Chapter 7

Approximation Algorithms for Steiner Trees

7.1 The problem and its complexity

Let $G = (V, E)$ be an undirected graph, let each edge $ij \in E$ have a nonnegative real-valued cost, and let N be a set of terminal nodes, $N \subseteq V$. A tree T of G (that is, the node set $V(T)$ is a subset of V and the edge set $E(T)$ is a subset of E) is called a *Steiner tree* if it contains all the terminal nodes, $V(T) \supseteq N$. Nonterminal nodes, nodes in $V \backslash N$, may be present in T; nodes in $V(T) \backslash N$ are called *Steiner nodes*. The problem is to find a Steiner tree T whose cost $c(T) = \sum_{ij \in E(T)} c(ij)$ is minimum among all Steiner trees; an optimal solution T is called a *Steiner minimal tree* (SMT for short). See Figure 7.1 for an example. The problem, even with unit costs on the edges, is NP-hard, so an efficient algorithm (i.e., one that runs in time polynomial in $|V|$) for finding an optimal solution would give efficient algorithms for solving all NP-complete problems. So it is likely that any algorithm ever designed for finding Steiner minimal trees will be inefficient.

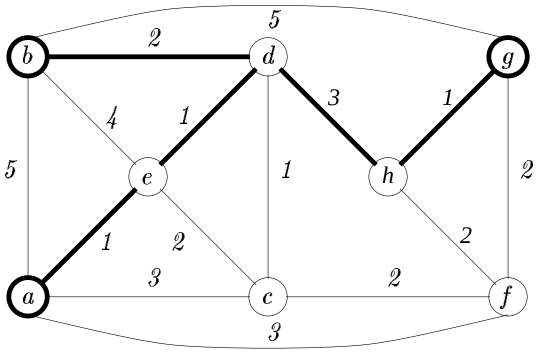

Figure 7.1: An example of the Steiner tree problem; the terminal node set is $N = \{a, b, g\}$. The terminal nodes (nodes in N) are circled in bold. A Steiner minimal tree is indicated by thick lines; it has cost 8.

Nevertheless, the problem has many practical applications such as routing in VLSI layout, the design of communication networks, accounting and billing for the use of telephone networks, the phylogeny problem in biology, etc. Moreover, it has generated interesting mathematical questions and conjectures, from the days of Fermat (1601–1665) to the present.

Two special cases of the Steiner tree problem are solvable by efficient algorithms: If all nodes are terminals, that is, if $N = V$, then we have the minimum spanning tree problem, and an optimal solution can be found using, e.g., the greedy algorithm (Kruskal's algorithm). If there are exactly two terminal nodes, say, $N = \{s, t\}$, then we have the shortest s-t path problem, and an optimal solution can be found using Dijkstra's algorithm.

Our main focus in this chapter is on heuristics for the Steiner tree problem and on deriving their performance guarantees. First we present a simple heuristic called the distance network heuristic with performance ratio 2 and an efficient variant due to Mehlhorn. Then we present Zelikovsky's recent algorithm with a better performance ratio of $\frac{11}{6}$.

7.2 Distance network heuristics for Steiner trees

7.2.1 Introduction

An instance of the Steiner tree problem consists of a graph $G = (V, E)$, a nonnegative real-valued cost $c : E \to \Re_+$ on the edges, and a set N of terminal nodes. Let $T_G(N)$ and $c(T_G(N))$ denote a Steiner minimal tree and its cost, respectively. For any two nodes v and w of G, let $d_G(v, w)$ denote the cost of a shortest path between v and w in G, i.e., $d_G(v, w) = \min \ \sum_{i=1}^{\ell} c(v_{i-1}v_i)$, where the minimization is over all paths $P = v_0v_1 \ldots v_\ell$ in G from $v = v_0$ to $w = v_\ell$. For a node v and a set of nodes S, $d_G(S, v)$ denotes $\min\limits_{w \in S} d_G(w, v)$, i.e., the cost of a shortest path from any node in S to v. Given any subset V' of the nodes of G, the distance network $D_G(V')$ is defined to be a graph whose node set is V' and such that for every pair of nodes $v \in V'$, $w \in V'$, the edge vw is present; the cost of an edge vw, denoted $c_D(vw)$, is defined to be $d_G(v, w)$. In other words, $D_G(V')$ is the complete graph on the node set V' together with the edge costs $c_D(vw) = d_G(v, w)$ (for all edges vw). For every edge vw in G, observe that $d_G(v, w) \leq c(vw)$. This gives us the following.

Fact: For every edge vw of the distance network $D_G(V')$, if vw is an edge of G, then $c_D(vw) \leq c(vw)$.

A real-valued nonnegative function on pairs of nodes $\ell : V \times V \to \Re_+$ is said to satisfy the triangle inequality if for every three nodes v, w and x, $\ell(v, w) \leq \ell(v, x) + \ell(x, w)$. Clearly, the shortest paths costs $d : V \times V \to \Re_+$ satisfy the triangle inequality, and this gives the following.

Fact: In the distance network $D_G(V')$, the edge costs c_D satisfy the triangle inequality.

The next few results consider the distance network $D = D_G(V)$, i.e., all nodes of G are present in D.

Proposition 7.1 *Let N be the set of terminal nodes, let $T_G(N)$ be a Steiner minimal tree of G with terminal node set N, and let $T_D(N)$ be a Steiner minimal tree of the distance network $D = D_G(V)$ with terminal node set N. Then the cost of $T_D(N)$ equals the cost of $T_G(N)$.*

Proof: We show first that $c_D(T_D(N)) \leq c(T_G(N))$, and then that $c(T_G(N)) \leq c_D(T_D(N))$. For every edge vw of $T_G(N)$, note that it is present in D and that its cost in D, $c_D(vw)$, is at most $c(vw)$. Hence, we obtain the first inequality. To prove the second inequality, replace each edge vw of $T_D(N)$ by a shortest v-w path of G, and call the resulting graph T'. Clearly, T' is a connected subgraph of G that contains all nodes in N, and moreover its cost $c(T') = \sum_{e \in E(T')} c(e)$ equals $c_D(T_D(N))$. Let T'' be a spanning tree of T'. Then T'' is a Steiner tree of G for the terminal node set N, and its cost $c(T'')$ is at most $c(T') = c_D(T_D(N))$. ▮

Lemma 7.2 *Let N be a set of terminal nodes. There exists a Steiner minimal tree $T_D(N)$ in the distance network $D = D_G(V)$ such that each Steiner node has degree at least 3.*

Proof: If $T_D(N)$ has any Steiner nodes having degree one, then these may be removed; the resulting tree is still a Steiner tree for the terminal set N. So suppose that every Steiner node of $T_D(N)$ has degree ≥ 2. Consider any Steiner node v of degree two, and let u and w be the nodes adjacent to v in $T_D(N)$. Clearly, the distance network D contains all three edges uw, uv and vw, and by the fact above, the edge costs satisfy the triangle inequality:

$$c_D(uw) \leq c_D(uv) + c_D(vw).$$

By replacing the node v and its two incident edges in $T_D(N)$ by the edge uw, we obtain a Steiner tree for N whose cost is at most $c_D(T_D(N))$. We repeat this procedure until there are no Steiner nodes of degree less than three. ▮

Proposition 7.3 *Let N be a set of terminal nodes. There exists a Steiner minimal tree $T_D(N)$ in the distance network $D = D_G(V)$ such that the number of Steiner nodes is at most $|N| - 2$.*

Proof: Let s be the number of Steiner nodes in $T = T_D(N)$. By the previous lemma, we may assume that each Steiner node has degree at least 3 in T. Hence, the sum of the degrees of all nodes in T is at least $|N| + 3s$. Also, the sum of the node degrees equals $2|E(T)| = 2|V(T)| - 2 = 2|N| + 2s - 2$. Since $|N| + 3s \leq 2|N| + 2s - 2$, we see that $s \leq |N| - 2$. ▮

7.2.2 The basic distance network heuristic

We construct the distance network $D_G(N)$, where N is a specified set of terminal nodes, and then find a minimum spanning tree M of $D_G(N)$. By replacing each edge vw of M by a shortest v-w path in G we obtain a new subgraph T' of G. Finally, we find a minimum spanning tree of T', repeatedly remove any Steiner nodes of degree one, and output the resulting tree T_{DNH}. Clearly, T_{DNH} contains all the terminal nodes. See Figure 7.2 for a worked example.

