Geometric Steiner Trees

From the book: "Optimal Interconnection Trees in the Plane"

By Marcus Brazil and Martin Zachariasen

Part 2: Global properties of Euclidean Steiner Trees and GeoSteiner

Marcus Brazil

2015

Part 2: Global properties of Euclidean Steiner Trees and GeoSteiner

1. Structural properties of minimum Steiner trees
2. The GeoSteiner Algorithm — Overview
3. The GeoSteiner Algorithm — Generation Phase
4. The GeoSteiner Algorithm — Concaternation Phase

Structural properties

Here we study some necessary geometric and combinatorial conditions that must be satisfied by minimum Steiner trees for a given terminal set N, independently of the topology of the tree. These conditions can be checked rapidly, and hence can be used as efficient pruning conditions for eliminating non-feasible topologies, or families of topologies, when attempting to construct a minimum Steiner tree.

The properties fall into two types:

- An *empty region* is a region in the plane that can be shown to be free of Steiner points and/or terminals if certain conditions are fulfilled.
- An *edge-length bound* provides a lower and/or an upper bound on the lengths of certain edges in a minimum Steiner tree.

We are interested in identifying empty regions and edge-length bounds that can be efficiently computed without having to first compute a minimum Steiner tree.

Empty regions — the wedge property

Define a *wedge* to be any translation of a convex cone in the plane.

Lemma 6

Let W be an open wedge having angle $2\pi/3$ or more and containing no elements of N. Then W contains no Steiner point of any minimum Steiner tree for N.

Proof: Suppose W contains at least one Steiner point s. Without loss of generality, we introduce a Cartesian coordinate system (x, y) with positive x-axis along the angle bisector of the wedge, and choose s to be a Steiner point in W with the largest x-coordinate. Of the three incident edges at s, one leaves s in a direction within $\pm\pi/3$ of the positive x-axis. This edge cannot leave W (and so cannot end at a terminal) and its other endpoint has a larger x-coordinate than that of s, giving a contradiction.

Empty regions — the Steiner hull

Definition: Steiner hull

Given any set of terminals N, a *Steiner hull* for N is a bounded region of the plane containing every minimum Steiner tree for N.

A Steiner hull can be found by taking the complement of a union of wedges satisfying the wedge property. For example, it follows that the *convex hull* of N is a Steiner hull of N.

Let P_N denote a polygon with a subset of the terminal set $N = \{t_1, \ldots, t_n\}$ as its vertices, and such that P_N is the boundary of a Steiner hull for N. At least one such Steiner hull exists, namely the convex hull of N.

We can potentially improve on this Steiner hull by performing a series of legal replacements.

Empty regions — the Steiner hull

Let $t_i t_j$ be an edge of P_N. A replacement of $t_i t_j$ by a pair of edges $t_i t_k$ and $t_k t_j$, $t_k \in N \setminus \{t_i, t_j\}$, denoted by $t_i t_j \to t_i t_k t_j$, is said to be *legal* if

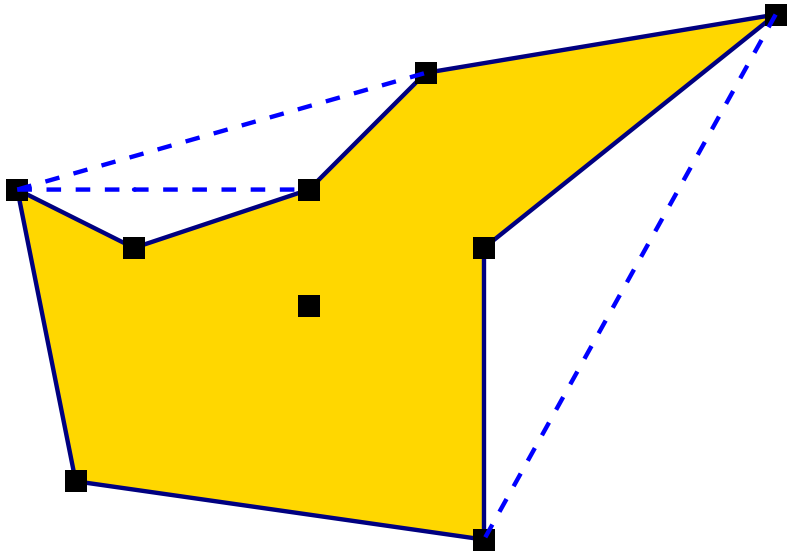
- $\triangle t_i t_k t_j$ is contained in P_N;
- $\triangle t_i t_k t_j$ contains no terminals other than its vertices; and
- $\angle t_i t_k t_j \geq 2\pi/3$.

