2-Approximation Algorithm for the Minimum Weighted Steiner Tree Problem

Anh T. Tran and Tam T. Nguyen

Econometrics and Operations Research, School of Business and Economics, Maastricht University

Abstract

The Minimum Spanning Tree problem is well-known and has been studied extensively. The solution to this problem spans all vertices of a graph. Nonetheless, a more generalized problem - the Steiner Minimal Tree problem - is yet to be delved into thoroughly. The solution to this problem spans only a required subset of vertices of a graph. In this paper, we describe several algorithms to solve the Steiner Minimal Tree problem, and investigate specifically how the the Steiner Minimal Tree problem can be solved using a 2-approximation algorithm, with an application in essentially large instances. Due to the NP-hardness of the problem, approximation algorithms prove to be the sole feasible solution, unless the given instances are exceptionally small. We also discuss data structures that optimize the approximation algorithm for large graphs, and evaluate the time complexity of our implementation of the algorithm.

Keywords: optimization, minimum spanning tree, minimum Steiner tree, time complexity, algorithm

1 Introduction

The Steiner Minimal Tree (STM) problem is a generalization of two other well-known problems in combinatorial graph optimization, namely the Minimum Spanning Tree (MST) problem and the (non-negative) Shortest Path (SP) problem. In the STM problemfor a given graph, the vertices are divided into two sets: *terminals* and *nonterminals*. The terminals are the vertices that must be included in the solution. The cost of a Steiner tree is defined as the total weight of the edges in the tree. Unlike the MST which must span all vertices of a given graph, the SMT may contain some nonterminals besides the terminals, to reduce the total cost of the tree. Interestingly, if a Steiner tree consists of exactly two terminals, the SMT problem reduces to finding the shortest path between the said vertices in the graph; on the other hand, if all vertices are terminals, then the problem boils down to finding the MST. Consequently, the SMT problem possesses fascinating properties of the two smaller and simpler problems. Steiner trees have crucial applications in VLSI routing [1], wirelength estimation [2], phylogenetic tree reconstruction in biology [3], and network routing [4]. The SMT Problem is NP-hard even in the Euclidean or rectilinear metrics [5], and its decision variant is NP-complete by a transformation from the Exact Cover by 3-Sets problem [6].

The SMT problem is formally defined as follows:

Definition 1.1: *Given an undirected graph $G = (V, E, w)$, where w is the weight function of G, and a subset of vertices Y consisting of terminals of V, the Steiner Minimal tree problem is to find the subtree that spans at least all of the vertices in Y, while having the minimum total edge cost.*

Various variants and generalizations of the SMT problem have been studied in past literature. Some of the most common variants are listed below:

- *Eucledian Steiner Tree problem* [7] (ESTP): Given a finite set Y of points in the (Euclidean) plane, find a point set R and a minimum spanning tree T for R, such that the length of T is minimized in the ℓ_2-norm[1] (Euclidean distance).

- *Rectilinear Steiner Tree problem*[8] (RSTP): Given a finite set Y of points in the (Euclidean) plane, find a point set R and a minimum spanning tree T for R, such that the length of T is minimized in the ℓ_1-norm[2] (Manhattan/rectilinear distance).

[1] For 2 points $a = (x_1, y_1)$ and $b = (x_2, y_2)$, the distance in ℓ_2-norm (Euclidean distance) is $d(a, b) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$

[2] For 2 points $a = (x_1, y_1)$ and $b = (x_2, y_2)$, the distance in ℓ_1-norm (Manhattan/rectilinear distance) is $d(a, b) = |x_1 - x_2| + |y_1 - y_2|$

- *Genralized Steiner Tree problem* [9]: Given an undirected graph $G = (V, E, w)$, a subset $Y \subseteq V$, and partitions $Y_1, Y_2, ..., Y_k$, find a minimum tree T such that at least one point from each partition is in T.

- *Weighted Steiner Tree problem* [10]: Given an undirected graph $G = (V, E, w, c)$, where c is a weight vector of vertices, and a subset $Y \subseteq V$, find a tree T spanning all the points in Y, such that the total weight of edges and vertices in T is minimum.

Various algorithms are used to solve the SMT problem, such as exact algorithms [11], heuristic algorithms [12], and approximation algorithms [13]. This paper focuses mainly on a 2-algorithm algorithm for the SMT problem. Before explaining said algorithm in Section 2, a brief overview of the classical dynamic programming approach developed by Dreyfus and Wagner to find the exact solution to the SMT problem is first given in the same section. Results of the approximation algorithm for the SMT problem will be discussed in Section 3.6.

2 Algorithms for the SMT problem

2.1 Exact algorithm

Since the SMT problem is in NP, one could expect an *exact* algorithm to have an exponential running time in the worst case; one such simple algorithm is *exhaustive search*, which is to check all possible trees until the optimum tree is obtained. The well-known exact algorithm for the SMT problem, developed by Dreyfus and Wagner, will be briefly introduced.

The Dreyfus-Wager algorithm [14] is a dynamic programming algorithm which recursively computes the length of a minimum Steiner subtree for a given terminal set Y from its minimum subtrees. To start, let $k = |Y|$, i.e. the number of terminals. In fact, we can use a small parameter k to find a solution in polynomial time with respect to the input size, and in exponential time with respect to k [15]. Such a family of algorithms is called *fixed parameter tractable*, or *parameterized algorithm*. [16, 17].

