CS4234: Optimisation Algorithms Lecture 3

STEINER-TREE (3 variants)

V1.0: Seth Gilbert, V1.1: Steven Halim *August 23, 2016*

Abstract

Today we consider a new network construction problem where we are given a set of vertices in a graph to connect. A Steiner Tree is a subgraph that connects a set of required terminals, and we want to find the Steiner Tree with the minimum cost, i.e. the solution to STEINER-TREE problem. Today, we discuss three variants of this problem: the EUCLIDEAN-STEINER-TREE, the METRIC-STEINER-TREE, and the GENERAL-STEINER-TREE Problem, and show how to approximate each of these.

1 STEINER-TREE

In this section, we will define the STEINER-TREE[1] problem. We begin with a well-known problem, i.e., that of a MIN-SPANNING-TREE, and then generalize from there.

1.1 MIN-SPANNING-TREE

Imagine you were given a map containing a set of cities, and were asked to develop a plan for connecting these cities with roads. Building a road costs 1 000 000 SGD per kilometer, and you want to minimize the length of the highways. Perhaps the map looks like this (drawn on a 2D plane, units: kilometers):

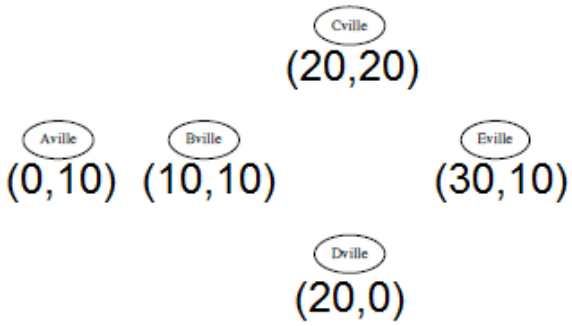


Figure 1: How do you connect the cities with a road network as cheaply as possible?

This is a standard network design question, and the solution is typically to find the minimum cost spanning tree. Recall that the problem of finding a MIN-SPANNING-TREE[2] is defined as follows:

Definition 1 *Given a graph $G = (V, E)$ and edge weights $w : E \to \mathbb{R}$, find a subset of the edges $E' \subseteq E$ such that: (i) the subgraph (V, E') is a spanning tree, and (ii) the sum of edge weights $\sum_{e \in E'} w(e)$ is minimized.*

We can then solve the road network problem described above using the following general approach:

[1] This problem is named after a Swiss mathematician named Jakob Steiner (1796-1863).
[2] Recall CS1231/CS2010/CS2020!

- For every pair of location (u, v) calculate the distance $d(u, v)$. This part is $O(V^2)$.
- Build a graph $G = (V, E)$ where V is the set of locations, and for every pair of vertices $u, v \in V$, define edge $e = (u, v)$ with weight $d(u, v)$. This results in a complete graph K_V.
- Find a minimum spanning tree of G using either Kruskal's Algorithm[3] or Prim's Algorithm[4]. Please review `http://visualgo.net/mst` for more details about these two classic MST algorithms that run in $O(E \log V)$, or $O(V^2 \log V)$ in this case as we are working with a complete graph K_V.
- Return the minimum spanning tree.

In this case, you will get a road network that looks something like on the left side of Figure 2.

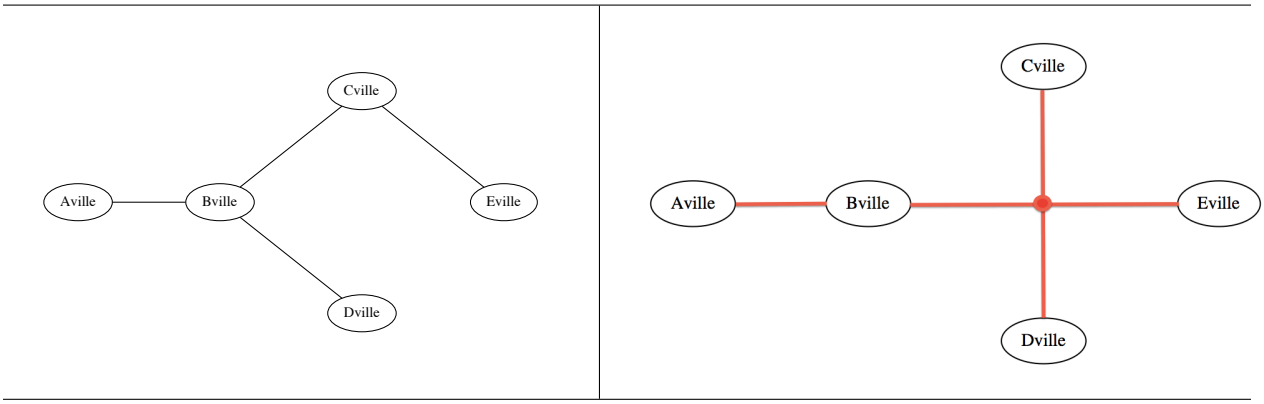


Figure 2: On the left is the graph that results from finding a minimum spanning tree with cost ≈ 52.42 M SGD. On the right is an alternate method of connecting the cities with a road network, by creating a new intersection in the middle with cost 50 M SGD.

