28093	4596803	co-authorship
DBLP	1314050	18986618	co-authorship
Enron	87273	1148072	communication
Epinions	131828	841372	social
Facebook	46952	876993	communication
Slashdot	51083	140778	communication

It is possible that the graphs do not have a feasible temporal spanning tree. In order to obtain a feasible solution from the above datasets in our experiments, we extract graphs that have a feasible solution. We use two methods to extract the data. One is the method that Huang *et al.* used in their experiment [13]. Another method is used to obtain some of the larger datasets and evaluate properties of our method and compare in running time and approximation ratio of the methods. In the datasets, the edges have a time stamp but no edge weight. Therefore, we use the same time stamp for the starting time and arrival time for each edge. In the model of [13, 14], the propagation probability of an edge is the inverse of the vertex degree connected by the edge. We then assign a weight $\log(d^+(t(e)))$ for all edges e where $d^+(v)$ is the outdegree of vertex v. In the assignment, the minimum spanning tree represents the maximum influence path from the root vertex in the dataset [13, 14].

In the research of Huang *et al.*, the authors generate a graph as follows. First, they restrict the time interval in which edges appeared on the graph. They used a time interval of one-tenth of the whole period of the original datasets. Second, they calculated the set of reachable vertices from a root vertex in the time restricted temporal graph generated the first phase. If they found that the root vertex had reached more than ten percent of the vertices, they then extracted the subgraph including the reachable vertices as a dataset. The graph size of each dataset is shown in Table 2 without a subscript.

In addition to the datasets from Huang *et al.*'s generation method, we generate more datasets to evaluate the methods extensively. In our method, we sort an edge set E with the time stamp of each edge. After sorting, we extract a graph induced by the edge set E_k where E_k is the subset of edge set E such that it contains the top-k edges. Then, we calculate the reachable vertices from a randomly chosen root vertex. We generate a dataset from the reachable vertex set. 10^4, 10^5 and 10^6 are all used as parameter k for different datasets. In Table 2, the parameter k is shown as dataset$_k$.

4.3 Experimental Results

In our experiment, we compare our integer linear programming formulation for a minimum directed spanning tree on temporal graphs, with the integer linear programming formulation for converted directed Steiner tree problems and Huang *et al.*'s approximation algorithm. We set a hour time limit for the methods that use the integer linear programming and the preprocessing part of Huang *et al.*'s method. We use Huang *et al.*'s implementation written in C++. We modify the preprocessing part of their implementation to improve time complexity from $O(|V|^3 + |V||E|\log|V|)$ to $O(|V||E|\log|V|)$.

Table 3 shows the cost of solutions and the execution times for the three methods. In columns *UB*, *LB* and time(s) of Our IP, we show the upper bound and the lower bound of the solution and the execution time of our method, respectively. In the columns UB, LB and time(s) of DST IP, we show the upper bound and lower bound of the solution and the execution time of a directed Steiner tree problem that is converted from the original instances. In the column cost and time(s) of Huang *et al.* [13], we show the cost of solutions and the total running time of Huang *et al.*'s approximation method. The lower bound of the Enron$_{10^5}$ dataset from our method is wrong. It seems that a numerical error in computation of the linear programming problem causes this incorrect result, see * in Table 3. Akiba and Iwata report a similar problem when they solve the vertex cover problem using CPLEX [1].

From this table, the methods based on the integer linear programming problem compute the optimal solution for small case. Our method computes the optimal solution for 16 instances and Steiner tree formulation computes the optimal solution for 13 instances. For the instances that our method computes the optimal solution, the method takes less than 10 minutes. Our method tends to obtain a better solution and approximation ratio than the directed Steiner tree formulation. In addition, our formulation runs faster than the directed Steiner tree formulation in most instances. This occurs when the size of converted instances becomes large. We found that only our formulation computes the optimal solution for the Enron$_{10^6}$, Epinions, the Facebook$_{10^5}$, and Slashdot$_{10^6}$ networks. In some instances, our method takes more time to calculate worse solutions than Huang

[1]http://konect.uni-koblenz.de/

Table 2: Size of the generated datasets. $|V_{dst}|$, $|E_{dst}|$ are the graph sizes of a converted directed Steiner tree instance. $|X_{dst}|$ is the number of terminals in a directed Steiner tree instance. $|E_{prep}|$ is size of edges after of Huang *et al.*'s preprocessing method. "-" means that preprocessing does not finish within one hour.