Algorithm Distance Network Heuristic for Steiner Minimal Trees
input: Graph $G = (V, E)$, nonnegative edge costs c, terminal node set N.
output: Tree T_{DNH} that contains all nodes in N; $c(T_{DHN}) \leq (2 - \frac{2}{|N|})c(T_G(N))$.

STEP 1: construct the distance network $D_G(N)$ for N, G and c;
(for each pair of terminal nodes $v,\ w$, edge vw has cost $c_D(vw) = d_G(v,w)$)
STEP 2: find a minimum spanning tree M of $D_G(N)$ (w.r.t. costs c_D);
STEP 3: replace each edge in M by a corresponding shortest path in $G,\ c$
to get a subgraph T' of G;
STEP 4: find a minimum spanning tree (call it T_{DNH}) for the subgraph of $G,\ c$
induced by the nodes in T';
STEP 5: repeatedly delete from T_{DNH} nonterminal nodes of degree one;
output the resulting tree T_{DNH};

Theorem 7.4 *For every graph G, cost function $c : E(G) \to \Re_+$, and terminal node set N, the distance network heuristic finds a Steiner tree T_{DNH} such that $c(T_{DNH}) \le (2 - \frac{2}{|N|})c(T_G(N))$.*

Proof: Let L be a walk on $T_G(N)$ that uses each edge exactly twice and that visits every node of $T_G(N)$. In other words, L is the "boundary" in a drawing of $T_G(N)$ on the plane. The cost of L is $c(L) = 2\ c(T_G(N))$. Order the terminal nodes according to their first occurrence on L, assuming that L starts with a terminal node z_1, and let the terminal nodes in the ordering be $z_1, z_2, \ldots, z_k$, where we use k for $|N|$ and we take z_{k+1} to be z_1. Let $L(z_i, z_{i+1}),\quad 1 \le i \le k$, denote the subwalk of L between the first occurrence of z_i and the first occurrence of z_{i+1}; note that $L(z_i, z_{i+1})$ may contain several other terminal nodes $z_j,\quad j < i$. By choosing the start node z_1 appropriately, we can ensure that the subwalk $L(z_k, z_{k+1}) = L(z_k, z_1)$ has cost at least $\dfrac{c(L)}{k} = \dfrac{2\ c(T_G(N))}{k}$. We remove the edges of this subwalk from L, and call the resulting walk L'. Clearly, L' has cost $c(L') \le \left(2 - \dfrac{2}{k}\right) c(T_G(N))$. The proof of the theorem follows from the next result.

Fact: The cost of T_{DNH} is at most $c(L')$.
Proof: To prove the claim, consider any subwalk $L(z_i, z_{i+1})$, $1 \le i \le k-1$, of L' and observe that the edge $z_i z_{i+1}$ in the distance network $D_G(N)$ has cost $c_D(z_i z_{i+1}) = d_G(z_i, z_{i+1}) \le c(L(z_i, z_{i+1}))$, since the minimum cost of a path in G from z_i to z_{i+1} is at most the cost of $L(z_i, z_{i+1})$. Hence, the union of the edges $z_i z_{i+1},\quad 1 \le i \le k-1$, of $D_G(N)$ gives a spanning tree T of $D_G(N)$ whose cost $c_D(T)$ is at most $\sum_{i=1}^{k-1} c(L(z_i, z_{i+1})) = c(L')$. Since the cost of the Steiner tree T_{DNH} found by the distance network heuristic is at most the cost of a minimum spanning tree of $D_G(N)$, it follows that $c(T_{DNH}) \le c_D(T) \le c(L')$. This proves the claim. ▮ ▮

Proposition 7.5 *The time complexity of the distance network heuristic is $O(|N|(|E|+|V|\cdot\log|V|))$.*

Proof: In the first step, in order to construct $D_G(N)$, we have to compute shortest paths costs starting from each terminal node $z \in N$. This needs $|N|$ applications of Dijkstra's shortest-paths algorithm. Each application of Dijkstra's algorithm contributes $O(|E| + |V| \cdot \log|V|)$ to the running time. The above time bound suffices for the remaining steps. ▮

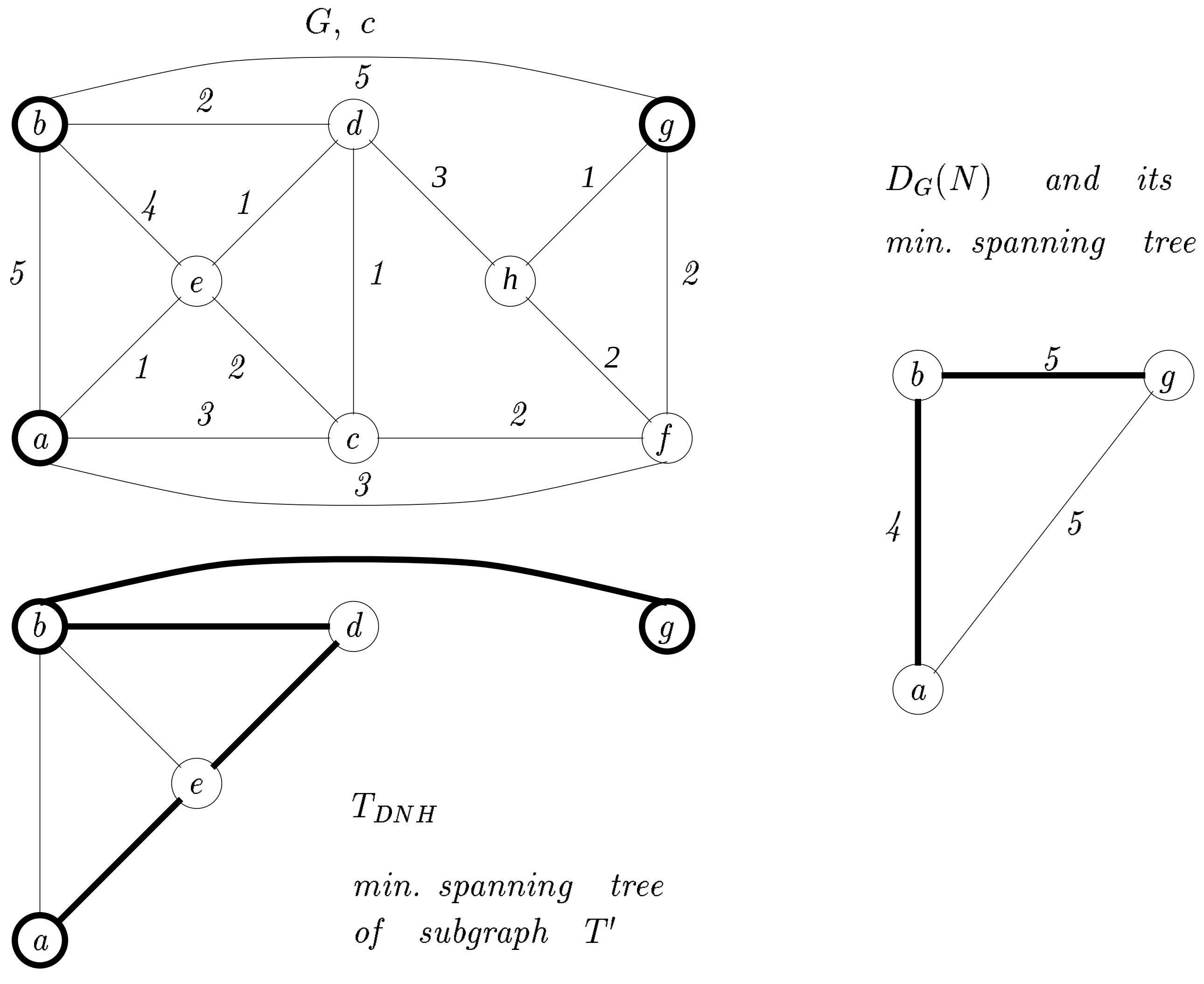

Figure 7.2: The example of the Steiner tree problem in Figure 7.1 is solved using the distance network heuristic. The terminal node set is $N = \{a, b, g\}$. The networks $G,\ c$ and $D_G(N),\ c_D$ are shown, along with subgraph T' and T_{DNH}. Note that T_{DNH} is not an SMT.

