Abstract

Today we consider two new problems, both related to finding the best way to connect a set of vertices in a graph. The Steiner Tree Problem is to find a subgraph that connects a set of terminals at minimal cost. The Traveling Salesman Problem is to find a tour that connects all the nodes in the graph at a minimal cost. Each of these problems has several variants, most of which we can approximate quite well. In addition, both of these problems are closely connected to finding a minimum spanning tree, and we use this to develop good approximation algorithms.

1 Three Similar Problems

In this section, we will define three different problems, along with several variants.

1.1 Minimum Spanning Trees

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 SGD1,000,000 per kilometer, and you want to minimize the length of the highways. Perhaps the map looks like this:

![Map of cities with roads](image)

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 minimum spanning tree 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:

- For every pair of location $(u, v)$ calculate the distance $d(u, v)$.
- Build a graph $G = (V, E)$ where $V$ is the set of locations, and for every pair of nodes $u, v \in V$, define edge $e = (u, v)$ with weight $d(u, v)$. This results in a complete graph.
Find a minimum spanning tree of $G$ using Kruskal’s Algorithm (or Prim’s Algorithm). (Recall, Kruskal’s Algorithm iterates through the edges from lightest to heaviest, adding an edge if it does not create a cycle.)

Return the spanning tree.

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

1.2 Steiner Trees

Is this, 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 $(B, E)$ and $(C, D)$ 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.

Euclidean Steiner Tree. The goal, then, of the Euclidean Steiner Tree is to find the best way to connect the cities, even when you are allowed to add new nodes to the graph (e.g., the new intersection above). Formally, we define it as follows:

Definition 2 Assume 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 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.

As you can see above, adding new points to the graph results in a spanning tree of lower cost! The goal of the Euclidean Steiner Tree problem is to determine how much we can reduce the cost.

Unlike the problem of finding a minimum spanning tree, finding a minimum Euclidean Steiner tree is NP-hard. 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.

As an exercise, prove these facts to be true. (Hint: given a Steiner tree that does not satisfy these properties, show how to replace the Steiner points with a version that does satisfy these requirements.)

**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 nodes $(u, v)$. 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. 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 $u, v, w \in V$, $d(u, v) + d(v, w) \geq d(u, w)$.

(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* nodes $R$,
- a set of *Steiner* nodes $S$,
- 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 points 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:

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

**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 nodes are required and some of the nodes are Steiner points.

**Definition 5** Assume we are given:
• a graph $G = (V, E)$,
• edge weights $w : E \rightarrow \mathbb{R}$,
• a set of required nodes $R \subseteq V$,
• a set of Steiner nodes $S \subseteq V$.

Assume that $V = R \cup S$. The General Steiner Tree problem is to find a subset $S' \subset S$ of the Steiner points 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:

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

**Summarizing the 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 is restricted to be finite (or countable), then the Metric Steiner Tree problem is not simply a generalization of the Euclidean Steiner Tree problem—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. (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. As an exercise, prove that the General Steiner Tree problem is NP-hard by reduction from Set Cover. (Hint: think of the underlying elements as the required nodes and the sets as Steiner nodes.)

### 1.3 Traveling Salesman Problem

The Steiner Tree problem is closely related to another famous problem: the Traveling Salesman Problem. The “TSP” problem is perhaps one of the most famous (and most studied) problems in combinatorial optimization. Instead of trying to find a spanning tree, in the traveling salesman problem, we want to find a circuit. The problem is stated as follows:

**Definition 6** The Traveling Salesman Problem is defined as follows: given a set $V$ of $n$ points and a distance function $d : V \times V \rightarrow \mathbb{R}$, find a cycle $C$ of minimum length. (The length of a cycle $C = (e_1, e_2, \ldots, e_m)$ is defined to be $\sum_{e \in C} d(e)$.)

Notice that, unlike the Steiner tree problem, there are no Steiner nodes: you have to visit every city. See Figure 3 for an example of one instance of the traveling salesman problem.