Lemma 7

Let P_N be the boundary of a Steiner hull for N. If there is a legal replacement $t_i t_j \to t_i t_k t_j$ of the edge $t_i t_j$ of P_N, then the reduced polygon is also the boundary of a Steiner hull for N.

This gives a recursive procedure for finding a minimum area Steiner hull with terminals as vertices.

Empty regions — the Steiner hull

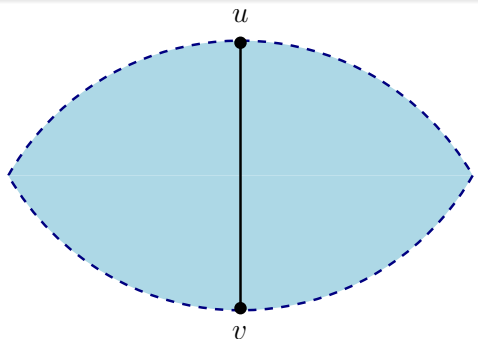


The replacements can be done in any order; every maximal sequence of legal replacements results in the same final P_N. There is a $\Theta(n \log n)$ time algorithm for constructing this minimal Steiner hull,

Empty regions — the lune property

Definition: Steiner hull

Given a line segment uv, we define the *lune* of uv, $\mathcal{L}(u, v)$, to be the intersection of open discs $D(u)$ and $D(v)$ centred at the points u and v, respectively, and each of radius $|uv|$. Equivalently, $\mathcal{L}(u, v)$ is the region consisting of all points x such that $|ux| < |uv|$ and $|vx| < |uv|$.



Lemma 8

If uv is an edge of a minimum Steiner tree T, then the lune $\mathcal{L}(u, v)$ does not contain any points of the tree T other than the interior of uv.

Bottleneck Steiner distance bound

Assume that $t_i, t_j \in N$ is a pair of distinct terminals. Let $P_{\bar{T}}(t_i, t_j)$ denote the unique path between t_i and t_j in some minimum spanning tree (MST) $\bar{T}$ for N.

Definition: Bottleneck Steiner distance

Given two terminals t_i and t_j in N, the *bottleneck Steiner distance*, $\text{BSD}(t_i, t_j)$, is equal to the length of the longest edge on $P_{\bar{T}}(t_i, t_j)$ for some MST $\bar{T}$ of N.

We note that $\text{BSD}(t_i, t_j)$ is well defined, since any MST results in the same bottleneck Steiner distances. Let $P_T(t_i, t_j)$ denote the unique path between t_i and t_j in a minimum Steiner tree T.

Lemma 9

Let t_i, t_j be two terminals of a minimum Steiner tree T. For any edge $e \in P_T(t_i, t_j)$, we have $|e| \leq \text{BSD}(t_i, t_j)$.

Bottleneck Steiner distance bound - Proof of Lemma 9

Assume that there exists an edge $e \in P_T(t_i, t_j)$ such that $|e| > \mathrm{BSD}(t_i, t_j)$. Remove e from T, and consider the two connected components (subtrees) T_1 and T_2.
Choose any minimum spanning tree $\bar{T}$ and consider the path $P_{\bar{T}}(t_i, t_j)$. The longest edge on this path has length $\mathrm{BSD}(t_i, t_j)$. Hence the first edge f on the $P_{\bar{T}}(t_i, t_j)$ that connects a terminal in T_1 with a terminal in T_2 has length $|f| \leq \mathrm{BSD}(t_i, t_j)$.
By reconnecting T_1 and T_2 using edge f, a shorter tree interconnecting N is obtained — a contradiction to the minimality of T. □

The necessary condition provided by Lemma 9 is very powerful in practice. Note that the bottleneck Steiner distance bound is independent of the norm; the above proof is not reliant on the geometry of the Euclidean plane.