1.2 EUCLIDEAN-STEINER-TREE

Is the minimum spanning tree solution, however, really the best solution to the problem? What about the road network on the right side of Figure 2? Notice that this road network is not actually connected in the original graph! The two edges $(B(ville), E(ville))$ and $(C(ville), D(ville))$ do not share an endpoint. Instead, they *intersect* at some new point in the middle of nowhere. Thus the minimum spanning tree approach *will never find* this solution[5].

The goal, then, of the EUCLIDEAN-STEINER-TREE problem is to find the best way to connect the cities when you are allowed to add new vertices to the graph in Euclidean space (e.g., the new intersection above). Formally, we define it as follows:

Definition 2 *Assume that you are given a set R of n distinct points in the Euclidean (2-dimensional) plane. Find a set of points S and a spanning tree $T = (R \cup S, E)$ such that that the weight of the tree is minimized. The **weight** of the tree is defined as:*

$$\sum_{(u,v)\in E} |u - v| ,$$

*where $|u - v|$ refers to the Euclidean distance from u to v. The resulting tree is called a **Euclidean Steiner Tree** and the points in S are called **Steiner** points.*

[3]Recall, Kruskal's Algorithm iterates through the edges from lightest to heaviest, adding an edge if it does not create a cycle.

[4]Recall, Prim's Algorithm grows a spanning tree from a single source vertex, repeatedly takes lightest neighboring edge that does not create a cycle until the tree spans the entire graph.

[5]That is, if we stop at CS2010/CS2020 level, we will not know about the existence of these better solution(s).

As you can see above, adding new points to the graph can results in a spanning tree of lower cost than just the standard minimum spanning tree result! The goal of the EUCLIDEAN-STEINER-TREE problem is to determine how much we can reduce the cost.

Unlike the MIN-SPANNING-TREE problem, finding the minimum solution for EUCLIDEAN-STEINER-TREE problem is NP-hard [1]. Unfortunately we do not have an easy-to-describe-yet-reasonably-good algorithm for this variant, so we will not discuss this variant in depth. We do, however, know some facts about the structure of any optimal Euclidean Steiner Tree:

- Each Steiner point in an optimal solution has degree 3.
- The three lines entering a Steiner point form 120 degree angles, in an optimal solution.
- An optimal solution has at most $n - 2$ Steiner points.

Using these known facts, we can revisit Figure 1 and Figure 2 above to come up with an even better solution, shown in Figure 3 below.

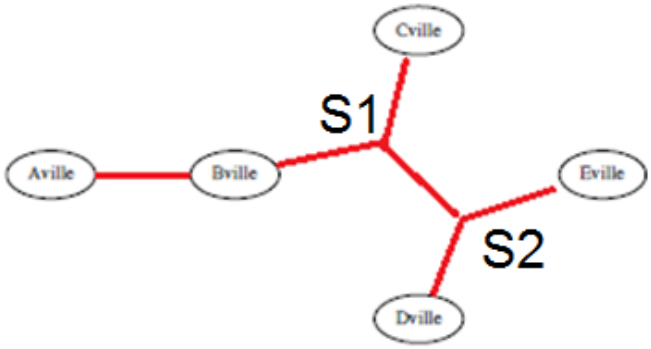


Figure 3: An even better solution with cost ≈ 48.61 M SGD

1.3 METRIC-STEINER-TREE

It is natural, at this point, to generalize beyond the Euclidean plane. Assume you are given n points, as before. In addition you are given a distance function $d : V \times V \to \mathbb{R}$ which gives the pairwise distance between any two vertices (u, v). If the points are in the Euclidean plane, we can simply define $d(u, v)$ to be the Euclidean distance between u and v. However, the distance function d can be *any* metric function, e.g. Manhattan distance[6]. Recall the definition of a metric:

Definition 3 *We say that function $d : V \times V \to \mathbb{R}$ is a **metric** if it satisfies the following properties:*

- Non-negativity*: For all $u, v \in V$, $d(u, v) \geq 0$.*
- Identity*: For all $u \in V$, $d(u, u) = 0$.*
- Symmetric*: For all $u, v \in V$, $d(u, v) = d(v, u)$.*
- Triangle inequality*: For all $u, v, w \in V$, $d(u, v) + d(v, w) \geq d(u, w)$.*

[6]Or taxicab distance. Manhattan distance of two points (a, b) and (c, d) is formally defined as $| a - c | + | b - d |$.