Input: $G = (V, E, w)$, $Y \subseteq V$;
Output: A minimum total edge cost of a subtree $T = (V', E')$, $Y \in V'$;

Compute P_{vw} for all $w, v \in V$;
for $(i = 2, \ldots, k-1)$ **do**
  For any $|X| = i$ and any $w \in Y$, $v \in V$:
    $T(X \cup Y) = min\{P_{vw} \cup T(X' \cup w) \cup T(X'' \cup w)\}$
end

Algorithm 1: Dreyfus-Wagner

The algorithm works as follows. Starting from a terminal leaf[3] v, the Steiner tree is $T(X \cup v)$ for some $X \subset Y$. Then, v is joined by the shortest path P_{vw} to an interior node $w \in T(V)$ with $d(w) \geq 3$.[4] Node w splits $T \setminus P_{vw}$ into two subgraphs: $T(X' \cup w)$, and $T(X'' \cup w)$. By finding the minimum shortest path and the two component recursively, we get a Steiner tree as the exact solution to the SMT problem.

Finding the shortest path can be done using *Floyd-Warshall* algorithm, which runs in $O(n^3)$ time [18]. For the next step, finding a minimum for a given $X \subset Y$ of size $|X| = i$ can be done in $O(n * 2^i)$ time. Nonetheless, because we are seeking a terminal node with degree of at least 3, the complexity of this step is $O(n * 3^k)$. In total, the complexity of the algorithm is $O(n^3 + n * 3^k) = O(3^k)$.

[3] A leaf is a node of degree 1
[4] w can be terminal, i.e. $w \in X$, or non-terminal, i.e. $w \notin X$.

2.2 Approximation algorithm

An MST is widely used to approximate an SMT [19]. First, a metric closure [5] is constructed on the set of terminals Y of the original graph G. The metric closure is a complete graph with terminals Y as vertices, and edge costs equal to the lengths of the corresponding shortest paths in G. Then, an MST is constructed from the metric closure. Lastly, the MST is transformed to a Steiner tree by 'merging' all shortest paths (in G) between every pair of terminals that are connected by an edge in the MST. Some post-processing may be done to remove any possible cycles.

Input: $G = (V, E, w)$, $Y \in V$;
Output: a minimum total edge cost of a subtree $T = (V', E')$, $Y \in V'$;

Construct the metric closure G_Y on Y;
Find an MST T_Y on G_Y;
Set $T \leftarrow \emptyset$;
for *each edge* $e = (u, v) \subset E(T_Y)$ **do**
  Find a shortest path P from u to v on G
  if P *contains less than two vertices in* T **then**
    Add P to T
  end
  else
    Let p_i and p_j be the first and last vertices already in T
    Add subpaths from u to p_i and from p_j to v
  end
end

Algorithm 2: Approximation

A sample execution of the approximation algorithm is illustrated in Figure 1.

2.2.1 Approximation Ratio

We will evaluate the quality of the solution of this algorithm in theory. Let $G_Y = (Y, E(G_Y), \bar{w})$. Note that $w(T) \leq T_Y$, since at most all of the shortest paths between each pair of terminals are inserted. We want to compare T_Y with an SMT. Let T_Y^* be the SMT. Consider a Eulerian tour[6] X on the doubling tree (see Figure 2 for the definition of a Eulerian tour on a doubling tree). Since the tour passes each edge exactly twice, we have $w(X) = 2(T_Y^*)$. On the other hand, since the tour visits all the terminals in the graph, we have

$$w(X) \geq w(tsp(G_Y)) \geq w(T_Y),$$

$$w(T) \leq w(T_Y) \leq 2 \times w(T_Y^*)$$

To see that the upper bound is asymptotically tight, consider an extreme example (Figure 3). Suppose that there are k terminals, and 1 nonterminal. The terminals form a cycle with $w(e) = 2$ for each edge e in the cycle. The nonterminal is adjacent to each terminal with an edge of length 1. The SMT is the star centered at the nonterminal in the middle of the graph and has total cost k. Meanwhile, an MST is the path consisting of the terminals and has total cost $2(k-1)$. Therefore, the *Steiner ratio* is

$$\rho = \frac{w(MST(G_Y))}{w(SMT(G_Y))} = \frac{2(k-1)}{k} = 2 - \frac{2}{k}$$

Hence, the approximation is a 2-approximation. The Steiner ratio in general metric is 2, and is smaller in the Euclidean metric as the Euclidean space is a restricted version of the general metric.

2.2.2 Time Complexity

The time complexity of the algorithm is $O(nk^2)$. The majority of the running time is due by the construction of the metric closure.

[5] also known as a metric completion

[6] An Eulerian cycle, also called an Eulerian tour, on a graph is a trail starting and ending at the same vertex. In other words, it is a graph cycle which uses each edge exactly once.