The GeoSteiner Algorithm

As we will see later in the course, the Euclidean Steiner tree problem is an *NP-hard* problem, meaning that there (almost certainly) does not exist a polynomial time algorithm for solving it. But this does not preclude the possibility of developing efficient exact algorithms for solving real-world problem instances. NP-hardness is a measure of *worst-case performance*, and such difficult instances may almost never occur in practice.

The *GeoSteiner* algorithm is by far the most efficient exact algorithm for computing a minimum Steiner tree.
The running time of GeoSteiner shows good average behaviour and scaling in practice.

Top level Algorithm

A naïve algorithm for computing a minimum Steiner tree is as follows:

1. enumerate all full Steiner topologies for every subset of N (the set of terminals);
2. apply the Melzak-Hwang algorithm to compute a full Steiner tree (FST) for each such full Steiner topology;
3. obtain a minimum Steiner tree by identifying a subset among the constructed FSTs that interconnects N and has minimum length.

However, since the number of terminal subsets is exponential in N — and the number of full Steiner topologies for each subset is super-exponential — this algorithm would exhibit very bad scaling. It is infeasible in practice to compute minimum Steiner trees with more than, say, 15 terminals using this naïve algorithm.

Top level Algorithm

Almost all of the work in the naïve algorithm is wasted, as:

- most of the full Steiner topologies have no associated FST; and
- most of the constructed FSTs are not shortest networks for their terminals.

This is addressed by the *GeoSteiner algorithm* which reduces the work of the naïve algorithm significantly by implicit instead of explicit enumeration of FSTs.

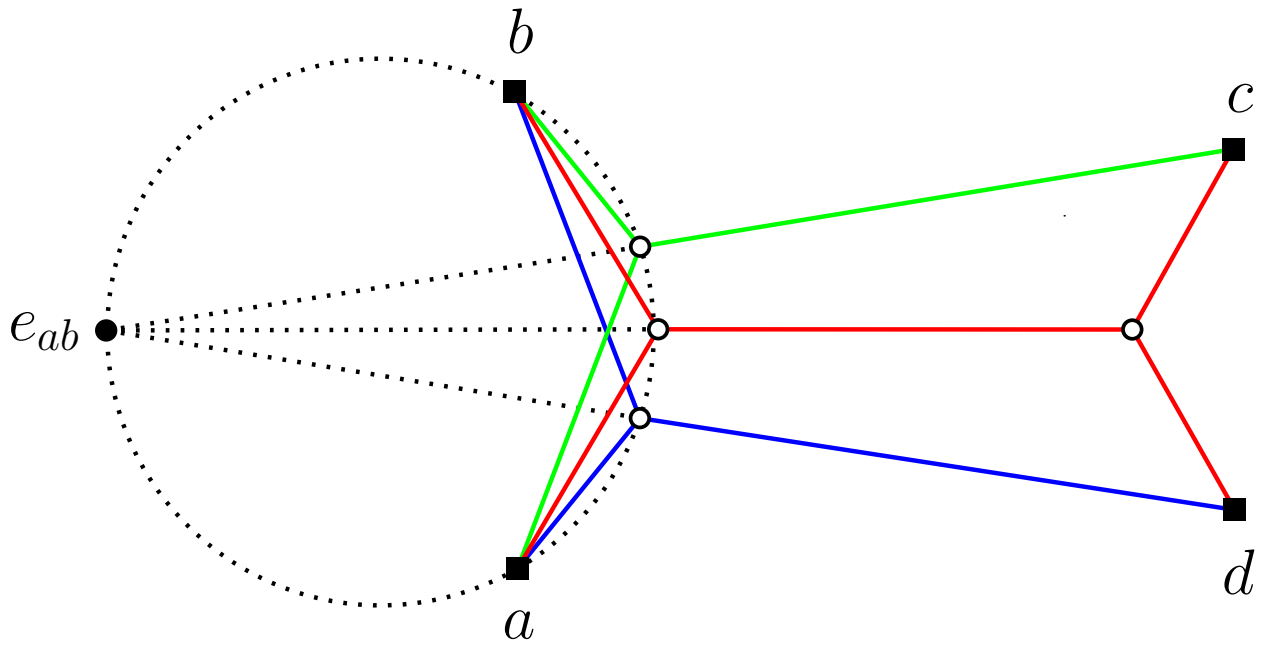
GeoSteiner algorithm

Input: Set of points (terminals) N in the plane.
Output: A minimum Steiner tree for N.