(Technically, this is often referred to as a pseudometric, since we allow distances $d(u, v)$ for $u \neq v$ to equal 0.) The key aspect of the distance function d is that it must satisfy the triangle inequality.

If we want to think of the input as a graph, we can define $G = (V, E)$ where V is the set of points, and E is the set of all $\binom{n}{2}$ pairs of edges, where the weight of edge (u, v) is equal to $d(u, v)$.

As in the Euclidean case, we can readily find a minimum spanning tree of G. However, the STEINER-TREE problem is to find if there is any better network, if we are allowed to add additional points. Unlike in the Euclidean case, however, it is not immediately clear which points can be added. Therefore, we are also given a set S of possible Steiner points to add. The goal is to choose some subset of S to minimize the cost of the spanning tree. The Metric Steiner Tree problem is defined more precisely as follows:

Definition 4 *Assume we are given:*

- *A set of **required** vertices R,*
- *A set of **Steiner** vertices S,*
- *A distance function $d : (R \cup S) \times (R \cup S) \to \mathbb{R}$ that is a distance metric on the points in R and S.*

The **METRIC-STEINER-TREE** *problem is to find a subset $S' \subset S$ of the Steiner vertices and a spanning tree $T = (R \cup S', E)$ of minimum weight. The **weight** of the tree $T = (R \cup S', E)$ is defined to be:*

$$\sum_{(u,v)\in E} d(u, v) \ .$$

1.4 GENERAL-STEINER-TREE

At this point, we can generalize even further to the case where d is not a distance metric. Instead, assume that we are simply given an arbitrary graph with edge weights, where some of the vertices are required vertices and some of the vertices are Steiner vertices.

Definition 5 *Assume we are given:*

- *a graph $G = (V, E)$,*
- *edge weights $w : E \to \mathbb{R}$,*
- *a set of **required** vertices $R \subseteq V$,*
- *a set of **Steiner** vertices $S \subseteq V$.*

Assume that $V = R \cup S$. The **GENERAL-STEINER-TREE** *problem is to find a subset $S' \subset S$ of the Steiner vertices and a spanning tree $T = (R \cup S', E)$ of minimum weight. The **weight** of the tree $T = (R \cup S', E)$ is defined to be:*

$$\sum_{(u,v)\in E} d(u, v) \ .$$

1.5 Summarizing the Three Different Variants

So far, we have defined three variants of the STEINER-TREE problem:

- *Euclidean:* The first variant assumes that we are considering points in the Euclidean plane.
- *Metric:* The second variant assumes that we have a distance metric.
- *General:* The third variant allows for an arbitrary graph.

Notice that the GENERAL-STEINER-TREE problem is clearly a generalization of the METRIC-STEINER-TREE problem. On the other hand, if the set of Steiner points/vertices is restricted to be finite (or countable)[7], then the METRIC-STEINER-TREE problem is not simply a generalization of the EUCLIDEAN-STEINER-TREE problem as the EUCLIDEAN-STEINER-TREE problem allows *any* points in the plane to be a Steiner point!

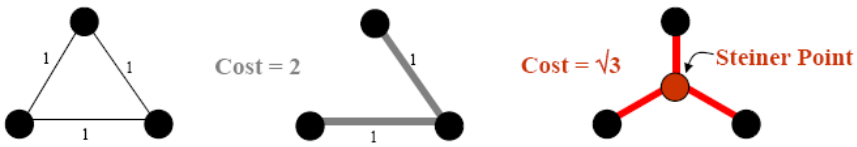
All the variants of the problem are relatively important in practice. In almost any network design problem, we are really interested in Steiner Trees, not simply Minimum Spanning Trees[8]. (One common example is VLSI layout, where we need to route wires between components on the chip.)

All three variants of the problem are NP-hard [1].