7.2.3 Mehlhorn's variant of the distance network heuristic

Mehlhorn [5] modifies the first step of the distance network heuristic so that it computes a graph $D' = (N, E')$ and arc costs $c' : E' \to \Re_+$ instead of the distance network $D_G(N)$. The rest of Mehlhorn's variant (after the construction of D' and c') is the same as in the distance network heuristic. It turns out that D' and c' can be computed by *one* application of Dijkstra's shortest-paths algorithm, hence the overall time complexity for finding T_{DNH} becomes $(|E| + |V| \cdot \log |V|)$. The performance guarantee of Mehlhorn's variant, $\dfrac{c(T_{DNH})}{c(T_G(N))}$, is the same as in the distance network heuristic, namely, $\left(2 - \dfrac{2}{|N|}\right)$.

For each terminal node $z_i \in N$, define $N(z_i)$ to be the set containing z_i and all nonterminal nodes $v \in V \backslash N$ that are "closer" to z_i than to any other terminal node z_j, $z_j \neq z_i$; if a nonterminal node v is "closest" to two or more terminal nodes $z_i, z_j, \ldots$, i.e., if $d_G(v, z_i) = d_G(v, z_j) = \cdots = \min_{z \in N} d_G(v, z)$, then v is included in the set of the terminal node having the smallest index among $i, j, \ldots$. In symbols,

$$N(z_i) \quad = \quad \{z_i\} \cup \left\{ \begin{array}{ll} v \in V \backslash N : & d_G(v, z_i) = \min_{z \in N} d_G(v, z) \text{ and} \\ & i = \min\{j = 1, \ldots, |N| \ : \ d_G(v, z_j) = \min_{z \in N} d_G(v, z)\} \end{array} \right\}.$$

Thus, the sets $N(z_i)$, $z_i \in N$, form a partition of the node set V. For every pair of terminal nodes z_i, z_j, if G has an edge between a node in $N(z_i)$ and a node in $N(z_j)$, then the edge $z_i z_j$ is present in D', i.e.,

$$E' = \{z_i z_j, \ z_i \in N, \ z_j \in N \ : \ \exists v \in N(z_i), \ \exists w \in N(z_j) \ : \ vw \in E(G)\}.$$

Informally, an edge of D' may "represent" several different edges of G. The cost of an edge $z_i z_j$ in E' is taken to be the minimum cost of a path from z_i to z_j in G that uses exactly one of the edges that "represent" $z_i z_j$. The path may contain several edges that either have both end nodes in $N(z_i)$ or have both end nodes in $N(z_j)$. That is, $c' : E \to \Re_+$ is defined by

$$c'(z_i z_j) = \min_{vw \in E(G), \ v \in N(z_i), \ w \in N(z_j)} \{d_G(z_i, v) + c(vw) + d_G(w, z_j)\}.$$

D' is a subgraph of the distance network $D_G(N)$, and it may be a proper subgraph of $D_G(N)$, i.e., some edges of $D_G(N)$ may be missing from D'. Moreover, even if an edge $z_i z_j$ of $D_G(N)$ is present in D', its cost in D', $c'(z_i z_j)$, may be greater than its cost in $D_G(N)$, $c_D(z_i z_j) = d_G(z_i, z_j)$. Figure 7.3 shows D' and $D_G(N)$ for an example.

D' can be easily constructed using a single application of Dijkstra's shortest-paths algorithm. We add a new start node v_0 to G, and for each terminal node $z_i \in N$, we add the edge $v_0 z_i$ to G and fix the edge cost to be zero, $c(v_0 z_i) = 0$. We assume that none of the original edges of G has zero cost. Then we take the start node to be v_0 and find a shortest-paths tree T_S with root v_0. Removing v_0 and its incident edges from T_S gives a collection of subtrees, each subtree containing exactly one terminal node z_i and zero or more nonterminal nodes (here, we use the assumption on the costs of the original edges of G). For each terminal node z_i, $N(z_i)$ is the set of nodes in the subtree containing z_i. The shortest-paths costs $d(v)$, $v \in V$, computed by Dijkstra's algorithm

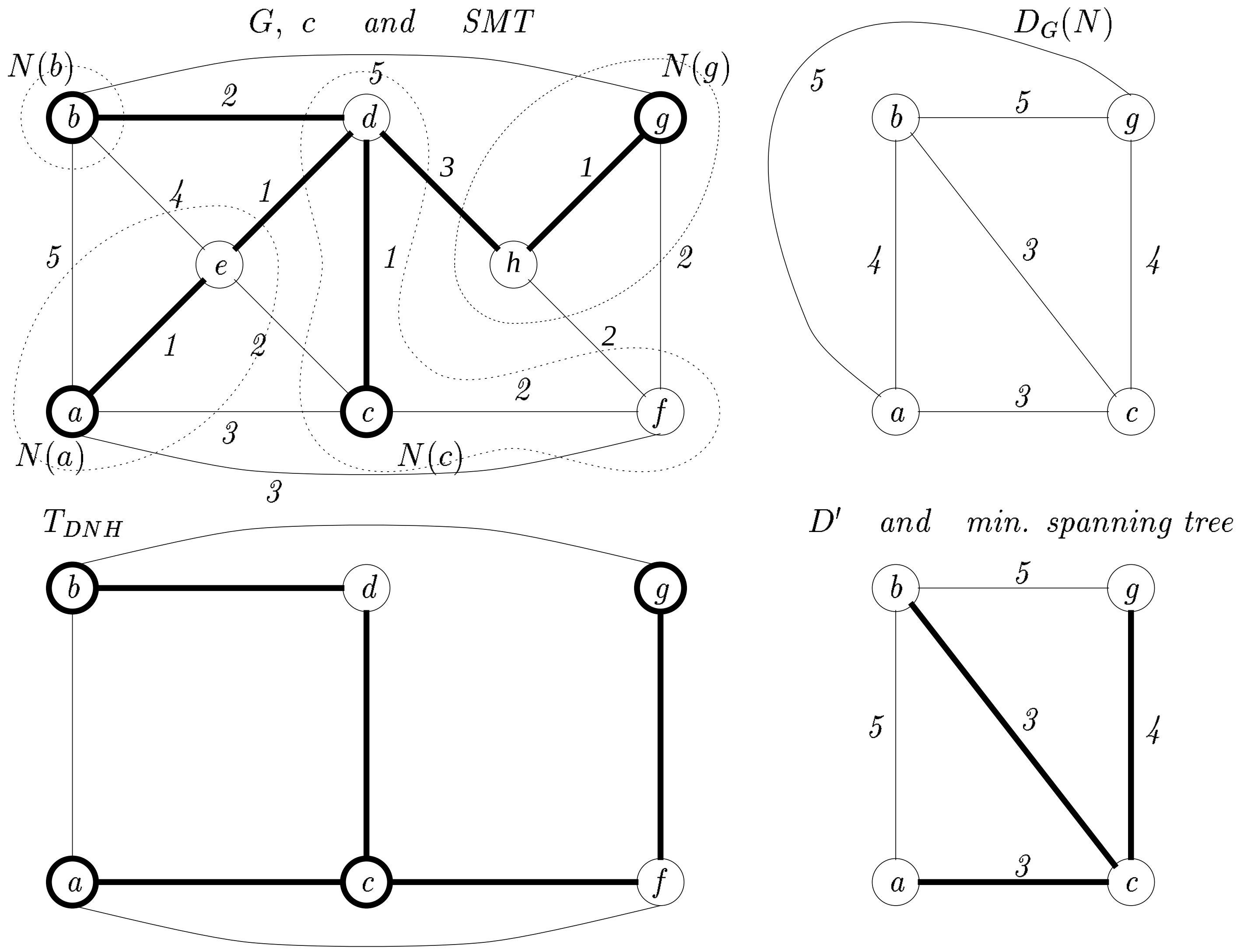

Figure 7.3: The example of the Steiner tree problem in Figure 7.1 is changed to have terminal node set $N = \{a, b, c, g\}$, and then solved using Mehlhorn's variant of the distance network heuristic. The network G, c is shown, along with an SMT and the node sets $N(z)$, $z \in N$. The networks $D_G(N)$, c_D and D', c' are shown for comparison, though Mehlhorn's variant uses only D', c'. Note that T_{DNH} is not an SMT.