As with the Steiner tree problem, there are several variants:

• **Metric vs. General:** In the metric version of the traveling salesman problem, the distance function $d$ is a metric, i.e., it satisfies the triangle inequality. In the general version, $d$ can assign any arbitrary weight to an edge.
Figure 3: Example of the traveling salesman problem. Here, there are six locations. The problem is to find the shortest delivery circuit starting from Ikea and visiting each of the five houses, and then returning to Ikea.

- **Repeated visits vs. No-Repeats**: The goal of the TSP problem is to find a cycle that visits every node. Can the cycle contain repeated nodes (or does it have to be a simple cycle)? In the version with No-Repeats, the TSP cycle must visit each node exactly once. In the version with repeats, it is acceptable to visit each node more than once (if that results in a shorter route).

We will summarize these four variants as follows:

<table>
<thead>
<tr>
<th>Metric</th>
<th>Repeats</th>
<th>No-Repeats</th>
</tr>
</thead>
<tbody>
<tr>
<td>General</td>
<td>G-R TSP</td>
<td>G-NR TSP</td>
</tr>
<tr>
<td>Metric</td>
<td>M-R TSP</td>
<td>M-NR TSP</td>
</tr>
</tbody>
</table>

Again, all the variants of the traveling salesman problem are NP-hard. In fact, the problem is NP-hard even for planar graphs with maximum degree 3. We can prove the traveling salesman problem to be NP-hard by reduction from Hamiltonian Cycle. The Hamiltonian Cycle problem asks whether a graph has a cycle that visits every node exactly once. Clearly if we could solve the general no-repeats version of TSP (G-NR TSP), when we could solve the Hamiltonian Cycle problem. There are similar reductions for the other variants.

In fact, the general no-repeats version of TSP (G-NR TSP) is NP-hard even to approximate! If $P \neq NP$, there is no constant factor approximation algorithm. In fact, something even stronger is true. Let $r(n)$ be any polynomial time computable function. (For example, perhaps $r(n) = 2^{100n^2}$.) Let $A$ be an $r(n)$-approximation algorithm for G-NR TSP. If $A$ is a polynomial time algorithm, then $P = NP$. Clearly, this is a problem that is very hard to approximate.

Even so, we are going to see how to (easily!) approximate the other three variants. In almost all common real-world cases (e.g., where the distance function satisfies the triangle inequality, or where repeats are acceptable), there are good approximation algorithms.
2 Steiner Tree Approximation: Bad Examples

There is a simple and natural heuristic for solving the Steiner Tree problem: ignore all the Steiner nodes and simply find a minimum spanning tree for the required nodes $R$. 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, 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). 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 of the optimal Euclidean Steiner tree. Is it always at least a $2/\sqrt{3}$-approximation of optimal? That remains an open conjecture. No one knows of any example that is worse than the triangle.

Now consider the more general 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.

Here, we have three cases: a triangle, a square, and a pentagon. Each $n$-gon has $n$ required nodes, where each pair of nodes is connected by edges of length 10. (In the picture, I have only drawn the outer edges.) Each has a single Steiner node in the middle. There is an edge from each required node to the Steiner node 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. (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 node in the middle and has $n$ edges, one
connecting the Steiner node to each required node. 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 in general. Consider this example:

In this case, just considering the three outer nodes as required, the minimum spanning tree has cost 6000. However, the minimum Steiner tree, including the Steiner node 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 7** For a set of required nodes \(R\), a set of Steiner nodes \(S\), 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 nodes (i.e., the blue ones) and two Steiner nodes (i.e., the red ones). All the edges drawn have distance 1; all the other edges (undrawn) have distance 2.

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.

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 nodes multiple times: a node with \(x\) children in the tree will appear \(2(x + 1)\) times, twice for each of the \(x + 1\) adjacent edges.)