2 Steiner Tree Approximations: OK and Bad Examples

There is a simple and natural heuristic for solving the Steiner Tree problem: Just ignore all the Steiner vertices and simply find a minimum spanning tree for the required vertices R. Does this find a good approximation? We will see that for the Euclidean Steiner Tree problem and the Metric Steiner Tree problem, an MST is a good approximation of the optimal Steiner Tree. However, for the General Steiner Tree problem, an MST is *not* a good approximation.

First, consider this example of the Euclidean Steiner Tree problem:

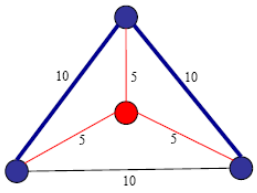


On the left picture, we are given an equilateral triangle with side-length 1. The minimum spanning tree for this triangle includes any two of the edges, and hence has length 2 (as in the middle). However, the minimum Steiner tree for the triangle adds one Steiner point: The point in the middle of the triangle (as on the right picture). With this extra Steiner point, now the total cost is $\sqrt(3)$. (You can calculate this based on the fact that the angle in the middle is 120 degrees.) Thus, a minimum spanning tree is, at best, a $2/\sqrt(3)$-approximation (or 1.15-approximation) of the optimal Euclidean Steiner tree. Is it always at least a $2/\sqrt(3)$-approximation of optimal? That remains an open conjecture. As of today, no one knows of any example that is worse than the triangle.

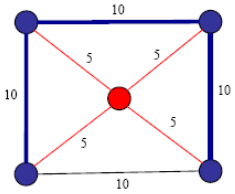
Now consider the Metric Steiner Tree problem. Obviously, a minimum spanning tree still cannot be better than a $2/\sqrt(3)$-approximation, since again we could consider the triangle with a single Steiner point in the middle. Now, however, we can construct a better example.

[7] Which actually makes the STEINER-TREE problem 'easier'...

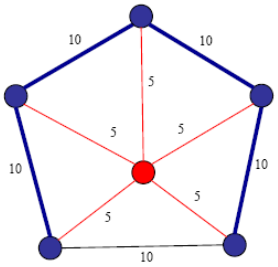
[8] So technically, stopping at CS2010/CS2020 level is not really enough to deal with these kinds of problems :O...



MST: 20
Steiner Tree: 15
Ratio: 1.33



MST: 30
Steiner Tree: 20
Ratio: 1.5



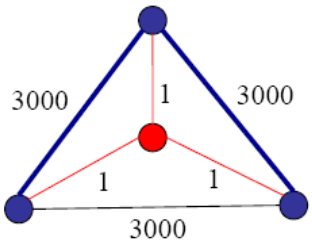
MST: 40
Steiner Tree: 25
Ratio: 1.8

Assume in this example that the distance between every pair of required vertices is 10. (In the picture, only outer edges between adjacent vertices are drawn, but the distance metric must define the distance between every pair of vertices.)

Here, we have three cases: a triangle, a square, and a pentagon (and we can generalize this further as these cases are Wheel Graphs). Each n-gon has n required vertices, where each pair of vertices is connected by edges of length 10. Each has a single Steiner vertex in the middle. There is an edge from each required vertex to the Steiner vertex in the middle, and each of these edges has length 5. Notice that this is not Euclidean: For example, the diagonal of a square would have length $10\sqrt{(2)}$, but here has only length 10. However, all the distance satisfy the triangle inequality, so the distances are metric. (Verify that this is true!)

For an n-gon where all distances are 10, a minimum spanning tree has exactly $n-1$ edges, and hence has cost $10(n-1)$. On the other hand, the minimum Steiner tree uses the Steiner vertex in the middle and has n edges, one connecting the Steiner vertex to each required vertex. The total cost of the Steiner tree is $5n$. Thus, the minimum spanning tree is no better than a $2(n-1)/n$-approximation of the optimal spanning tree. As n gets large, this approaches a 2-approximation. We will later show that a minimum spanning tree is (at least) a 2-approximation of optimal.

Finally, it should now be clear that the minimum spanning tree is not a good approximation for General Steiner Tree problem. Consider this example:



In this case, just considering the three outer vertices as required, the minimum spanning tree has cost 6 000. However, the minimum Steiner tree, including the Steiner vertex in the middle, has cost 3. Moreover, this ratio can be made as large as desired. Notice that this depends critically on the fact that the triangle inequality does not hold. That is, the distances here are not a metric.

3 METRIC-STEINER-TREE Approximation Algorithm

We are now going to show that, in a metric space, a minimum spanning tree is a good approximation of a Steiner tree. This yields a simple algorithm for finding an approximately optimal Steiner tree, in the metric case: Simply ignore the Steiner points and return the minimum spanning tree!