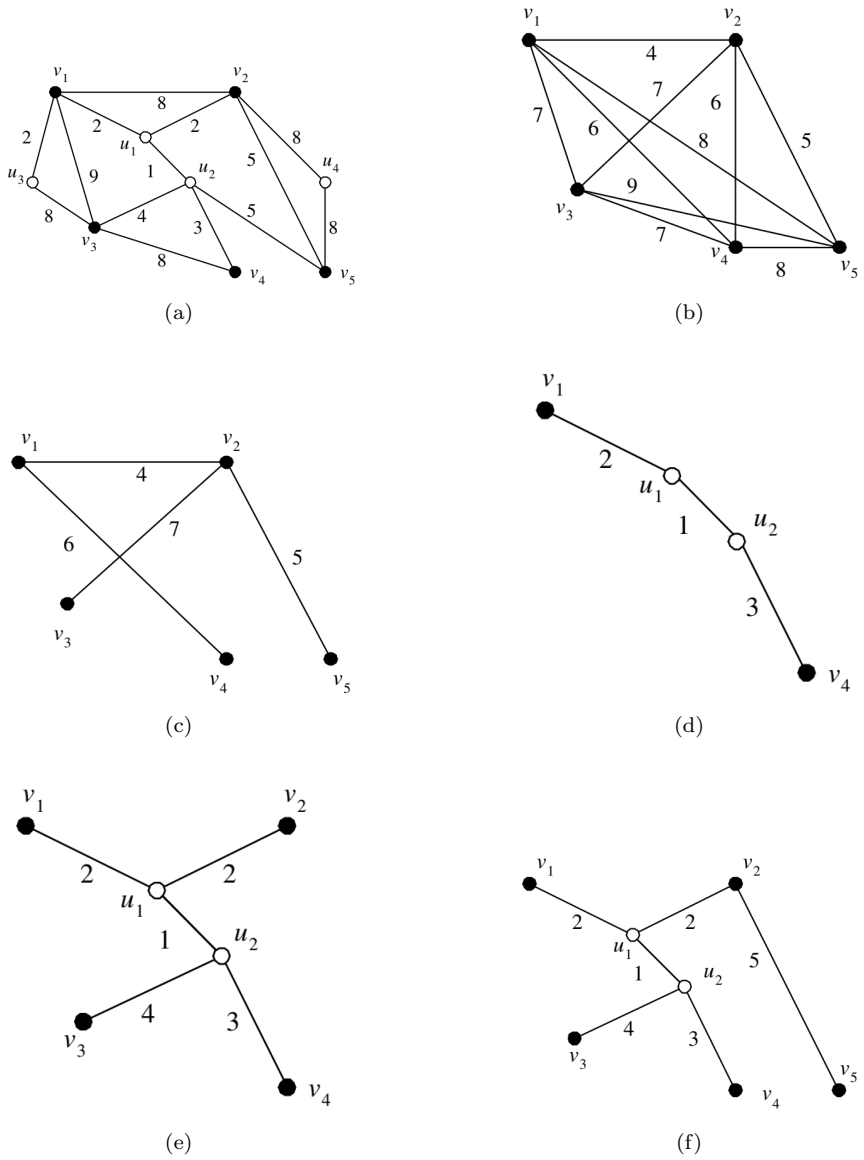

Figure 1: A sample execution of the approximation algorithm. (a) is the given graph G, in which $Y = v_1, v_2, v_3, v_4, v_5$ is the terminal set and u_i, $1 \leq i \leq 4$ are nonterminals. (b) and (c) are the metric closure G_Y and an MST T_Y on G_Y respectively. Initially, tree T is empty. Suppose that edge $(v_1, v\ 4) \in E(T_Y)$ is first chosen edge. The corresponding shortest path in G is (v_1, u_1, u_2, v_4). Since all vertices on the path are not in T, the whole path is inserted into T (d). At the second iteration, edge $(v_2, v_3) \in E(T_Y)$ is chosen. The corresponding shortest path is (v_2, u_1, u_2, v_3). However, u_1 and u_2 are already in T. Therefore, only (v_2, u_1) and (u_2, v_3) are inserted (e). At the third iteration, edge (v_2, v_5) is chosen, and edge (v_2, v_5) is inserted (f). Finally, the last edge in T_Y is (v_1, v_2). Since both v_1 and v_2 are already in T, no edge is inserted, and the tree in (f) is the output tree.

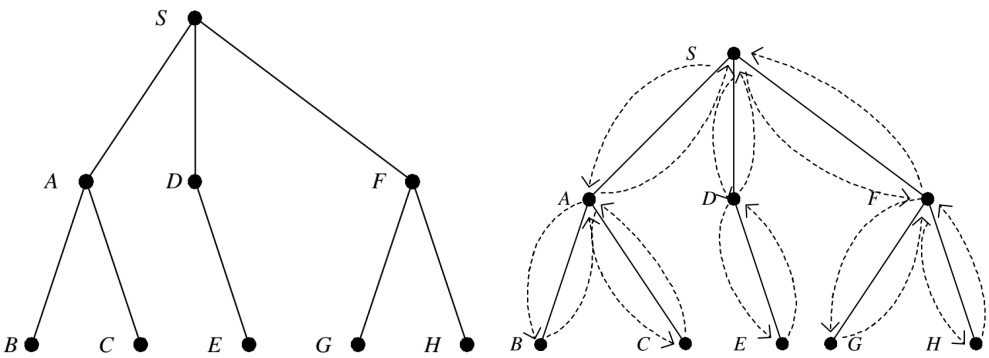

Figure 2: The Eulerian cycle on a doubling tree.

