Solving the Steiner tree problem by an approximation algorithm

E.Bahrami Rad
e-mail: s6embahr@uni-bonn.de

Contents

1 Introduction **2**

2 Steiner tree problem **2**

3 The approximation algorithm **2**

4 Implementation **3**

5 Results **4**

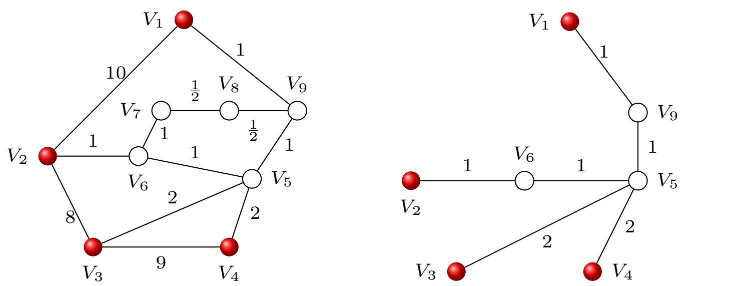
6 Conclusions **5**

1 Introduction

An approximation algorithm for finding a Steiner tree of a connected, undirected distance graph was implemented. This algoritm has approximation ratio of $2-\frac{2}{l}$, which means it gives us a Steiner tree with total distance of all edges at most $2-\frac{2}{l}$ times that of the optimal tree, where l is the number of leaves in the optimal tree. The worst case time complexity of this algorithm is $\mathcal{O}(|V|\log|V|+|E|)$, where E is the set of edges and V is the set of vertices.

2 Steiner tree problem

Given an undirected distance graph $G=(V,E,d)$ and a set S, where E is the set of edges in G, V is the set of vertices in G, d is a distance function which maps E into the set of nonnegative numbers and $S \in V$ is a subset of V. Let Q be any subset of vertices in a connected subgraph G' of G. We shall say that G' spans Q. A spanning tree of G is tree subgraph of G that spans V. The minimal spanning tree of G is a spanning tree of G such that the total distance of on its edges is minimal among all spanning trees. The Steiner tree for a given G and S is the tree that spans S, and the minimal Steiner tree is the one among all Steiner trees for G and S that has minimal total distance. The problem for finding a minimal Steiner tree for any given G and S has been proved to be NP-Complete[2].



(a) Graph G (b) Minimal Steiner tree for G

Figure 1: Example of Steiner tree problem with terminal set $S=\{V_1,V_2,V_3,V_4\}$

3 The approximation algorithm

The algorithm is a modified version of Kou, Markowsky and Berman[3], which gives a better running time and also because of reducing the problem to shortest path and minimum spanning tree it is simpler. Given $G=(V,E,d)$ a connceted, undirected distance graph, and the set of terminals $S \subseteq V$. Then the graph $G'_1=(S,E'_1,d'_1)$ can be cosntructed as follows.

For every vertex $s \in S$ let N(s) be the set of vertices in V which are closer to s than to any other vertex in S(in computational geometry N(s) is called the Voronoi region of vertex s). We can partition V according to N(s) by considering $\{N(s); s \in S\}$, in other words every vertex $v \in V$ belongs uniquely to one of the partitions N(s) with s as its center, where v is closer to s than any other vertex $q \in S$, $q \neq s$. Now, the graph $G'_1 = (S, E'_1, d'_1)$ is defined by

$$\begin{aligned} E'_1 = \{(s,t); s,t \in S \text{ and there is an edge} \\ (u,v) \in E \text{ with } u \in N(s), v \in N(t)\} \end{aligned} \tag{1}$$

and

$$\begin{aligned} d'_1(s,t) = min\{d_1(s,u) + d(u,v) + d_1(v,t); \\ (u,v) \in E, u \in N(s), v \in N(t)\}. \end{aligned} \tag{2}$$

In equation 2 $d_1(v_i, v_j)$ is equal to the distance of shortest path from v_i to v_j for $v_i, v_j \in V$ in graph G.

$d'_1(s,t)$ in equation 2 is the length of shortest path in G restricted to $N(s) \bigcup N(t)$. And the edge $(u', v') \in E'_1$ corresponds to the bridge edge $(u, v) \in E$ which acts like a bridge that goes from region N(s) to region N(t). Now the algorithm is as follow :

1. *Construct the graph* $G'_1 = (S, E'_1, d'_1)$
2. *Find a minimum spanning tree* G_2 *of* G'_1.
3. *Construct a subgraph* G_3 *of G by replacing each edge in* G_2 *by its corresponding shortest path in G.*
4. *Find a minimum spanning tree* G_4 *of* G_3.
5. *Construct a Steiner tree* G_5 *from* G_4 *by deleting edges in* G_4*, if necessary, so that no leaves in* G_5 *from the set* $V - S$

Algorithm 1

Step 1 of Algorithm 1 dominates the computational time. This step requires the solution of a single shortest path problem which takes $\mathcal{O}(|V| \log |V| + |E|)$ by using Dijkstra algorithm and Fibonacci heaps[4].