Theorem 6 *For a set of required vertices R, a set of Steiner vertices S, and a metric distance function d, the minimum spanning tree of (R, d) is a 2-approximation of the optimal Steiner Tree for (R, S, d).*

Proof Throughout the proof, we will use as an example the graph in Figure 4. Here we have a graph with six required vertices (i.e., the blue ones) and two Steiner vertices (i.e., the red ones). All the edges drawn have distance 1; all the other edges (not drawn) have distance 2.

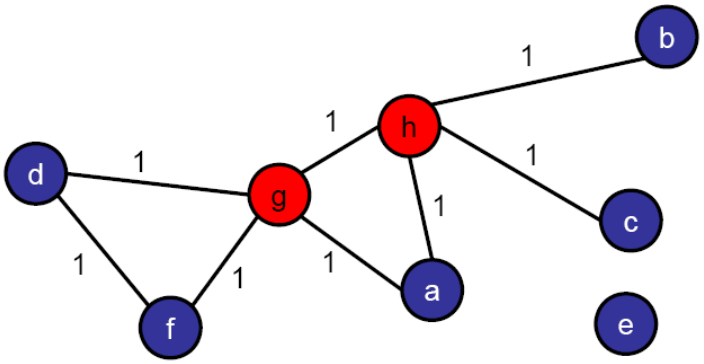


Figure 4: Example for showing that a minimum spanning tree is a 2-approximation of the optimal Steiner tree, if the distances are a metric. Assume that all edges *not* drawn have distance 2. Verify that these distances satisfy the triangle inquality.

Let $T = (V, E)$ be the optimal Steiner tree, for some $V \subseteq R \cup S$. For our example, we have drawn this optimal Steiner tree in Figure 5.

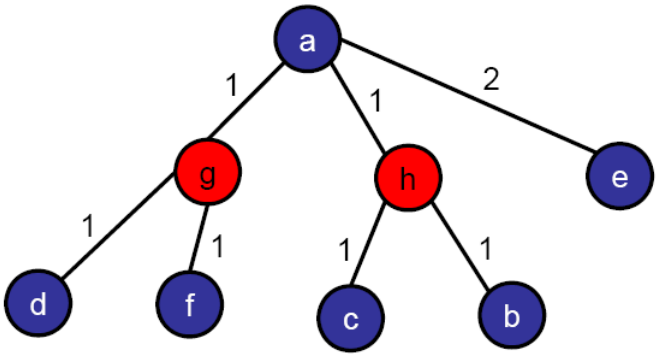


Figure 5: Here we have drawn the optimal Steiner tree T of graph shown in Figure 4.

The first step of the proof is to transform the tree T into a cycle C of at most twice the cost. We accomplish this by performing a DFS of the tree T, adding each edge to the cycle as it is traversed (both down and up). Notice that the cycle begins and ends at the root of the tree, and each edge appears in the cycle exactly twice: Once traversing down from a parent to a child, and once traversing up from a child to a parent. (Also notice that this cycle visits some vertices multiple times: A vertex with x children in the tree will appear $2(x+1)$ times ($2x$ for root vertex), twice for each of the $x+1$ (x for root vertex) adjacent edges.)

In our example, the cycle C is as follows:

$$(a,g) \to (g,d) \to (d,g) \to (g,f) \to (f,g) \to (g,a) \to (a,h) \to$$
$$(h,c) \to (c,h) \to (h,b) \to (b,h) \to (h,a) \to (a,e) \to (e,a)$$

Notice that this cycle has 14 edges, whereas the original tree has 7 edges and 8 vertices.

We have already defined the $cost(T) = \sum_{e \in E} d(e)$. Similarly, the cost of cycle C is defined as $cost(C) = \sum_{e \in C} d(e)$. Since every edge in the tree T appears exactly twice in the cycle C, we know that $cost(C) = 2 \times cost(T)$. In our example, we see that the cost of the original tree T is 8 and the cost of the cycle C is 16.

The cycle C contains both required vertices in R and Steiner vertices in S. We now want to remove all the Steiner vertices from C, without increasing the cost of the cycle C. Find any two consecutive edges in the cycle (u,v) and (v,w) where the intermediate vertex v is a Steiner vertex. (At this point, it does not matter whether u and w are required or Steiner). Replace the two edges (u,v) and (v,w) with a single edge (u,w), thus deleting the Steiner vertex v. We refer to this procedure as "short-cutting v." Notice that this replacement does not increase the cost of the cycle C, since $d(u,w) \leq d(u,v) + d(v,w)$—by the triangle inequality. *(Hint: Always pay attention to where we use the assumption; here is where the proof depends on the triangle inequality.)* Continue short-cutting Steiner vertices until all the Steiner vertices have been deleted from C.