In our example, the cycle \(C\) is as follows:

\[(a, g) \rightarrow (g, d) \rightarrow (d, g) \rightarrow (g, f) \rightarrow (f, g) \rightarrow (g, a) \rightarrow (a, h) \rightarrow (h, c) \rightarrow (c, h) \rightarrow (h, b) \rightarrow (b, h) \rightarrow (h, a) \rightarrow (a, e) \rightarrow (e, a)\]

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

Figure 5: Example for showing that a minimum spanning tree is a 2-approximation of the optimal Steiner tree, if the distances are a metric. Here we have drawn the optimal Steiner tree $T$.

We have already defined the $\text{cost}(T) = \sum_{e \in E} d(e)$. Similarly, the cost of cycle $C$ is defined as $\text{cost}(C) = \sum_{e \in C} d(e)$. Since every edge in the tree $T$ appears exactly twice in the cycle $C$, we know that $\text{cost}(C) = 2 \text{cost}(T)$. 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 nodes in $R$ and Steiner nodes in $S$. We now want to remove all the Steiner nodes 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 node $v$ is a Steiner node. (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 node $v$. 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 nodes until all the Steiner nodes have been deleted from $C$.

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

$$(a, d) \rightarrow (d, f) \rightarrow (f, a) \rightarrow (a, c) \rightarrow (c, b) \rightarrow (b, a) \rightarrow (a, e) \rightarrow (e, a)$$
This revised cycle now has cost: $2 + 1 + 2 + 2 + 2 + 2 + 2 + 2 = 15$, which is no greater than the original cost of the cycle $C$ (which was 16).

We now have a cycle $C$ where $\text{cost}(C) \leq 2\text{cost}(T)$. (Notice that the cost may have decreased during the short-cutting process, but it could not have increased.) Moreover, this cycle visits every required node in $R$ at least twice (and there are no Steiner nodes in the cycle). The next step is to remove duplicates. Beginning at the root, traverse the cycle labeling each node: the first time a node is visited, mark it \textit{new}; mark every other instance of that node in the cycle as \textit{old}. (Mark the root node visited at the beginning and end as \textit{new}, also.) As with Steiner nodes, we now short-cut all the \textit{old} nodes: find two edges $(u, v)$ and $(v, w)$ where $v$ is an old node 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 nodes in the following order:

$$a \rightarrow d \rightarrow f \rightarrow a \rightarrow c \rightarrow b \rightarrow a \rightarrow e \rightarrow a$$

I have bolded the new nodes and colored them blue, including node $a$ at the beginning and the end. I have colored the old nodes red. This leaves two instances of node $a$ to be shortcut. Once we shortcut past the old nodes, we are left with the following cycle $C$:

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

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

We now have a cycle $C$ where $\text{cost}(C) \leq 2\text{cost}(T)$, and each required node 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 node in the graph exactly once. That is, $C$ is a spanning tree with cost at most $2\text{cost}(T)$. In our example, the spanning tree $C$ (where one arbitrary 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 nodes $R$. Since $M$ is the spanning tree of minimum cost, clearly $\text{cost}(M) \leq \text{cost}(C) \leq 2\text{cost}(T)$. 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 nodes and repeated required nodes, without increasing the cost. (Use the assumption that $d$ satisfies the triangle inequality.)
4. Remove an 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 spanning tree.

One question is whether this bound is tight. We know that the minimum spanning tree is no better than a $2(1 - 1/n)$ approximation, and we have proved here that it is at least a 2-approximation. On the problem set this week, you will find a matching bound.
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 $w : E \rightarrow \mathbb{R}^{\geq 0}$. The nodes $V$ are divided into required nodes $R$ and Steiner nodes $S$. 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 8** Given a graph $G = (V, E)$ and non-negative edge weights $w$, we define the **metric completion** of $G$ to be the distance function $d: V \times V \rightarrow \mathbb{R}$ 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 nodes, not just the edges in $E$. Also notice that, computationally, $d$ is relatively easy to calculate, e.g., via an All-Paris-Shortest-Paths algorithm (such as Floyd-Warshall, which runs in $O(n^3)$ time). Critically, $d$ is a metric:

**Lemma 9** 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 $d(u, v) = d(v, u)$.

The most interesting property is the triangle inequality. We need to show that for all nodes $u, v, w \in V$, 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. \hfill \qed

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 $G^9$ 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 $\alpha$-approximate minimum cost Metric Steiner tree.