are exactly the values $d_G(v, z_i)$, $z_i \in N$ and $v \in N(z_i)$, that we need to compute $c' : E' \to \Re_+$. By examining each edge vw of G such that v and w are in different sets $N(z_i)$ and $N(z_j)$, it is straightforward to compute the edge costs c' for D'. The next lemma, which is the key result on D', shows that every minimum spanning tree of D', c' is a minimum spanning tree of $D_G(N)$, c_D. Based on this lemma, we compute a minimum spanning tree M' of D', c'. Then we replace each edge vw of M' by a shortest path from v to w in G, c to obtain a subgraph T'. T_{DNH} can be obtained from T', by using the last two steps of the basic distance network heuristic. Figure 7.3 shows a worked example.

Theorem 7.6 *Mehlhorn's variant of the distance network heuristic computes T_{DNH} in time $O(|E| + |V| \cdot \log |V|)$.*

Lemma 7.7 *Let M' be a minimum spanning tree of D' with respect to the edge costs c'. Then M' is a minimum spanning tree of $D_G(N)$ with respect to the edge costs c_D.*

Proof: We claim that there exists a minimum spanning tree M^* of $D_G(N)$ such that each edge z_iz_j of M^* is present in D', and the cost of the edge in D', $c'(z_iz_j)$, equals the cost in $D_G(N)$, $c_D(z_iz_j)$. Suppose that the claim is true. Then M^* is obviously a minimum spanning tree of D' w.r.t. c', since any spanning tree of D' has cost at least $c_D(M^*)$. By definition, any other minimum spanning tree M' of D' w.r.t. c' has exactly the same cost as M^*, i.e., $c'(M') = c'(M^*) = c_D(M^*)$, therefore M' must be a minimum spanning tree of $D_G(N)$ w.r.t. c_D.

The claim is proved by contradiction. Suppose that the claim is false. Let M be a minimum spanning tree of $D_G(N)$ that

(i) has the maximum number of edges in D', i.e., $|E(M) \cap E'|$ is maximized over all minimum spanning trees of $D_G(N)$, c_D, and moreover,

(ii) the total cost with respect to c' of the edges in both M and D' is minimum, i.e., $\sum_{e \in E(M) \cap E'} c'(e)$ is minimized over all minimum spanning trees of $D_G(N)$, c_D satisfying (i).

Either an edge z_sz_t of M is not present in D', or there is an edge z_sz_t of M with $c'(z_sz_t) > c_D(z_sz_t) = d_G(z_s, z_t)$. Let P be a shortest path from z_s to z_t in G, with respect to the costs c. Consider any edge v^*w^* of P such that v^* and w^* are in distinct sets, say, $v^* \in N(z_i)$ and $w^* \in N(z_j)$. By the definition of $N(z_i)$, we get $d_G(z_i, v^*) \leq d_G(z_s, v^*)$, and by the definition of $N(z_j)$, $d_G(z_j, w^*) \leq d_G(z_t, w^*)$. Clearly, D' contains the edge z_iz_j. The cost of this edge w.r.t. c' is

$$\begin{aligned}
c'(z_iz_j) &= \min_{vw \in E(G),\ v \in N(z_i),\ w \in N(z_j)} \{d_G(z_i, v) + c(vw) + d_G(w, z_j)\} \\
&\leq d_G(z_i, v^*) + c(v^*w^*) + d_G(w^*, z_j) \\
&\leq d_G(z_s, v^*) + c(v^*w^*) + d_G(w^*, z_t) \\
&= c(P) \\
&= d_G(z_s, z_t) \\
&= c_D(z_sz_t).
\end{aligned}$$

That is, for every one of the edges v^*w^* of P with v^* and w^* in distinct sets $N(z_i)$ and $N(z_j)$, the edge z_iz_j is present in D', and $c'(z_iz_j) \leq c_D(z_sz_t)$. These edges form a walk P' from z_s to z_t in D' since P is a z_s-z_t path in G: indeed, if we start with the subgraph P and for each $z_i \in N$ we "contract" all nodes in $N(z_i)$ into a single node z_i', then we obtain P'.

If we remove edge z_sz_t from M, then we obtain a partition N_s, N_t of the nodes of $D_G(N)$, where N_s is the set of nodes in the subtree of $M\backslash\{z_sz_t\}$ containing z_s, and $N_t = N\backslash N_s$. Since P' is a walk from z_s to z_t, at least one of its edges $z_i^*z_j^*$ has one end node in N_s and the other in N_t, i.e., $z_i^*z_j^*$ is in the cut (N_s, N_t). By replacing the edge z_sz_t in M by the edge $z_i^*z_j^*$, we obtain another spanning tree $\overline{M}$ of $D_G(N)$. $\overline{M}$ is a minimum spanning tree of $D_G(N)$ because its cost $c_D(\overline{M}) = c_D(M) - c_D(z_sz_t) + c_D(z_i^*z_j^*)$ is at most $c_D(M)$, since $c_D(z_i^*z_j^*) \leq c'(z_i^*z_j^*) \leq c_D(z_sz_t)$ by the derivation above. Now, by our assumption on M, either edge z_sz_t is not in D' or z_sz_t is in D' but $c'(z_sz_t) > c_D(z_sz_t)$. In the first case, $\overline{M}$ has more edges in D' than M, and in the second case

$$\sum_{e\in E(\overline{M})\cap E'} c'(e) < \sum_{e\in E(M)\cap E'} c'(e) \qquad \text{since } c'(z_i^*z_j^*) \leq c_D(z_sz_t) < c'(z_sz_t).$$

In either case, we have a contradiction to the definition of M, and so our claim is proved. The lemma follows. ∎

7.3 The Dreyfus-Wagner dynamic programming algorithm

The Dreyfus-Wagner algorithm (1972) is one of the oldest exact algorithms for finding a Steiner minimal tree. Its time complexity is exponential in $|N|$. The algorithm is based on dynamic programming. It uses a recursion to derive the cost of a Steiner minimal tree from the costs of Steiner minimal trees for subsets of the given terminal node set. For a node set $D \subseteq V$ and a node $v \in V$, let $S(v, D)$ denote the cost of a Steiner minimal tree for the terminal node set $D \cup \{v\}$. The recursion is:

$$S(v, D) = \min_{w\in V}\left\{d_G(v,w) + \min_{\emptyset\neq D'\subset D}\left(S(w, D') + S(w, D\backslash D')\right)\right\}.$$

Note that D' is a proper subset of D in the recursion. Possibly, $v = w$ in the recursion, in which case $d_G(v, w) = 0$. Informally, in a Steiner minimal tree for $D \cup \{v\}$, node w is a "junction node", i.e., a terminal node or a nonterminal node of degree at least 3, such that all internal nodes in the path from v to w are nonterminal nodes of degree 2. The three main cases of the recursion are shown in Figure 7.4.

Let N be the given set of terminal nodes, and let q be an arbitrary node in N. Initially, the algorithm computes $d_G(v, w)$ for all pairs of nodes $v,\ w \in V$; note that $S(v, \{w\}) = d_G(v, w)$. Then for all integers ℓ between 2 and $|N| - 2$, in sequence, for all $D \subseteq N\backslash\{q\}$ such that $|D| = \ell$ and for all $v \in V$, $S(v, D)$ is computed. Finally, $S(q, N\backslash\{q\})$ is computed, and this gives the cost of a Steiner minimal tree.

Theorem 7.8 *The Dreyfus-Wagner algorithm computes the cost of a Steiner minimal tree. The running time of the algorithm is* $O(3^{|N|}(|V|) + 2^{|N|}(|V|^2) + |V|^3)$.