In our example, the Steiner vertices to be removed are g and h. Thus, we update the cycle C as follows:

$$(a,d) \to (d,f) \to (f,a) \to (a,c) \to (c,b) \to (b,a) \to (a,e) \to (e,a)$$

This revised cycle now has cost[9]: $2+1+2+2+2+2+2+2 = 15$, which is no greater than the original cost of the cycle C (which was 16).

We now have a cycle C where $cost(C) \leq 2 \times cost(T)$. (Notice that the cost may have decreased during the short-cutting process, but it could not have increased.) Moreover, this cycle visits every required vertex in R at least twice (and there are no Steiner vertices in the cycle). The next step is to remove duplicates. Beginning at the root, traverse the cycle labeling each vertex: The first time a vertex is visited, mark it *new*; mark every other instances of that vertex in the cycle as *old*. (But mark the root vertex visited at the beginning and end as *new*.) As with Steiner vertices, we now short-cut all the *old* vertices: Find two edges (u,v) and (v,w) where v is an old vertex and replace those two edges in the cycle with a single edge (u,w). As before, this does not increase the cost of the cycle.

In our example, the cycle C visits the vertices in the following order:

$$\textbf{a} \to \textbf{d} \to \textbf{f} \to \underline{a} \to \textbf{c} \to \textbf{b} \to \underline{a} \to \textbf{e} \to \textbf{a}$$

The new vertices are colored with blue color, including vertex a at the beginning and the end. The old vertices are colored with red color and underlined. This leaves two instances of vertex a to be shortcut. Once we shortcut past the old vertices, we are left with the following cycle C:

$$(a,d) \to (d,f) \to (f,c) \to (c,b) \to (b,e) \to (e,a)$$

[9] When you compute the new cost, refer to the cost in original Figure 4. For example, there is an edge with weight 1 that connects vertex d and f in Figure 4.

This cycle has cost[10]: $2+1+2+2+2+2=11$, which is no greater than the original cost of the cycle (which was 16). This revised cycle C is depicted in Figure 6.

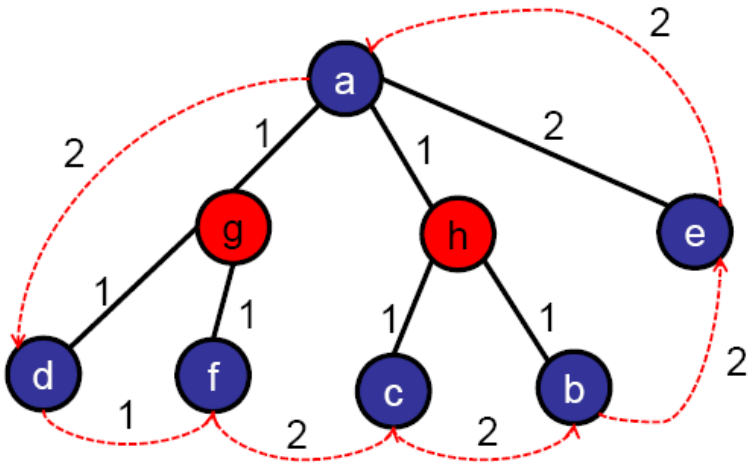


Figure 6: Here we have drawn the cycle C after the Steiner vertices have been short-cut and the repeated vertices have been deleted.

We now have a cycle C where $cost(C) \leq 2 \times cost(T)$, and each required vertex in R appears exactly once. Finally, remove any one arbitrary edge from C. (Again, this cannot increase the cost of C.) At this point, C is a path which traverses each vertex in the graph exactly once. That is, C is a spanning tree with cost at most $2 \times cost(T)$. In our example, the spanning tree C (where one arbitrary edge (e.g. the last edge) of the cycle has been deleted is):

$$(a,d) \rightarrow (d,f) \rightarrow (f,c) \rightarrow (c,b) \rightarrow (b,e)$$

This is a spanning tree with cost 9.

Let M be the minimum spanning tree of the required vertices R. Since M is the spanning tree of minimum cost, clearly $cost(M) \leq cost(C) \leq 2 \times cost(T)$. From this we conclude that M is a 2-approximation for the minimum cost Steiner tree of $R \cup S$. □

To summarize, the proof goes through the following steps:

1. Begin with the optimal Steiner Tree T.
2. Use a DFS traversal to generate a cycle of twice the cost.
3. Eliminate the Steiner vertices and repeated required vertices, without increasing the cost. (Use the assumption that d satisfies the triangle inequality.)
4. Remove on edge from the cycle, yielding a spanning tree of cost at most twice the cost of T.
5. Observe that the minimum spanning tree can have cost no greater than the constructed spanning tree, and hence no greater than twice the cost of T.