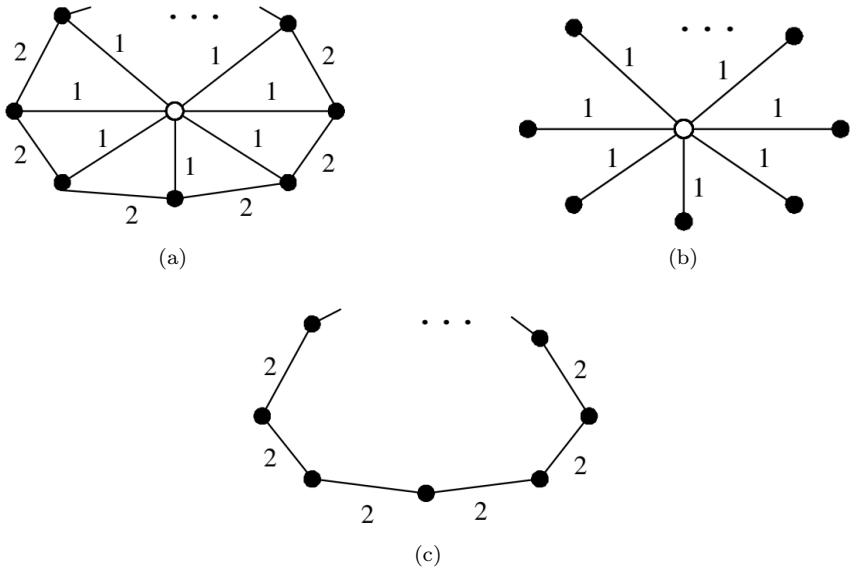

Figure 3: (a) A graph in which the white vertex is a nonterminal, and others are terminals. (b) An SMT. (c) An MST of the subgraph induced on the terminals.

3 Implementation

3.1 Motivation

Undoubtedly, difficulties will arise when the exact algorithm is used for large instances. Indeed, no exact algorithm for solving the SMT problem has been claimed as efficient enough to solve large-scale SMT instances so far[3, 19]. Obtaining a satisfactory output within a certain time frame becomes increasingly difficult given restricted computer resources and the lack of state-of-the-art processors. That being said, even if the obtained solution is not optimal, we still aim to able to approximate the SMT solution while satisfying the preferable expected bound. Thus, the *approximation algorithm* is chosen to serve this purpose, due to the feasibility and relative ease in implementing it.

3.2 Data structure

The programming language we use is Java. Recognizing that we need to implement an SP algorithm and an MST algorithm, as well as construct the metric closure and the final Steiner tree, we opt for adjacency matrices[7] to store our graphs. While the use of adjacency matrices facilitates indexing graphs and retrieving relevant values such as edge costs, for large instances of the MST problem, the graphs created by the approximation algorithm may end up highly *sparse*[8] yet still occupy much computational memory. In Java, there is no direct and immediate way to free memory. Ways to tackle the problem of memory overuse include reusing existing graph variables in the course of the algorithm, and returning only the total edge cost of the Steiner tree solution instead of the tree itself. Nonetheless, depending on our needs when solving the SMT problem, the desired output of the implemented approximation algorithm may differ.

3.3 Classes, variables and methods

In our Java implementation, the following classes, variables methods are used:

Class	Variables	Methods
DijkstraResult	final int[] shortestDistances final int[] previousNode	constructor getters void printShortestDistances void printParents
MetricResult	final int[][] metricClosure final List<List<Integer>>paths	constructor getters
MSTResult	final int[][] MST int totalCost	constructor getters
SteinerResult	final int[][] steiner int totalCost final int MSTCost final int V int E	constructor getters
Main	final int NO_PARENT = −1	main void writeGraphToFile SteinerResult steiner MetricResult metricClosure MSTResult MST List¡Integer¿ getPath DijkstraResult dijkstra void printGraph List¡Integer¿ terminalList int[][] createGraph

- **DijkstraResult** stores the shortest path outputs from the Dijkstra's Algorithm. The shortestDistances array stores the shortest distance between a starting node to every node in the input graph

[7] An adjacency matrix is a square matrix used to represent a finite graph. The elements of the matrix indicate whether pairs of vertices are adjacent or not in the graph and, if they are, the cost of the edge connecting those vertices.

[8] A sparse graph is a graph with only a few edges. Its adjacency matrix form is characterized by the domination of zeros.

G. The distance from the node to itself is assigned a value of -1. The previousNode array stores the preceding vertex in the shortest path. With this array, the necessary shortest paths can be returned by backtracking the array from the end node to the starting node. This class also include standard methods such as getters and print methods.