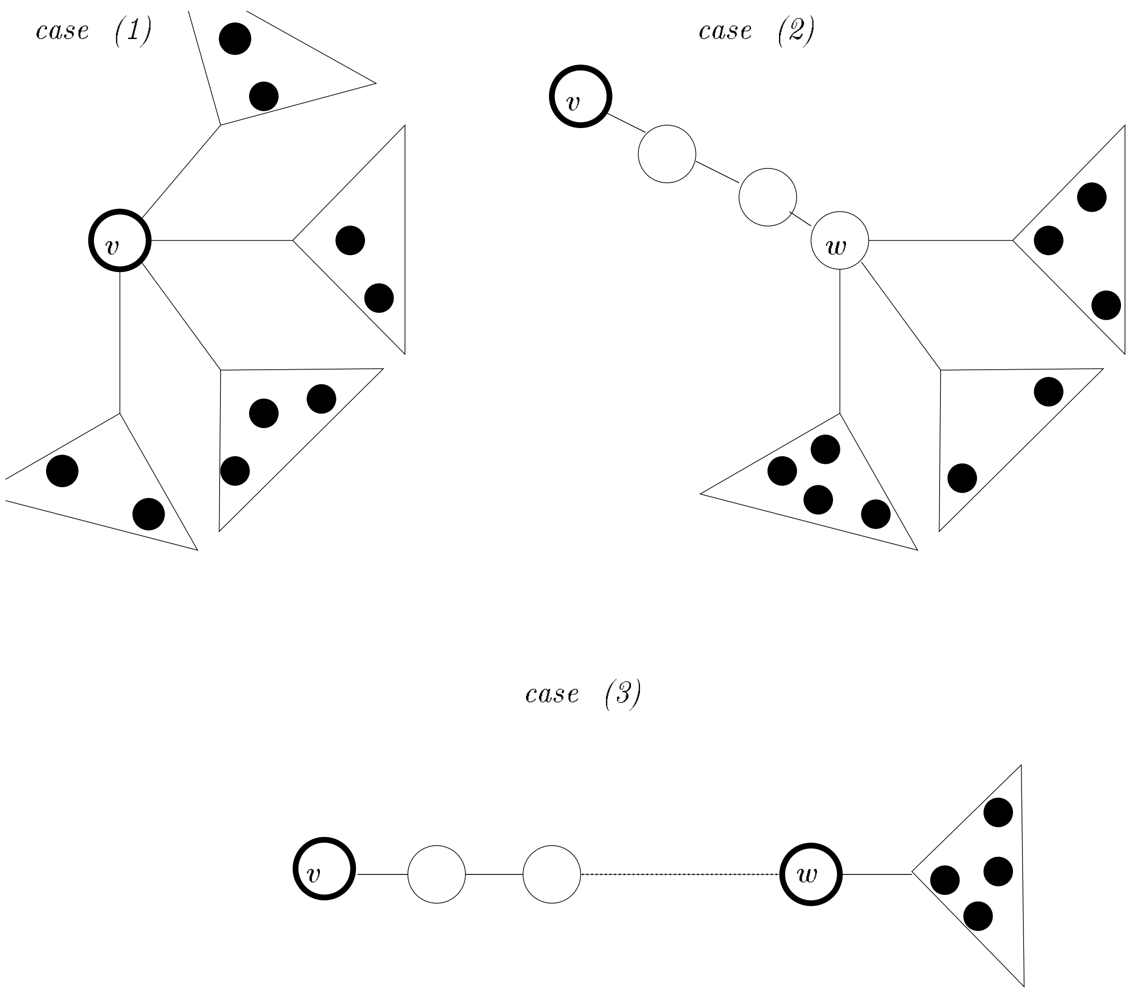

Figure 7.4: An illustration of the three main cases in the Dreyfus-Wagner recursion. The node v and the terminal nodes are indicated by either bold circles or shaded circles.

If $|N|$ is small, say, $|N| = O(\log\ |V|)$, then note that the algorithm runs in polynomial time.

The algorithm has to be supplemented in order to find a Steiner minimal tree, rather than the cost of an SMT. This is done by backtracing, that is, by examining the sets $D \subseteq N$ in the reverse order of the cost-determining computation, assuming that the values $S(v, D)$, $v \in V$, $D \subset N$, have been stored for later use. We start with $D = N \backslash \{q\}$, and find the "junction node" w and the partition $D = D' \cup D''$, $D' \cap D'' = \emptyset$, that determine the value of $S(q, D)$. Then we compute the "junction node" and the optimal partition for each of D' and D'' by recomputing $S(w, D')$ and $S(w, D'')$. Continuing this process, we find the Steiner minimal tree.

Algorithm Dreyfus-Wagner Dynamic Programming Algorithm for Steiner Minimal Trees.
input: Graph $G = (V, E)$, nonnegative edge costs c, terminal node set N.
output: Cost of a Steiner minimal tree.

```
STEP 0: for every pair of nodes v, w ∈ V, do
            compute d_G(v, w), the minimum cost of a v-w path;
        end;    (for)
        fix q to be an arbitrary node in N;
STEP 1: (singleton sets)
        for each t ∈ N\{q} do
            for each v ∈ V do S(v, {t}) := d_G(v, t); end;
        end;    (for)
STEP 2: (subsets of size 2, 3, ..., |N| − 2)
        for ℓ := 2 to |N| − 2 do
            for each D ⊂ N\{q} such that |D| = ℓ do
                S(v, D) := ∞, for all v ∈ V;    (initialize)
                choose an arbitrary node z ∈ D;
                for each w ∈ V do
                    s := ∞;    (temporary variable)
                    for each D' ⊂ D with z ∈ D' do
                        s := min(s, S(w, D') + S(w, D\D'));
                    end;    (for)
                    for each v ∈ V do
                        S(v, D) := min(S(v, D), d_G(v, w) + s);
                    end;    (for)
                end;    (for)
            end;    (for)
        end;    (for)
STEP 3: (find final solution for N\{q})
        C := ∞;    (initialize)
        choose an arbitrary node z ∈ N\{q};
        for each w ∈ V do
            s := ∞;    (temporary variable)
            for each D' ⊂ N\{q} with z ∈ D' do
                s := min(s, S(w, D') + S(w, (N\{q})\D'));
            end;    (for)
            C := min(C, d_G(q, w) + s);
        end;    (for)
        output C;
```

7.4 Zelikovsky's Algorithm

In this section, we present an algorithm with improved performance ratio for the Steiner problem due to Zelikovsky [7]. This version of the proof appeared in two lectures in January 1997 at MPI, Saarbruecken, Germany, by Ulrich Fössmeyer; The following treatment is from a manuscript by Kurt Mehlhorn, in which Naveen Garg filled in some details. The proof of a key lemma is incomplete however. The following description may be useful in presenting a different point of view of looking at Zelikovsky's algorithm.

7.4.1 Definitions

Recalle that we used $T_G(N)$ to denote the optimal Steiner tree in G for the terminals N. In the sequel, whenever we consider optimal Steiner trees for subsets S of nodes, we always do so in the input graph G so for ease of notation we drop the subscript and assume that we always compute the optimal tree in G. Thus we shall henceforth use $T(S)$ to denote an optimal Steiner tree for the terminal set S in G.

7.4.2 The algorithm

The starting point of Zelikovsky's algorithm is the 2-approximate solution from the distance network heuristic. The algorithm maintains a complete graph on the terminal nodes. The edge costs of this graph decrease as the algorithm progresses through several phases. In particular, in each phase, the cost of exactly two edges decreases. The approximate solution maintained by the algorithm is the minimum cost spanning tree of this graph.

Suppose there are f phases in the algorithm. Let G_i represent the graph maintained by the algorithm just before the i^{th} phase. Thus, in our notation, $G_0 = D_G(N)$, the complete graph on node set N with costs denoting shortest distances in G. Let M_i denote the minimum cost spanning tree of G_i. The performance guarantee of the distance network heuristic guarantees that $c(T_0) < 2c(T(S))$.