- Given a graph $G^g = (V, E^g)$ with requires nodes $R$, Steiner nodes $S$, 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 $\alpha$-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. We can now relate the cost of the optimal Steiner tree for $G^g$ to the cost of $T^m$, the approximately optimal Steiner tree that we have just calculated. Define $cost_d(T^m) = \sum_{e \in T^m} d(e)$, i.e., the cost of the tree is the sum of the distances of the edges (with respect to the metric completion).

**Lemma 10** Let $OPT^g$ be the optimal minimum cost Steiner tree for $G^g$. Then $cost_d(T^m) \leq \alpha \cdot cost_w(OPT^g)$.
Proof \( \text{OPT}^g \) is the optimal Steiner tree for \( G^g = (V, E^g) \) with respect to \( R \) and \( S \) and weights \( w \). Notice that \( \text{OPT}^g \) is also a valid Steiner tree in the metric case, i.e., for \((R \cup S)\) and \( d \). (However, it may not be the optimal solution in the metric case.) Consider the cost of \( \text{OPT}^g \) in \((R \cup S)\) with respect to the metric completion \( d \), i.e., \( \text{cost}_d(\text{OPT}^g) \).

Let \( e = (u, v) \) be some edge in \( \text{OPT}^g \). Then clearly \( d(u, v) \leq w(u, v) \), since \( d(u, v) \) represents the shortest path from \( u \) to \( v \) and the edge \((u, v)\) is a path of length \( w(u, v) \). Since the total cost is simply the sum of the edge costs, we conclude that \( \text{cost}_d(\text{OPT}^g) \leq \text{cost}_w(\text{OPT}^g) \).

Let \( \text{OPT}^m \) be the optimal Steiner tree in \((R \cup S)\) with respect to \( d \). Since \( \text{OPT}^m \) is optimal, we know that \( \text{cost}_d(\text{OPT}^m) \leq \text{cost}_d(\text{OPT}^g) \leq \text{cost}_w(\text{OPT}^g) \). That is, the real optimal Steiner tree in the metric case has cost no greater than the cost of the optimal Steiner tree for the general case.

Finally, since \( T^m \) was calculated using an \( \alpha \)-approximation algorithm, we know that \( \text{cost}_d(T^m) \leq \alpha \cdot \text{cost}_d(\text{OPT}^m) \), and hence putting the inequalities together, \( \text{cost}_d(T^m) \leq \alpha \cdot \text{cost}_w(\text{OPT}^g) \). \hfill \( \square \)

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., \( \text{cost}_w(p) = \sum_{e \in p} w(e) \). 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 \( \text{cost}_w(p_e) = d(u, v) \).

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 \( d \).

We now argue that this transformation from \( T^m \) to \( T^g \) has not increased the cost too much:

Lemma 11 \( \text{cost}_w(T^g) \leq \text{cost}_d(T^m) \)

Proof This is because \( \text{cost}_w(T^g) \leq \text{cost}_w(E') \) (as \( T^g \) consists of a subset of edges from \( E' \)), and \( \text{cost}_w(E') \leq \text{cost}_d(T^m) \) (since we constructed \( E' \) by adding paths whose cost summed to the cost of \( T^m \)). \hfill \( \square \)

Putting the pieces together. Overall, this procedure yields the following theorem:

Theorem 12 Given an \( \alpha \)-approximation algorithm for finding a Metric Steiner tree, we can find an \( \alpha \)-approximation for a General Steiner tree.
Proof Assume we have a graph $G^g = (V, E^g)$ with required nodes $R$, Steiner nodes $S$, and a non-negative edge weight function $w$. Let $T^m$ be an $\alpha$-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 $\alpha$-approximation of the minimum cost spanning tree for $G$.

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