- **MetricResult**, similarly, stores the metric closure graph generated from the original input graph. Since Dijkstra's algorithm is used in the metric closure step in the SMT algorithm, this class also stores a list of all shortest paths between every pair of terminals in the original input graph G. These paths are used in the transformation of the MST to the SMT in the last steps of the algorithm.
- **MSTResult** stores the MST (of the metric closure) as well as the total cost of the MST.
- **SteinerResult** stores the Steiner tree, the total cost of the Steiner tree, the number of edges and vertices in the Steiner tree, as well as the cost of the MST constructed in the MST step.
- **Main** contains the driver method for executing the entire Java program. The class also contains methods to read graphs from '.gr' files and write output graphs (to '.txt') files. The class contains methods for the SP algorithm (i.e. Dijkstra's Algorithm) and the MST Algorithm (i.e. Prim's Algorithm). Other methods include getting a list of terminals, getting the shortest path between 2 nodes, creating the metric closure, and printing a graph.

3.4 Instances

100 graph instances are available for testing the Java implementation of the approximation algorithm.The source of these instances is the Parameterized Algorithms and Computational Experiments Challenge[9]. Each of the instances has the following format:

```
SECTION Graph
Nodes 5
Edges 6
E 1 2 1
E 1 4 3
E 3 2 3
E 2 4 4
E 3 5 10
E 4 5 1
END
SECTION Terminals
Terminals 2
T 2
T 4
END
EOF
```

The most important parts of the file are the edges (lines starting with E) and the terminals (lines starting with T). The edges are used to update the adjacency matrix directly. The terminals are added to the list of terminals to be used throughout the algorithm pipeline. The scanner method reads each data entry, separated by blank space, in each file.

3.5 Time complexity

The time complexity of each step in the algorithm can be estimated as follows:

- Dijkstra's Algorithm runs in $O(n^2 + m)$ time, where $n := |V|$ and $m := |E|$ in the graph G. Since priority queues are not used in our implementation, the running time cannot be improved to $O(n \log n + m)$.
- *getPath* returns the shortest path by traversing the array of previous nodes recursively. Since the shortest path may traverse through all nodes, in the worst case, this method runs in $O(n)$ time.
- Prim's Algorithm runs in $O(n^2)$ times.

[9] https://pacechallenge.wordpress.com/pace-2018/

- In the construction of the metric closure, Dijkstra's Algorithm is run k times, corresponding to k terminals. In addition, in our implementation, we also store in a list the shortest paths between each pair of terminals. There are $\frac{k(k-1)}{2}$ such pairs of terminals. Hence, the complexity of this step is $k \times O(n^2 + m) + \frac{k(k-1)}{2} \times O(n) = O(kn^2 + km + \frac{1}{2}(k^2n - kn)) = O(kn^2 + km + k^2n - kn)$.

- In constructing the output Steiner tree, first, the metric closure is created based on graph G. Then, the MST is constructed. In the transformation step, there are $\frac{k(k-1)}{2}$ iterations, in each of which the list of shortest paths is filtered for only the relevant paths. In the worst case, the filtering step runs in $O(k \times (k-1) \times ... \times 1) = O(k!)$. Merging the relevant paths, which can be as most as many as the shortest paths in the list stored in the metric closure construction step, takes at most $O(kn)$ time in the worst case when all the paths contain all V vertices.

Therefore, the time complexity of the whole approximation algorithm is

$$\begin{aligned}\text{complexity} &= O(kn^2 + km + k^2n + kn) + \mathrm{O(n^2)} + O(\tfrac{k(k-1)}{2}O(k!) + O(kn)) \\ &= O(kn^2 + km + k^2n - kn + n^2 + k^2 \times k! - k! + kn) \\ &= O(kn^2 + k^2n + k^2 \times k! - k!)\end{aligned}$$

3.6 Results

3.6.1 Results for a tight example

For the sake of brevity and conciseness, we first ran out implementation of the graph given in the example mentioned previously,(Figure 1) as it gives a big picture of the algorithm before implementing it to large-scale instances . The output Steiner tree is as follows:

STEINER TREE
Number of vertices: 7
Number of edges: 12
Tree weight: 17
Running time (ms): 39.270731

MST (in Minimum Steiner tree algorithm)
Tree weight: 22

Steiner tree graph:
0 0 0 0 0 2 0
0 0 0 0 5 2 0
0 0 0 0 0 0 4
0 0 0 0 0 0 3
0 5 0 0 0 0 0
2 2 0 0 0 0 1
0 0 4 3 0 1 0

The rows/columns of the adjacency matrix corresponds to the vertices: $v_1, v_2, ..., v_5, u_1, u_2$. As we already know, $v_1, v_2, ..., v_5$ are terminals. The output Steiner graph is the same as the solution of the approximation algorithm in the example. This indicates that our implementation indeed returns a correct solution. In addition, the MST weight is higher than the SMT weight, which makes sense as the SMT may make use of nonterminals to reduce the tree cost, thus potentially having a lower total edge cost than the MST. The Steiner Ratio for this example is

$$\rho = \tfrac{22}{17} = 1.2941 \approx 2 - \tfrac{2}{k} = 2 - \tfrac{2}{5} = 1.6$$

3.6.2 Results for given instances

100 instances from PACE are all large-scale instances with varying file size, from the smallest of 9,500 bytes to the largest of 7,500,000 bytes. The authors were trying to implement the approximation algorithm on all of the sorted instances from smallest to largest file size, however, given the limited computer resources, only *72* instances were run completely. Among those successfully solved instances, only the first (and lighter in terms of input size) 58 ones were run on an individual computer. After this 58th instance, the next ones consume all of the computer memory and crash the computer. In an attempt to make sure that the algorithm works with any kind of instance, from the 59th instance onward all the instances were run on a remote secure shell server.