Before we describe the details of each phase, we need a few more definitions. For terminals x and y we use $Bridge_i(x, y)$ to denote the heaviest edge on the path from x to y in M_i. For a triple $Tr = \{x, y, z\}$ of distinct vertices let

$$Bridge_i(Tr) = \{Bridge_i(x, y), Bridge_i(x, z), Bridge_i(y, z)\}.$$

Proposition 7.9 $|Bridge_i(T)| = 2$.

Proof: Let M_{Tr} be the subtree of M_i spanned by Tr. The tree M_{Tr} contains a unique point c such that the tree-paths p_x, p_y, and p_z from c to x, y, and z respectively are pairwise edge-disjoint[1]. Assume without loss of generality that the heaviest edge of M_{Tr} lies on p_x then $Bridge_i(x, y) = Bridge_i(x, z) \neq Bridge_i(y, z)$. ∎

For ease of notation we assume in the sequel that $Bridge_i(Tr)$ consists of $Bridge_i(x, y)$ and $Bridge_i(x, z)$. In this case, we also say that the triple Tr is x-light with respect to the tree M_i.

[1] c may be equal to one of x or y or z in which case the corresponding path has length zero.

Note that this is equivalent to saying that the heaviest edge in M_{Tr} defined in the proof above lies on the path p_x. Note that $c(Bridge_i(Tr)) = c(Bridge_i(x,y)) + c(Bridge_i(x,z))$. For a triple Tr let

$$gain_i(T) = c(Bridge_i(T)) - c(T(Tr)).$$

Zelikovsky's algorithm terminates when there is no triple of positive gain. If there is a triple of positive gain in the i-th phase let $Tr_i = \{x,y,z\}$ be the triple with maximum gain. We obtain G_{i+1} by adding edges xy and xz to G_i with costs $c(Bridge_i(x,y)) - gain_i(Tr_i)$ and $c(Bridge_i(x,z)) - gain_i(Tr_i)$ respectively. The minimum spanning tree M_{i+1} of G_{i+1} is clearly $M_i - Bridge_i(Tr_i) \cup \{xy, xz\}$. Also, $c(M_{i+1}) = c(M_i) - 2 \cdot gain_i(Tr_i)$, i.e., the cost of the minimum spanning tree decreases by twice the gain. This implies that

$$c(M_f) = c(M_0) - 2 \cdot \sum_i gain_i(Tr_i). \tag{7.1}$$

7.4.3 Performance Guarantee

To analyze the performance guarantee we first observe some important properties of the spanning tree M_i during the course of the algorithm. We do this next.

In each phase the algorithm replaces two edges of the current spanning tree by new ones. It is helpful to view these replacements as taking place consecutively. So, the M_i's evolve by selecting pairs $\{a,b\}$ of terminals and replacing $Bridge_i(a,b)$ by ab. The new edge has lower cost than the old edge, the replaced edge had the largest cost on the cycle closed by the new edge, and the new edge separates exactly the same terminals as the old edge, i.e., for any terminals x and y, the path from x to y went through $Bridge_i(a,b)$ before the replacement if it goes through ab after the replacement.

This allows us to argue that for any two terminals x and y the cost of their bridge never increases.

Lemma 7.10 *For any two terminals x and y and for any $i \in \{0,1,\ldots,f-1\}$, the following is true: $c(Bridge_i(x,y)) \geq c(Bridge_{i+1}(x,y))$.*

Proof: When moving from M_i to M_{i+1} we replace two edges. A replacement substitutes an edge ab for the edge $Bridge_i(a,b)$. If ab does not lie on the path from x to y then $Bridge_i(x,y)$ does not change. Otherwise, the edge ab closes a cycle $p \circ q$ in M_i and paths p_x and p_y connect x and y to the common endpoints of p and q. In M_i the path from x to y is $p_x \circ q \circ p_y$ and in M_{i+1} the path from x to y is $p_x \circ p \circ p_y$. Since $Bridge_i(a,b)$ is the heaviest edge on the cycle $p \circ q$ the claim follows. ∎

We can now show that every edge whose cost is decreased and hence is included in the minimum spanning tree is retained in the final tree M_f.

Lemma 7.11 *For all j, $Bridge_j(Tr_j)$ consists only of edges of G_0.*

Proof: The proof of this lemma is at the crux of the analysis but is however beyond the scope of these notes. A short version of the proof may be found in the Appendix of [3]. ∎

The Steiner tree finally output when the algorithm terminates is the shortest paths in G corresponding to edges of G_0 retained in M_f, as well as the optimal trees for all the triples used to decrease edge costs in the course of the algorithm. It is now straightforward to bound the cost of this tree.

Lemma 7.12 *The Steiner tree output by Zelikovsky's algorithm has cost at most* $(c(M_0)+c(M_f))/2$.

Proof: Each edge of M_f is either original or new. Lemma 7.11 tells us that once a new edge is added to some M_i it will stay in the minimum spanning tree of the auxiliary graph till the end of the algorithm. In other words, M_f consists of some original edges and the new edges introduced by the triples, Tr_0, Tr_1, ..., Tr_{f-1}. Construct a subgraph G' of G consisting of the paths corresponding to the edges of $M_f \cap G_0$ and the Steiner minimal trees $T(Tr_0)$, ..., $T(Tr_{f-1}))$ and let $\tilde{T}$ be a minimum spanning tree of G'. The cost of $\tilde{T}$ is at most $c(M_f) + \sum_i (c(T(Tr_i)) - (c(Bridge_i(Tr_i)) - 2 \cdot gain_i(Tr_i))) = c(M_f) + \sum_i gain_i(Tr_i)$ since any triple Tr_i adds a Steiner tree of cost $c(T(Tr_i)($ to G' in replacement of edges of cost $c(Bridge_i(Tr_i)) - 2 \cdot gain_i(Tr_i)$ of M_f. Hence,

$$\begin{aligned} c(\tilde{T}) &\leq c(M_f) + \sum_i gain_i(Tr_i) \\ &= (c(M_f) + c(M_f) + 2 \cdot \sum_i gain_i(Tr_i))/2 \\ &= (c(M_f) + c(M_0))/2 \end{aligned}$$

since $c(M_f) = c(M_0) - 2 \cdot \sum_i gain_i(Tr_i)$ from equation 7.1. ∎

We recall the main result from the previous section.

Lemma 7.13 $c(M_0) < 2 \cdot c(T(N))$.

To bound the cost of the tree $\tilde{T}$ output by Zelikovsky's algorithm, we still need to bound $c(M_f)$ independent of $C(M_0)$. We shall prove the following result in the next section.

Lemma 7.14 $c(M_f) \leq \frac{5}{3} \cdot c(T(N))$.

With this, we can prove the performance guarantee of Zelikovsky's algorithm.

Theorem 7.15 *Zelikovsky's algorithm finds a Steiner tree* $\tilde{T}$ *with cost at most* $\frac{11}{6}$ *times the minimum.*

Proof: From lemmas 7.12, 7.13 and 7.14, we get

$$\begin{aligned} c(\tilde{T}) &\leq (c(M_0) + c(M_f))/2 \\ &\leq (2 + 5/3) \cdot c(T(N))/2 \\ &= (11/6) \cdot c(T(N)). \end{aligned}$$

∎

7.4.4 Full Steiner Trees

A Steiner tree for terminal set S is said to be *full* if all nodes in S are leaves of this tree. The *full components* of any Steiner tree for S are its maximal full subtrees. Finally, a Steiner tree of S is said to be *k-restricted* if all its full components have at most k leaves. We shall denote an optimal (minimum cost) k-restricted Steiner tree for S by $T^k(S)$. Note that $T(S) = T^{|S|}(S)$.

We prove Lemma 7.14 in this section by proving the following two results.

Lemma 7.16 $c(M_f) \leq c(T^3(N))$.

Lemma 7.17 $c(T^3(N)) \leq \frac{5}{3} \cdot c(T(N))$.

Note that Zelikovsky's algorithm and the analysis via the above two lemmas can be used to derive even better performance guarantees for the case of special metrics such as rectilinear or ℓ_1 metrics. See [1] for more on this development.