4 Implementation

Step 1 of Algorithm 1 requires the voronoi region $\{N(s); s \in S\}$ of each vertex $v \in V$. The partition $\{N(s); s \in S\}$ can be computed by adjoining an auxiliary vertex S_0 and edges $(s_0, s), s \in S$, with length 0 to G and then performing a single source shortest path with source s_0. This step was implemented in the program by using a min priority queue based on a priority heap.The priority queue provides $\mathcal{O}(log(n))$ time for enqueing and dequeing. For the Dijkstra algorithm the priority queue stores at most $|V|$ number of item, thus each of enqueing and dequeing need $\mathcal{O}(log(|V|))$ and totally there are $|E|$ number of such operations, which gives the running time of $\mathcal{O}(|E| \log |V|)$ for step 1. This step also needs $\mathcal{O}(|E|)$ for finding G_1' edges. For finding G_1''s edges the program goes through all edges $(u, v) \in E$ and generates the triples $(s(u), s(v), d_1(s(u), u) + d(u, v) + d_1(v, s(v))$ in which $s(v) \in S$ is the center of region that v belongs to.Then the program finds the minimum distance for edge $(s(u), s(v)$ over all distances $d_1(s(u), u) + d(u, v) + d_1(v, s(v))$.

Step 2 computes the minimum spanning tree G_2 of G_1'. The graph G_1' has $\mathcal{O}(|E|)$ edges and hence this step can be carried out in $\mathcal{O}(|E|log(|S|))$ by using Kruskal algorithm and Union Find data structure[4].

In step 3 the edges of G_2 have to be replaced by its corresponding edges from the shortest path computed in step 1.Consider the edge (u, v) of G_2, this edge corresponds to the shortest path between u and v in the graph G, here all the edges of G from this shortest path are plugged back to G_2. In step 1 the predecessor of each vertex has been computed and stored in an array, hence by means of this array the shortest path can be constructed.
Step 4 is the same is step 2 in which the minimum spanning tree of the graph from step 3 was computed. A

5 Results

Several tests has been performed to evaluate the performance of this aproximation algorithm. The code is implemented in Java 1.8 and the dataset from 11th DIMACS Implementation Challenge[5] is used. All runs are performed on a machine with score 181.433863(the score is calculated according to DIMACS Implementaion Challenge benchmark code to compare the results from different machines).

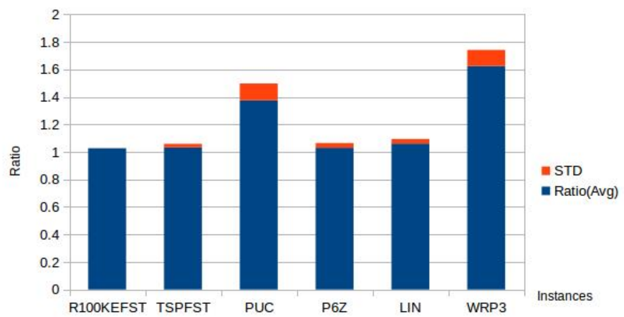
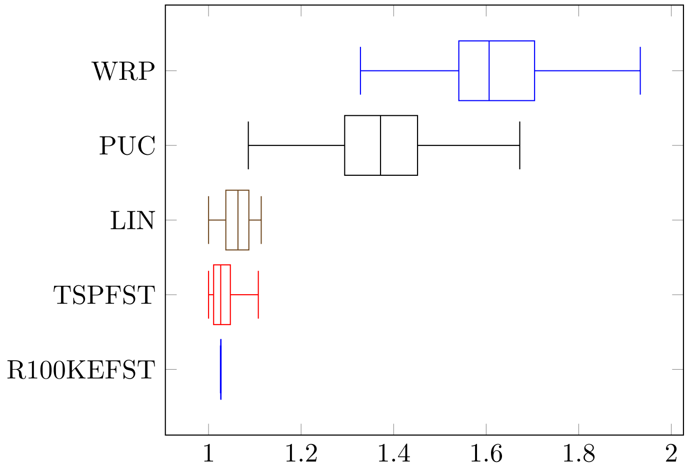


Figure 2: Comparing approximation ratio over instances of Steiner tree problem

The ratio in Figure 2 is the average(arithmetic mean) of several instances for each kind. R100KEFST, is the instance with fractional weight, Euclidean distance. TSPFST, PUC, P6Z and LIN are the instances of *SteinLib*[6] testsets.



6 Conclusions

We have presented an implementation of an approximation algorithm for Steiner tree problem. The algorithm has approximation ratio of $2 - \frac{2}{l}$,where l is the minimum number of leaves in any Steiner tree for the given graph. The algorithm running time is dominated by a single shortest path problem which is computed in our code using Dijkstra algorithm and priority queue which yields the running time of $\mathcal{O}(|E|log|V|)$.

References

[1] Mehlhorn, K. 1987, A faster approximation algorithm for the Steiner problem in graphs, Information Processing Letters 27, 125-128

[2] Karp, R.M. 1972, Reducibility among combinatorial problems, *In R. E. Miller and J. W. Thatcher (editors). Complexity of Computer Computations.* New York: Plenum. pp. 85-103.

[3] L. Kou, G. Markowsky, and L. Berman A Fast Algorithm for Steiner Trees *Acta Informatica*, 15,141-145(1981)

[4] M.L. Fredman and R.E. Tarjan, Fibonacci Heaps and Their Uses in Improved Network Optimization Algorithms(IEEE, 1984) 338-246

[5] Sariel Har-Peled, Greedy Algorithms for Minimum Spanning Trees, CS 473: Fundamental Algorithms, Fall 2011, `https://courses.engr.illinois.edu/cs473/fa2011/lec/12_notes.pdf`

[6] 11th DIMACS Implementation Challenge in Collaboration with ICERM: Steiner Tree Problems, `http://dimacs11.zib.de/downloads.html`

[7] SteinLib Testdata Library, SteinLib is a collection of Steiner tree problems in graphs and variants. `http://steinlib.zib.de/steinlib.php`