Figure 4 shows the SMT results of 72 of the given instances. The average Steiner ratio is 1.1615, which is quite far from the desired 2 ratio. Nonetheless, all our output SMTs have lower (or equal) total edge costs than those of their corresponding MSTs, which is consistent with our understanding of the algorithm. The Steiner ratios are displayed in Figure 6.

As shown in 7, the total cost of a MST is always larger than or equal to that of a SMT. This consolidates the fact that the MST of the metric closure of the original graph is an approximation to the SMT.

Figure 5 shows the running times of the algorithm on the 72 instances. In general, the running time does increase as the size of graph G increases, albeit there are several outliers. These outliers could be due to the input graph having more edges.

4 Conclusion

To conclude the paper, the algorithm has proven to be feasible for any kind of instance with acceptable time complexity. Moreover, a noteworthy average of approximation ratio of *1.16* magnifies the excel of the approximation without having used all the computer resources up for the exact solution. Nonetheless, an open question concerning the SMT problem studied here is the assumption of metric space containing the vertices. As stated previously, we have assumed that the edge costs satisfy the triangle inequality, but the given instances do not explicitly follow the aforementioned property. In fact, we just assumed it. Thus, it raises some concerns about the correctness of the algorithm, as it might give incorrect solution for an arbitrarily generalized graph.

5 Appendix

5.1 Dijkstra's Algorithm

$l(s) = 0, S = \emptyset$;
For all $v \in V \setminus \{s\} : l(v) = \infty$;
while $S \neq V$ **do**
  $u = \arg\min\{l(u) : u \in V \setminus S\}$;
  $S = S \cup \{u\}$;
  For all $v : (u, v) \in A$ do
  if $l(v) > l(u) + c_{uv}$ **then**
    $l(v) = l(u) + c_{uv}$;
  end
end

Algorithm 3: Dijkstra's Algorithm

5.2 Prim's Algorithm

$T = \emptyset$, $W = \{v\}$ for some $v \in V$;
while $W \neq V$ **do**
  find $e = \{v, w\} \in \delta(W)$ s.t. $c_e \leq c_f \forall f \in \delta(W)$;
  $T = T \cup \{e\}$;
  $W = W \cup \{v, w\}$
end

Algorithm 4: Prim's Algorithm

id	V	SMTWeight	MSTWeight	rho	t	V_G	E_G
1	341	2324	2324	1	590.579369	6405	10454
3	400	19397	19838	1.022735475	1703.222116	8007	14743
5	471	23681	24173	1.02077615	2024.37983	8013	14749
9	399	16370975	22493251	1.373971373	513.169665	3803	6213
11	729	34868	35340	1.013536767	38986.522	19100	35653
13	91	9090	9898	1.088888889	50.31834	550	5013
15	92	181	197	1.08839779	105.531202	1000	13500
17	434	18511	19367	1.046242775	653.622438	3716	6750
21	113	112	121	1.080357143	158.775767	1331	19965
23	113	223	241	1.080717489	164.316854	1331	19965
25	124	123	126	1.024390244	73.296981	512	2304
27	315	10000803	12600807	1.259979524	210.367414	1822	3610
29	231	8900684	13200684	1.483108939	148.588675	987	1923
31	137	136	147	1.080882353	284.44415	1728	28512
33	137	271	293	1.081180812	289.491518	1728	28512
35	317	11900829	15000835	1.260486559	281.377638	1761	3370
37	368	12300809	15400811	1.252016107	465.692097	2346	4656
39	109	26133	26712	1.022155895	181.795854	320	640
41	85	23831	24021	1.007972809	76.489732	320	1845
43	337	12600672	15800676	1.253955027	169.279757	1491	2831
45	80	23061	23061	1	67.207742	320	10208
47	80	22756	22756	1	291.904935	320	51040
49	381	13900715	16400715	1.179846864	749.379165	3168	6220
51	252	11000550	16800564	1.527247638	116.065021	528	1017
53	254	149000536	174000536	1.167784631	122.036691	743	1409
55	305	140000594	180000596	1.285713088	194.226061	1343	2594
57	395	153000909	186000910	1.215685	410.702373	1976	3836
59	409	146000921	190000927	1.301368003	668.618603	2518	4985
61	387	156000813	194000814	1.243588481	560.190848	2262	4545
63	224	10964	11039	1.006840569	7902.436461	9469	22743
65	589	4580	4726	1.031877729	9727.273783	10393	18043
67	1121	31209414	36204691	1.160056738	1147.769332	3224	5285
69	212	47661	49675	1.042256772	151.74284	1024	5120
71	230	52759	55643	1.054663659	214.574659	640	1280
73	170	48167	48167	1	119.528857	640	4135
75	160	46052	46052	1	693.2177	640	40896
83	372	37553	40280	1.072617367	883.082865	2200	7997
85	357	35962	40204	1.117957844	379.039133	1200	10002
87	333	127234	127234	1	9706.736546	7527	18170
89	1817	13216	13647	1.032611985	330395.778	33901	62816
91	360	798	804	1.007518797	1127.844433	2187	15308
93	398	89180	90188	1.011302983	1084.090599	2048	11263
95	386	385	510	1.324675325	235.142736	512	2304
97	422	421	598	1.420427553	2786.269245	3300	18073
99	3180	86737080	106118506	1.223450294	22738.66724	8755	14449
101	3106	108482846	123938563	1.14247153	27193.13596	9287	14975
105	727	726	810	1.115702479	804.691189	783	2262
107	4236	107429973	142582814	1.327216325	86257.16259	14685	23466
109	4379	106995619	139357358	1.302458543	69424.1736	12715	20632
111	4506	120686224	149044941	1.234978907	72576.31261	12355	19962
113	771	770	1022	1.327272727	911.100174	1024	5120
115	4427	136292914	162919664	1.195364155	44990.93005	9574	16208
119	1021	1020	1102	1.080392157	1787.058417	1081	3174
121	2856	283747120	306135646	1.078903095	19434.29012	5536	8552
123	6210	148457938	172412109	1.161353251	508168.5792	21808	34921
125	6008	177758995	208224913	1.171388897	466923.3048	20547	33230
127	5634	167899553	197983827	1.179180191	102996.7558	13316	22033
129	5449	179085248	202854946	1.132728398	166076.9548	15122	24371
131	6189	187312896	210345652	1.12296407	135491.4169	13294	21948
133	6538	203226537	230316304	1.133298374	212978.4416	15714	25567
135	8796	252231054	300219554	1.190256113	981740.8854	25081	40739
139	8281	210124799	249852322	1.189066322	1706763.817	29905	47457
141	9641	298641682	360451449	1.206969659	1856481.908	30857	49591
143	2042	242074563	258069148	1.066072969	22638.47451	2676	3894
145	2115	246126746	261933377	1.064221509	22422.13776	2865	4267
149	1534	1533	2046	1.334637965	6267.131457	2048	11264
153	12195	328571933	393693087	1.198194512	2931203.176	33188	53680
157	12377	463728456	533981420	1.151495909	2321751.108	26840	43661
159	12966	417848200	500697735	1.198276635	2971563.313	30585	50454
167	3048	3051	4094	1.341855129	53842.78149	4096	24576
193	11055	197707	209979	1.062071651	3943771.801	17127	27352
199	18243	18242	35588	1.950882579	1.74E+07	18245	36038