Next we demonstrate that M_f is not costlier than an optimal 3-restricted Steiner tree by examining the natural 3-restricted tree defined by it on the terminals.

Proof: of Lemma 7.16 Let M_f be the minimum spanning tree of the final auxiliary graph. There are no triples of positive gain with respect to M_f. Consider an optimal 3-restricted Steiner tree $T^3(N)$. Each full component of $T^3(N)$ has either two or three leaves. Let Tr be the triples of terminals induced by the full components with three leaves and let Pr be the pairs of terminals induced by the full components with two leaves. Then

$$c(T^3(N)) = \sum_{tr \in Tr} c(T(tr)) + \sum_{pr \in Pr} c(T(pr))$$

We need to show that $c(M_f) \leq c(T^3(N))$. To this end, we define a tree M_a of cost at most $c(T^3(N))$ on N and then show that M_f is a minimum spanning tree of the graph $M_f \cup M_a$. The edges of M_a are defined as follows: For each pair $p = xy \in Pr$ we put the edge xy with cost $c(T(pr))$ into M_a and for each triple $tr = \{x, y, z\}$ we put the edges xy and xz into M_a with costs $c(Bridge_f(x, y))$ and $c(Bridge_f(x, z))$, respectively[2]. Since $gain_f(tr) \leq 0$ we have $c(Bridge_f(tr)) \leq c(T(tr))$. Thus, $c(M_a) \leq c(T^3(N))$. The graph M_a is clearly connected and since the number of edges of M_a is $|Pr| + 2|Tr| = |N| - 1$, M_a is a tree.

It remains to show that M_f is a minimum spanning tree of $M_f \cup M_a$. This follows from the cycle rule for minimum spanning trees. Assume that we start with M_f and add the edges of M_a one by one. Let xy be an edge of M_a. If xy comes from a pair p then the cost $c(T(pr))$ of the edge is at equal to the cost of the edge xy in G_0. Similarly, if xy comes from a triple then it is clearly a heaviest edge in the cycle which it closes. ∎

Finally we bound on the cost of an optimal 3-restricted tree with respect to the optimum solution[3].

Proof: of Lemma 7.17

Recall that $T(N)$ denotes an optimal Steiner tree for the terminals N. We may assume w.l.o.g. that $T(N)$ is full. Otherwise, we apply the following argument to each full component.

[2] Recall that we adopted the convention that $Bridge(T)$ consists of $Bridge(x, y)$ and $Bridge(x, z)$.

[3] The following proof is due to Vijay Vazirani transmitted via Naveen Garg.

We also assume for the moment that $T(N)$ is binary. If not, we replace any internal node of out-degree larger than 2 by a chain of nodes of degree 3, give all new edges cost zero, and apply the argument to the resulting tree. If there are nodes of degree 2 in the original tree, we may shorcut over such nodes by using the triangle inequality on the costs (w.l.o.g.).

Root $T(N)$ at an arbitrary internal vertex, add an edge to the root, and three-color the edges of $T(N)$. We define a 3-restricted Steiner tree for each color class of edges and show that the total cost of the three trees is at most $5c(T(N))$. The lemma follows by averaging.

We may assume w.l.o.g. that $T(N)$ is ordered such that for every vertex v the path to the rightmost leaf $\alpha(v)$ in its subtree is the shortest path to any leaf in v's subtree. Note that this definition is consistent with all the internal nodes: if the internal node u has l as its closest leaf and the path from u to l uses the child a of u, then l is also a closest leaf to a. We define the 3-restricted Steiner tree for a color class by specifying the triples of nodes that are to form full subtrees.

Consider any of the color classes and let $(u, |parent|(u))$ be an edge of the color class.

- If u has children a, b and a sibling c then include the optimal Steiner tree for the triple $\{\alpha(a), \alpha(b), \alpha(c)\}$ into the Steiner tree for the color class and call $\{a, b, c\}$ the core of the triple.

- If u has children a, b but no sibling, i.e., u is the root, add the edge $\{\alpha(a), \alpha(b)\}$ into the Steiner tree for the color class and call $\{a, b\}$ the core of the pair.

- If u has a sibling c but no children, i.e., u is a terminal, add the edge $\{u, \alpha(c)\}$ into the Steiner tree for the color class and call $\{u, c\}$ the core of the pair.

We claim that all the edges added for a color class define a Steiner tree for N, which, by construction, is 3-restricted. This is readily proved by induction. Consider the edges of any color class bottom up and argue that after considering an edge $(u, parent(u))$ all terminals in the subtree rooted at $parent(u)$ are connected by the edges added so far for this color class.

It remains to show that the total cost of the three trees is at most $5 \cdot (T(N))$. Note first that the cost of each tree is the sum of the costs of the minimum Steiner trees of the pairs and triples defining the tree. Note next that the cost of the minimum Steiner tree for a pair of triple is bounded by the cost of the subtree of $T(N)$ spanned by the pair or triple. The cost of the subtree spanned by a pair or triple has two components:

- the cost of subtree spanned by the elements in the core of the pair or triple.

- the cost of the paths connecting v to $\alpha(v)$ for v in the core of the triple or pair and

We bound the total cost of the two contributions separately.

Consider first the subtrees spanned by the cores of pairs and triples. They have the shapes shown in Figure 7.5.

Each edge of T(N) belongs to exactly three such trees, namely once as the edge inducing the pair or triple, once as the sibling of the inducing edge, and once as the child of the inducing edge. Thus, the sum of the costs of the subtrees spanned by the cores is exactly $3 \cdot (T(N))$.

It remains to sum the costs of the paths from v to $\alpha(v)$ over all v over all cores. We use S to denote this sum. Clearly, we need to consider only v's that are not terminals (leaves). Each such

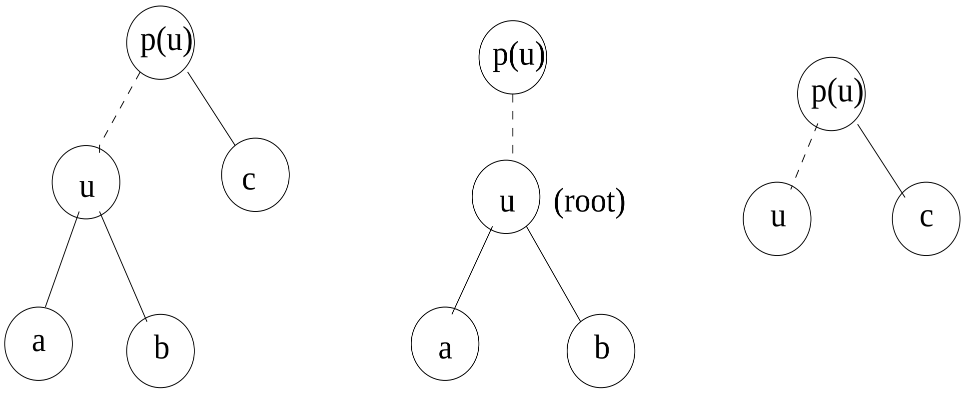

Figure 7.5: Shapes of subtrees spanned by cores of pairs and triples.

v is contained in the boundary of at most two subtrees spanned by cores, namely the core induced by the edge connecting the sibling of v with the parent of v and the core induced by the edge connecting the parent of v with the grand-parent of v. We conclude that S is bounded by twice the cost of the paths from v to $\alpha(v)$ summed over all v. Let $\beta(v)$ be the leaf reached from v by going right once and then always left. The path $v \longrightarrow \beta(v)$ is at least as long as the path $v \longrightarrow \alpha(v)$ and the former paths are disjoint. Thus, the sum of their lengths is bounded by $c(T(N))$ completing the proof. ▮

Further improvements on Zelikovsky's ideas are investigated by Berman and Ramaiyer[2].

7.5 Exercises

1. Consider the Steiner tree problem. Suppose that the set S^* of Steiner nodes in an optimal solution is available. How would you find a Steiner minimal tree?

2. Let N be a set of points in the Euclidean plane. Take the cost of an edge ij to be the Euclidean distance between points i and j.

   (a) Let T^* be a Steiner minimal tree in the Euclidean plane, and let x be a Steiner point in T^*. Explain whether the number of edges of T^* incident to x can be either (i) one? or (ii) two?