Dataset	$\|V\|$	$\|E\|$	$\|V_{dst}\|$	$\|E_{dst}\|$	$\|X_{dst}\|$	$\|E_{prep}\|$
HepPh	146	3706	845	4165	146	100848
HepPh_{10^4}	189	1613	722	1990	189	34658
HepPh_{10^5}	2534	79872	15449	90455	2534	20451111
HepPh_{10^6}	9111	939482	91063	1016019	9111	-
DBLP	4795	29426	14177	38808	4795	59779483
DBLP_{10^4}	152	970	476	1294	152	32547
DBLP_{10^5}	3031	19783	10691	27443	3031	18592360
DBLP_{10^6}	52349	426763	196611	571025	52349	-
Enron	722	4921	5642	9746	722	4181528
Enron_{10^4}	656	4180	4824	8242	656	2909012
Enron_{10^5}	6522	80813	87248	157884	6522	1216033144
Enron_{10^6}	38672	883430	921028	1734565	38672	-
Epinions	863	7111	6990	12329	863	10933013
Epinions_{10^4}	1916	3362	3831	5277	1916	211304
Epinions_{10^5}	16620	90550	33239	107169	16620	62970494
Epinions_{10^6}	55117	638208	244224	815877	55117	-
Facebook	1430	14521	15949	27326	1430	10508416
Facebook_{10^4}	171	789	960	1412	171	43514
Facebook_{10^5}	5143	74514	79640	140672	5143	618767173
Facebook_{10^6}	32593	817175	849649	1589700	32593	-
Slashdot	898	2964	3815	4921	898	338336
Slashdot_{10^4}	373	1023	1381	1686	373	39653
Slashdot_{10^5}	4122	25614	29172	45153	4122	143462973
Slashdot_{10^6}	8607	59285	66369	104957	8607	905286427

et al.'s method, however our method gives us how far the solution is away from the optimal solution. The approximation ratios from upper bound and lower bounds of our method are roughly 30% in DBLP_{10^6}, except in the case of the Enron_{10^5} network. The approximation ratios of other instances is at most 0% to 20% in our formulation. Compared to Huang *et al.*'s method, our method computes the optimal solutions in less time in 12 instances. Huang *et al.*'s method is more time consuming when it makes the transitive closure in preprocessing. From this result, we demonstrate that many practical networks can be quickly solved using integer programming.

5. CONCLUSION

We studied the minimum directed spanning tree problem on temporal graphs, which is NP-hard. We then proposed a method that can compute optimal solutions using integer programming. Through experiments, we compared our minimum directed spanning tree formulation, the directed Steiner tree formulation, and Huang *et al.*'s approximation method. From experimental results, we showed that our method computes the optimal solution for some instances generated from real-world networks. For some datasets used in our experiment, we found the optimal solutions faster than Huang *et al.*'s approximation method. From these results, we believe that there may be an undiscovered property of real-world networks that makes the problem easy.

6. REFERENCES

[1] T. Akiba and Y. Iwata. Branch-and-reduce exponential/fpt algorithms in practice: A case study of vertex cover. In *ALENEX*, pages 70–81, 2015.

[2] D. L. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook. *The Traveling Salesman Problem: A Computational Study (Princeton Series in Applied Mathematics)*. Princeton University Press, Princeton, NJ, USA, 2007.

[3] M. Behle, M. Jünger, and F. Liers. A primal branch-and-cut algorithm for the degree-constrained minimum spanning tree problem. In *WEA*, pages 379–392, 2007.

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

[5] Y.-J. Chu and T.-H. Liu. On shortest arborescence of a directed graph. *Scientia Sinica*, 14(10):1396, 1965.

[6] G. B. Dantzig, D. R. Fulkerson, and S. M. Johnson. On a linear-programming, combinatorial approach to the traveling-salesman problem. *Operations Research*, 7(1):58–66, 1959.

[7] E. W. Dijkstra. A Note on Two Probles in Connexion with Graphs. *Numerische Mathematik*, 1:269–271, 1959.

[8] Y. A. Dinitz. Algorithm for solution of a problem of maximum flow in a network with power estimation. *Soviet Math Doklady*, 11:1277–1280, 1970.

[9] J. Edmonds. Optimum branchings. *Journal of Research of the National Bureau of Standards Section B Mathematics and Mathematical Physics*, 71B(4):233, 1967.

[10] 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(2):109–122, 1986.

[11] B. A. Galler and M. J. Fisher. An improved equivalence algorithm. *Commun. ACM*, 7(5):301–303, 1964.

[12] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.*

Table 3: Results from three methods. UB and LB denote the upper bound and lower bound from the IP.