Figure 4: Results of the approximation algorithm implementation on the given instances. The columns are: ID of input graph G, $|V_S MT|$, $w(SMT)$, $w(MST)$,ρ, time complexity (in milliseconds), $|V_G|$, and $|E_G|$

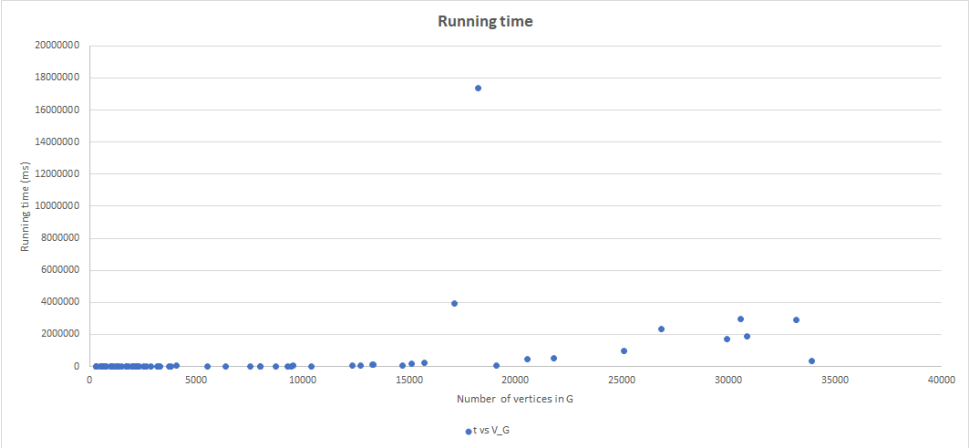

Figure 5: Time complexity of 72 instances.

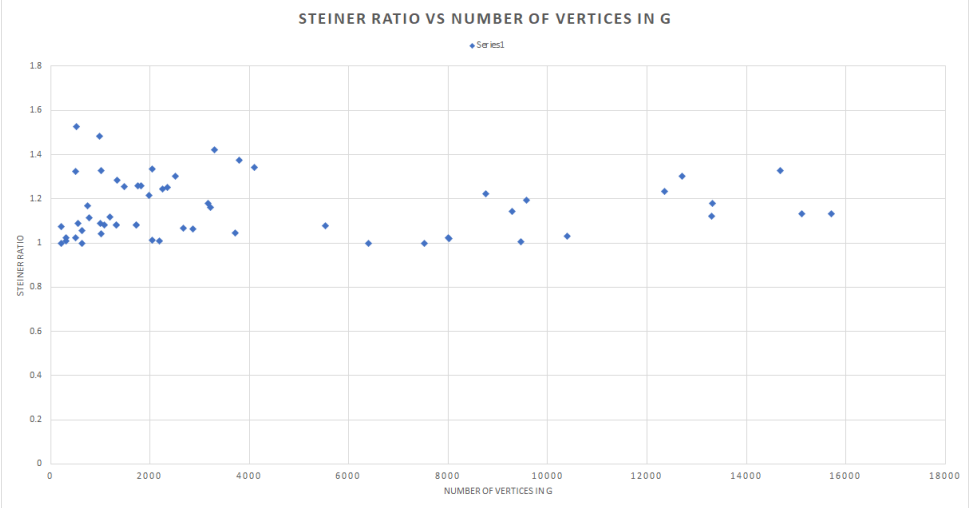

Figure 6: Steiner ratios across 72 instances. The values on the horizontal axis correspond to the original vertex counts of input graphs G.