This implies that the minimum spanning tree has cost at most twice the cost of T, the optimal Steiner Tree.

[10] Again, when you compute the new cost, refer to the cost in original Figure 4. For example, there are edges with weight 2 that connects vertex f and c, vertex b and e, and vertex e and a in Figure 4 (all not drawn).

4 GENERAL-STEINER-TREE Approximation Algorithm

In general, a minimum spanning tree is *not* a good approximation for the GENERAL-STEINER-TREE problem. Here we want to show how to find a good approximation in this case. Instead of developing an algorithm from scratch, we are going to use a reduction. Part of the goal is to demonstrate how to use reductions when we are talking about approximation algorithms.

The typical process, with a reduction, is something like as follows:

- Begin with an instance of the GENERAL-STEINER-TREE problem.
- Via the reduction, construct a new instance of the METRIC-STEINER-TREE problem.
- Solve the METRIC-STEINER-TREE problem using our existing algorithm.
- Convert the solution to the METRIC-STEINER-TREE instance back to a solution for the GENERAL-STEINER-TREE problem.

The key to the analysis would typically be a lemma that says something like, "If we have an algorithm for finding an optimal solution to the METRIC-STEINER-TREE problem, then our construction/conversion process yields an optimal solution to the GENERAL-STEINER-TREE problem."

With approximation algorithms, however, you have to be a little more careful. Normally, it is sufficient to show that an optimal solution to the translated problem yields an optimal solution to the original problem. However, we are looking here at approximation algorithms. Hence we need to show that, even though we are only finding an approximate solution to the METRIC-STEINER-TREE problem, that still yields an approximate solution to the GENERAL-STEINER-TREE problem. A reduction that preserves approximation ratios is known as a ***gap-preserving*** reduction.

Assume we are given a graph $G = (V, E)$ and non-negative edge weights $w : E \to \mathbb{R}^{\geq 0}$. The vertices V are divided into required vertices R and Steiner vertices S. Our goal is to find a minimum cost Steiner Tree. There are no restrictions on the weights w (i.e., they do not necessarily satisfy the triangle inequality).

Defining a metric. In order to perform the reduction, we need to construct an instance of the METRIC-STEINER-TREE problem. In particular, we need to define a metric. The specific distance metric we are going to define is known as the ***metric completion*** of G.

Definition 7 *Given a graph $G = (V, E)$ and non-negative edge weights w, we define the **metric completion** of G to the be the distance function $d : V \times V \to \mathbb{R}$ constructed as follows: For every $u, v \in V$, define $d(u, v)$ to be the distance of the* shortest path *from u to v in G with respect to the weight function w.*

Notice that the metric completion d provides distances between every pair of vertices, not just the edges in E. Also notice that, computationally, d is relatively easy to calculate, e.g., via an All-Pairs-Shortest-Paths algorithm such as Floyd-Warshall which runs in $O(V^3)$ time[11]. Critically, d is a metric:

Lemma 8 *Given a graph $G = (V, E)$, the metric completion d is a metric with respect to V.*

Proof Since all the edge weights are non-negative, clearly $d(u, v) \geq 0$ for all $u, v \in V$. Similarly, $d(u, u) = 0$, by definition. Since the original graph is undirected, the shortest path from u to v is also the shortest path from v to u; hence $d(u, v) = d(v, u)$.

[11] Compared to the NP-hardness of the original STEINER-TREE problem, running an $O(V^3)$ algorithm for its approximation algorithm is generally viewed as OK.

The most interesting property is the triangle inequality. We need to show that for all vertices $u, v, w \in V$, the distances $d(u, v) + d(v, w) \geq d(u, w)$. Assume, for the sake of contradiction, that this inequality does not hold, i.e., $d(u, v)+d(v, w) < d(u, w)$. Let $P_{u,v}$ be the shortest path from u to v in G (with respect to the weight function w), and let $P_{v,w}$ be the shortest path from v to w (with respect to the weight function w). Consider the path $P_{u,v} + P_{v,w}$, which is a path from u to w of length $d(u, v) + d(v, w)$. This path is of length less than $d(u, w)$. But that is a contradiction, since $d(u, w)$ was defined to be the length of the shortest path from u to w.