Dataset	Our IP			DST IP			Huang *et al.* [13]	
	UB	LB	time(s)	UB	LB	time(s)	cost	time(s)
HepPh	359.97	359.97	0.08	359.97	359.97	0.23	408.92	0.15
HepPh_{10^4}	370.67	370.67	0.10	370.67	370.67	0.21	463.06	0.06
HepPh_{10^5}	6347.86	6347.86	53.15	6347.86	6347.86	30.59	8699.85	37.44
HepPh_{10^6}	26020.00	26008.90	3609.44	26026.40	26010.00	3632.21	-	-
DBLP	10920.90	9102.36	3608.81	13691.10	8922.72	3600.29	11707.79	108.45
DBLP_{10^4}	307.31	307.31	0.77	307.31	307.31	6.57	329.57	0.06
DBLP_{10^5}	7536.32	6395.41	3602.30	9835.15	6315.10	3602.10	7951.61	41.49
DBLP_{10^6}	137808.00	106916.00	3614.50	193000.00	92941.90	3606.54	-	-
Enron	3458.84	3458.84	0.13	3458.84	3458.84	0.43	3556.57	5.91
Enron_{10^4}	3139.24	3139.24	0.09	3139.24	3139.24	0.30	3229.24	3.91
Enron_{10^5}	35872.30	35872.30*	42.09	30998.40	30998.40	91.61	32570.17	1969.99
Enron_{10^6}	228180.00	228180.00	274.07	240191.00	194650.00	3620.18	-	-
Epinions	2303.57	2303.57	370.63	3134.27	2273.22	3600.21	2600.98	15.14
Epinions_{10^4}	7758.97	7758.97	0.06	7758.97	7758.97	0.21	7896.01	0.76
Epinions_{10^5}	76764.20	67075.80	3602.47	77054.30	67075.60	3602.81	71377.72	172.75
Epinions_{10^6}	201130.00	167676.00	3611.22	222774.00	167574.00	3653.55	-	-
Facebook	4195.00	4195.00	0.55	4195.00	4195.00	6.43	4342.42	20.46
Facebook_{10^4}	519.27	519.27	0.02	519.27	519.27	0.04	529.36	0.10
Facebook_{10^5}	14938.30	14938.30	34.61	14941.50	14934.50	3603.49	16409.22	1009.70
Facebook_{10^6}	95122.40	94723.50	3609.64	160996.00	91291.80	3632.54	-	-
Slashdot	2233.99	2233.99	0.06	2233.99	2233.99	0.12	2315.42	0.97
Slashdot_{10^4}	885.33	885.33	0.02	885.33	885.33	0.03	896.95	0.12
Slashdot_{10^5}	13362.50	13362.50	15.47	13362.50	13362.50	4.98	14398.79	215.53
Slashdot_{10^6}	27996.10	27996.10	13.86	31122.20	27841.80	3601.43	31065.19	1540.95

W. H. Freeman, 1979.

[13] S. Huang, A. W. Fu, and R. Liu. Minimum spanning trees in temporal graphs. In *SIGMOD*, pages 419–430, 2015.

[14] D. Kempe, J. M. Kleinberg, and É. Tardos. Maximizing the spread of influence through a social network. In *SIGKDD*, pages 137–146, 2003.

[15] G. W. Klau, I. Ljubic, A. Moser, P. Mutzel, P. Neuner, U. Pferschy, G. R. Raidl, and R. Weiskircher. Combining a memetic algorithm with integer programming to solve the prize-collecting steiner tree problem. In *GECCO*, pages 1304–1315, 2004.

[16] T. Koch and A. Martin. Solving steiner tree problems in graphs to optimality. *Networks*, 32(3):207–232, 1998.

[17] J. B. Kruskal Jr. On the shortest spanning subtree of a graph and the traveling salesman problem. *American Mathematical Society*, 7(1):pp. 48–50, 1956.

[18] G. Laporte. The traveling salesman problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59(2):231–247, 1992.

[19] I. Ljubic, R. Weiskircher, U. Pferschy, G. W. Klau, P. Mutzel, and M. Fischetti. Solving the prize-collecting steiner tree problem to optimality. In *ALENEX*, pages 68–76, 2005.

[20] I. Ljubic, R. Weiskircher, U. Pferschy, G. W. Klau, P. Mutzel, and M. Fischetti. An algorithmic framework for the exact solution of the prize-collecting steiner tree problem. *Math. Prog.*, 105(2-3):427–449, 2006.

[21] C. E. Miller, A. W. Tucker, and R. A. Zemlin. Integer Programming Formulation of Traveling Salesman Problems. *Journal of the ACM*, 7(4):326–329, 1960.

[22] R. C. Prim. Shortest connection networks and some generalizations. *Bell System Technical Journal*, 36(6):1389–1401, 1957.

[23] S. Wang, W. Lin, Y. Yang, X. Xiao, and S. Zhou. Efficient route planning on public transportation networks: A labelling approach. In *SIGMOD*, pages 967–982, 2015.

[24] H. Wu, J. Cheng, S. Huang, Y. Ke, Y. Lu, and Y. Xu. Path problems in temporal graphs. *Proc. VLDB Endow.*, 7(9):721–732, 2014.

[25] H. Wu, J. Cheng, Y. Lu, Y. Ke, Y. Huang, D. Yan, and H. Wu. Core decomposition in large temporal graphs. In *IEEE BigData*, pages 649–658, 2015.

[26] H. Wu, Y. Huang, J. Cheng, J. Li, and Y. Ke. Reachability and Time-Based Path Queries in Temporal Graphs. In *ICDE*, 2016. to appear.