.

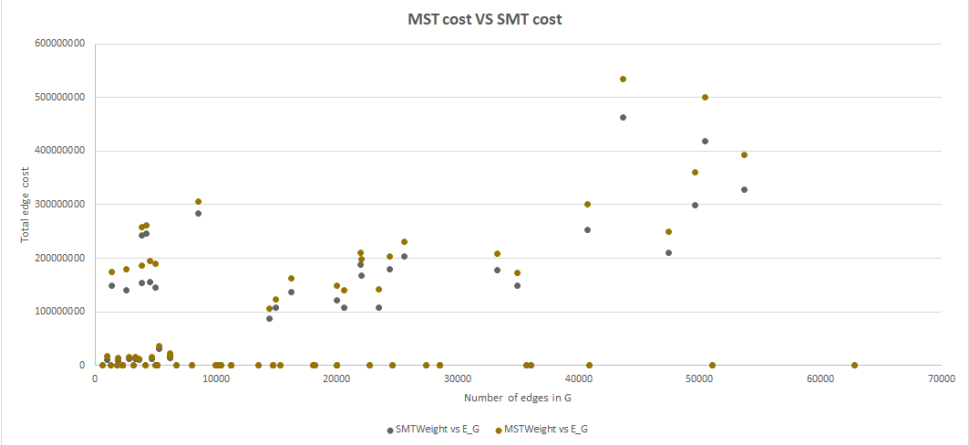

Figure 7: MST costs and SMT costs

References

[1] A. B. Kahng and G. Robins, *On optimal interconnections for VLSI* (Springer Science and Business Media, City Address, (Vol. 301)).

[2] A. E. Caldwell, A. B. Kahng, S. Mantik, I. L. Markov, and A. Zelikovsky, On wirelength estimations for row-based placement, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **18**, 1265 (1999).

[3] F. K. Hwang and D. S. Richards, Steiner tree problems, *Networks* **22**, 55 (1992).

[4] B. Korte, H. J. Prömel, A. Steger, et al., Steiner trees in vlsi-layout, *Paths, Flows, and VLSI-layout* **9**, 185 (1990).

[5] M. Garrey and D. Johnson, The rectilinear steiner problem is np-complete, *SIAM Journal of Applied Math* **13**, 1351 (1977).

[6] R. M. Karp et al., Complexity of computer computations, *Reducibility among combinatorial problems* **23**, 85 (1972).

[7] A. O. Ivanov and A. A. Tuzhilin, *Minimal NetworksThe Steiner Problem and Its Generalizations* (CRC press, 1994).

[8] F. Hwang, An o (n log n) algorithm for suboptimal rectilinear steiner trees, *IEEE Transactions on Circuits and Systems* **26**, 75 (1979).

[9] D. Fernández-Baca and J. Lagergren, A polynomial-time algorithm for near-perfect phylogeny, *SIAM Journal on Computing* **32**, 1115 (2003).

[10] A. Segev, The node-weighted steiner tree problem, *Networks* **17**, 117 (1987).

[11] C. Chiang, M. Sarrafzadeh, and C. Wong, An algorithm for exact rectilinear steiner trees for switch-box with obstacles, *[Proceedings] 1992 IEEE International Symposium on Circuits and Systems* **39**, 446455 (1992).

[12] K. Mehlhorn, A faster approximation algorithm for the steiner problem in graphs, *Information Processing Letters* **27**, 125128 (1988).

[13] M. W. Bern, Two probabilistic results on rectilinear steiner trees, *Proceedings of the eighteenth annual ACM symposium on Theory of computing - STOC 86* (1986).

[14] S. E. Dreyfus and R. A. Wagner, The steiner problem in graphs, *Networks* **1**, 195 (????), arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/net.3230010302.

[15] R. G. Downey and M. R. Fellows, *Parameterized Complexity* (Springer Publishing Company, Incorporated, 2012), ISBN 1461267986, 9781461267980.

[16] A. E. Feldmann and D. Marx, The complexity landscape of fixed-parameter directed steiner network problems, *CoRR* **abs/1707.06808** (2017), arXiv:1707.06808.

[17] M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness* (W. H. Freeman & Co., New York, NY, USA, 1990), ISBN 0716710455.

[18] R. W. Floyd, Algorithm 97: Shortest path, *Commun. ACM* **5**, 345 (1962), ISSN 0001-0782.

[19] B. Y. Wu and K.-M. Chao, *Spanning trees and optimization problems* (CRC Press, 2004).