Generation Phase Construct a sufficient set of FSTs $\mathcal{F} = \{T_1, T_2, \ldots, T_m\}$ by efficient enumeration

Concatenation Phase Identify a subset $\mathcal{F}^* \subseteq \mathcal{F}$ such that $\mathcal{F}^*$ interconnects N and has minimum total length.

The FST generation phase — Core concepts

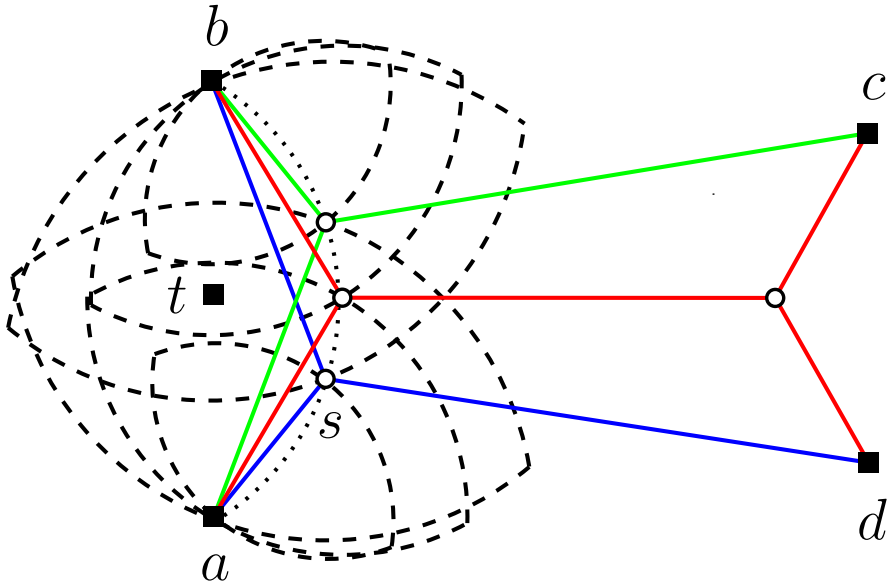


Consider the three FSTs in the figure. The terminal pair $\{a, b\}$ forms a cherry in each of these FSTs, and the Steiner points are located on a common Steiner arc $\widehat{ab}$. Here the three FSTs share a sub-topology (the cherry $\{a, b\}$) and the same equilateral point e_{ab}.

Instead of enumerating FSTs directly, GeoSteiner enumerates equilateral points with associated sub-topologies.

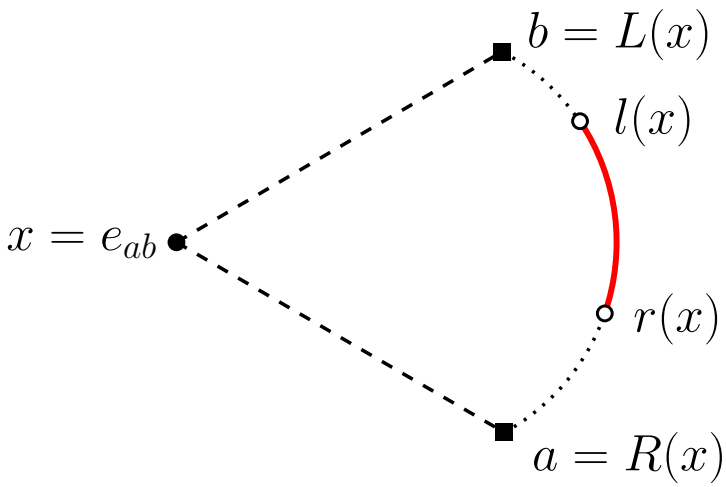
The FST generation phase — Core concepts

Equilateral points fix a part of the geometry of the associated FSTs, making it possible to check some structural properties early in the enumeration algorithm.