From this, we conclude that d satisfies the triangle inequality, and hence is a metric. □

In the following, we will sometimes be calculating the cost with respect to the metric completion d, and sometimes with respect to the edge weights w. To be clear, we will use $cost_d$ to refer to the former and $cost_w$ to refer to the latter. Similarly, we will refer to graph G^g when talking about the GENERAL-STEINER-TREE problem, and G^m when talking about the METRIC-STEINER-TREE problem.

Converting from General to Metric. We can now reduce the original instance of the GENERAL-STEINER-TREE problem to an instance of the METRIC-STEINER-TREE problem. Assume we have an algorithm A that finds an α-approximate minimum cost METRIC-STEINER-TREE.

- Given a graph $G^g = (V, E^g)$ with requires vertices R, Steiner vertices S, and a non-negative edge weight function w:
- Let d be the metric completion of G^g.
- Consider the METRIC-STEINER-TREE problem: (R, S, d).
- Let $T^m = A(R, S, d)$ be the α-approximate minimum cost Metric Steiner tree.

At this point, we have converted our GENERAL-STEINER-TREE problem into a METRIC-STEINER-TREE problem, and solved it using our existing approximation algorithm.

Converting from Metric back to General. We are not yet done, however, since T^m is defined in terms of edges that may not exist in G^g, and in terms of a different set of costs. We need to convert the tree T^m back into a tree in G^g.

For every edge $e = (u, v)$ in the tree T^m, let p_e be the shortest path in G^g from u to v. Let $P = \bigcup_{e \in T} p_e$, i.e., the set of all paths that make up the tree T^m. Notice that some of these paths may overlap.

Define the cost of a path in G^g (with respect to w) to be the sum of the costs of the edge weights, i.e., $cost_w(p) = \sum_{e \in p} w(e)$. Notice that the cost of a path in G^g is with respect to edge weights, while the cost of the tree T^m is with respect to the metric completion d. (This difference is because we are converting back from the metric to the general problem.) If $e = (u, v)$ is an edge in the tree T^m, then $cost_w(p_e) = d(u, v)$.

Consider all the paths $p_e \in P$, i.e., for every edge e in the tree T^m: add every edge that appears in any path $p_e \in P$ to a set E'. Notice that the graph $G' = (V, E')$ is connected, since the original tree T^m was connected and if there was an edge (u, v) in T^m then there is a path connecting u to v in E'. Also, notice that the cost of all the edges in E' (with respect to w) is no greater than the cost of all the edges in T^m (with respect to d), since each path p_e costs the same amount as the edge e in T^m.

Finally, we need to remove any cycles from the graph (V, E') so that we have a spanning tree. Let T^g be the minimum spanning of (V, E'). The tree T^g in the graph $G^g = (V, E^g)$ with edge weights w has cost no greater than the tree T^m with edge weights d.

Analysis. To analyze this, we need to prove two key lemmas (proof omitted). Notice that you need to prove two things: You need to relate the solution in the metric version to OPT in the general version, and you also need to relate the final solution in the general version to the solution found in the metric version.

Lemma 9 *Let OPT^g be the optimal minimum cost Steiner tree for G^g. Then $cost_d(T^m) \leq \alpha \cdot cost_w(OPT^g)$.*

Lemma 10 *Let T^g be the Steiner tree calculated by converting T^m back to graph G^g. Then $cost_w(T^g) \leq cost_d(T^m)$*

Putting these lemmas together, we get our final result:

Theorem 11 *Given an α-approximation algorithm for* METRIC-STEINER-TREE *problem, we can find an α-approximation for a* GENERAL-STEINER-TREE *problem.*

Proof Assume we have a graph $G^g = (V, E^g)$ with required vertices R, Steiner vertices S, and a non-negative edge weight function w. Let T^m be an α-approximate Steiner tree for (R, S, d), where d is the metric completion of G^g. Let T^g be the spanning tree constructed above by converting the edges in T^m into paths in $G^g = (V, E^g)$ and removing cycles. We will argue that T^g is an α-approximation of the minimum cost spanning tree for G.

First, we have shown that $cost_d(T^m) \leq \alpha \cdot cost_w(OPT^g)$. Second, we have shown that $cost_w(T^g) \leq cost_d(T^m)$. Putting the two pieces together, we conclude that $cost_w(T^g) \leq \alpha \cdot cost_w(OPT^g)$, and hence T^g is an α-approximation for the minimum cost Steiner tree for G with respect to w. □

References

[1] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.

Index