   (b) Let $|MST(N)|$ denote the cost of a minimum spanning tree of N, and let $|SMT(N)|$ denote the cost of a Steiner minimal tree. Prove that $|MST(N)| \leq (2 - \frac{2}{N})\ |SMT(N)|$.

3. Consider the following example of the set covering problem consisting of 10 points $x_1, \ \ldots, \ x_{10}$,

and 7 sets $S_1, \ldots, S_7$. The incidence matrix is

$$
\begin{array}{cc}
 & \begin{array}{ccccccc} S_1 & S_2 & S_3 & S_4 & S_5 & S_6 & S_7 \end{array} \\
A = & \left[\begin{array}{ccccccc}
 & & & & 1 & & 1 \\
 & & & 1 & & 1 & \\
 & & 1 & & 1 & & 1 \\
 & 1 & & 1 & & 1 & 1 \\
1 & & 1 & & 1 & 1 & \\
1 & 1 & & 1 & & & \\
1 & 1 & 1 & 1 & & & \\
1 & 1 & 1 & & 1 & & \\
1 & 1 & 1 & & & 1 & \\
1 & 1 & 1 & & & & 1
\end{array}\right]
\begin{array}{c} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \end{array}
\end{array}
$$

(a) Convert the set covering example into an example of the Steiner tree problem. Use the shortest paths heuristic to find a solution T_{SPH} to the Steiner tree problem. What is the theoretical guarantee g on the ratio $\frac{c(T_{SPH})}{c(T_{OPT})}$, where T_{OPT} is a Steiner minimal tree?

(b) Use your solution in part (a) to find a solution to the set covering example. Explain your working.

(c) An optimal solution of the dual (D) of the LP relaxation of the integer programming formulation (IP) of the set covering problem is

$$
\begin{array}{lllll}
y_1 = 1 & y_2 = \frac{1}{2} & y_3 = 0 & y_4 = 0 & y_5 = 0 \\
y_6 = 0 & y_7 = \frac{1}{2} & y_8 = 0 & y_9 = \frac{1}{2} & y_{10} = 0,
\end{array}
$$

where y_i is the dual variable for point x_i. Discuss whether your solution in part (b) is optimal, and explain your reasoning.

(d) Consider solving the set covering problem (in general), using the algorithm in parts (a) and (b). Is it true that the set cover J_{algo} found by this algorithm satisfies $|J_{algo}| \leq g \cdot |J_{opt}|$, where J_{opt} is an optimal solution, and g is as in part (a)? Explain your answer.

4. Solve (approximately) the Steiner tree problem defined by the graph $G = (V, E)$ and the edge costs c in Figure 5.2 taking the terminal node set to be $N = \{a, b, f\}$. Use each of the following algorithms:

   (a) the shortest paths heuristic,

   (b) Mehlhorn's variant of the distance network heuristic, and

   (c) Dreyfus and Wagner's dynamic programming algorithm.

5. Repeat parts (a)–(c) of the previous problem for the example of the Steiner tree problem in Figure 7.6; the terminal node set is $N = \{a, c, e, g\}$.

6. For a set of terminal nodes N, let $\ell(N)$ denote the minimum number of leaf nodes in a Steiner minimal tree of N. Clearly, $2 \leq \ell(N) \leq |N|$. Show that the performance guarantee of the distance network heuristic can be improved to

$$
\frac{c(T_{DNH})}{c(T_G(N))} \leq \left(2 - \frac{2}{\ell(N)}\right).
$$

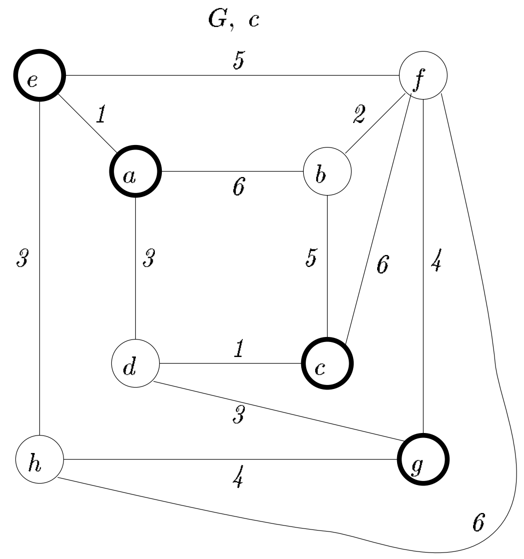

Figure 7.6: An example of the Steiner tree problem.

7. Give a detailed implementation of Mehlhorn's distance network heuristic, focusing on Steps 1, 3 and 5 of the basic distance network heuristic (page 86). Show that Step 4 (finding a minimum spanning tree of the (node-induced) subgraph T' of G) can be avoided by appropriately implementing Step 3 (transforming minimum spanning tree M of $D_G(N)$ into subgraph T' of G) using information from Step 1 (constructing $D_G(N)$).

8. The goal here is to study an integer programming formulation (IP) of the Steiner tree problem that is similar to the integer programming formulation of the minimum spanning tree problem in Chapter 5. Let $G = (V, E)$ be the input graph, let N be the set of terminal nodes, and let the cost of edge ij be denoted by c_{ij} $(c_{ij} \geq 0,\ \forall_{ij} \in E)$. The integer program (IP) has a zero-one variable z_i for each node $i \in V \backslash N$. The intention is that an optimal solution of (IP) has $z_i^* = 1$ iff node i is a Steiner node in the corresponding Steiner minimal tree.

$$
\boxed{
\begin{array}{llrcll}
\text{(IP)} & \text{minimize} & \sum\limits_{ij \in E} c_{ij} x_{ij} & & & \\
 & \text{subject to} & \sum\limits_{ij \in E} x_{ij} & \geq & \Big(\sum\limits_{i \in V \backslash N} z_i\Big) + |N| - 1 & \\
 & & \sum\limits_{ij \in E:\ i \in S, j \in S} x_{ij} & \leq & \Big(\sum\limits_{i \in S \backslash N} z_i\Big) + |S \cap N| - 1, & \forall S \subseteq V \\
 & & x_{ij} & \in & \{0, 1\}, & \forall ij \in E \\
 & & z_i & \in & \{0, 1\}, & \forall i \in V \backslash N
\end{array}
}
$$

(a) Consider the Steiner tree problem with G, c as in Figure 5.2 and terminal node set $N = \{a, b, e\}$. For this instance, formulate (IP), the LP relaxation (P), and the dual LP (D) of (P). Solve (IP) and (P) using the CPLEX optimizer.

(b) Discuss the complementary slackness conditions for the LP (P), and use these to design a primal-dual algorithm that (approximately) solves the Steiner tree problem.

(Note: When you submit (IP) to CPLEX, make sure that the right-hand side of each constraint is a *number*, i.e., move all the z_i's to the left-hand side.)

Bibliography

[1] P. Berman, U. Fössmeyer, M. Karpinski, M. Kauffman, and A. Zelikovsky, *Approaching the 5/4 approximation for rectilinear Steiner trees*, Proc. of European Symposium on Algorithms '94, LNCS **855** (1994), pp. 60-71.

[2] P. Berman and V. Ramaiyer, *Improved approximations for the Steiner tree problem*, J. Algorithms, **17(3)** (1994), pp. 381-408.

[3] D. Z. Du, Y. Zhang and Q. Feng, *On better heuristics for Euclidean Steiner minimum trees*, FOCS 91, 431-439.

[4] L. D Kou, G. Markowsky and L. Berman, *A fast algorithm for Steiner trees,* Acta Informatica **15** (1981), pp. 141-145.

[5] K. Mehlhorn, *A faster approximation algorithm for the Steiner problem in graphs*, Information Processing Letters **27(3)** (1988), pp. 125-128.

[6] H. Takahashi and A. Matsuyama, *An approximate solution for the Steiner problem in graphs*, Math. Japonica **24** (1980), pp. 573-577.

[7] A. Z. Zelikovsky, *The 11/6-approximation algorithm for the Steiner problem on networks,* Algorithmica **9** (1993), pp. 463-470.