For example, suppose there is a fifth terminal t halfway between a and b. No matter where s is located on $\widehat{ab}$, terminal t would be inside one of the two lunes defined by edges as and bs. Hence, none of the three FSTs shown in the figure would be generated.

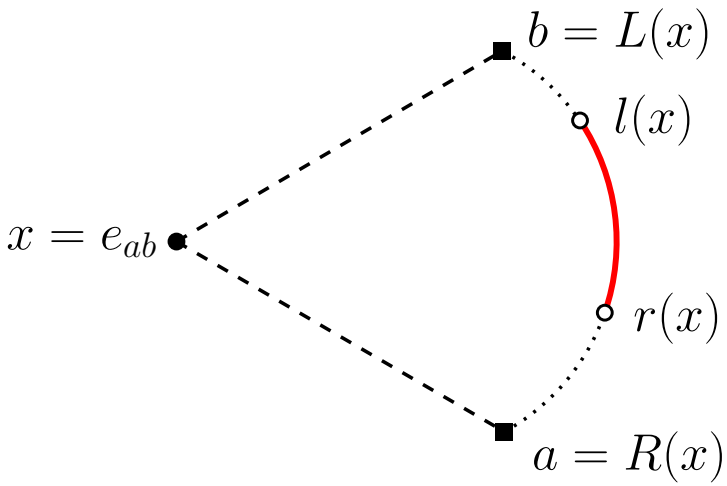
Enumerating equilateral points



The FST generation algorithm simulates the Melzak-Hwang algorithm but for multiple topologies at a time.

Consider an equilateral point $x = e_{ab}$. Define $R(x) = a$ and $L(x) = b$ to be the *base points* of x. The equilateral point x forms the root of a binary tree in which the children of an equilateral point are its base points — and where terminals are leaves.
The *order* of x is the depth of the associated binary tree.

Feasible subarcs



The Steiner point for a given equilateral point x must be located somewhere on the Steiner arc $\widehat{R(x)L(x)}$ of x. GeoSteiner maintains a *feasible subarc* $\widehat{r(x)l(x)}$ for x: a feasible Steiner point can only be located on the interior or this subarc.

Initially $r(x) = R(x)$ and $l(x) = L(x)$; by employing a number of pruning tests, the subarc is iteratively reduced — and possibly eliminated completely, in which case the equilateral point x can be removed altogether.

Branches and branch trees

More generally, we have the following definitions.

Definitions: Branch trees, branches

A *branch tree* B is a tree spanning a set of terminals N_B, and containing a ray (the *stem*) emanating from one of the vertices v of the tree (known as the *root*), such that for any point $a (\neq v)$ on the stem the union of av with all edges of B other than the stem is an FST for $\{a\} \cup N_B$. A *branch* $\mathcal{B}$ is a (possibly infinite) set of branch trees, each of which spans a common set of terminals $N(\mathcal{B})$, such that every branch tree $B \in \mathcal{B}$ has the same topology.

We define the *size* of a branch $\mathcal{B}$ (or branch tree B or FST) to be the number of terminals it spans.

GeoSteiner FST generation algorithm

The generation phase of GeoSteiner enumerates branches and FSTs of increasing size — essentially using a dynamic programming approach.

We first require the following lemma which gives us a small sufficient set of FSTs of size 2, namely the edges from a single minimum spanning tree (MST).

Lemma 10: MST edges in minimum Steiner tree

Let $\bar{T}$ be an MST for N. Then there exists a minimum Steiner tree T for N such that all terminal-terminal connections (or size 2 FSTs) in T are edges from $\bar{T}$.

The proof is similar to the proof of Lemma 9, and is left as an exercise.

GeoSteiner FST generation algorithm

Input: Terminal set $N = \{t_1, t_2, \ldots, t_n\}$.
Output: A sufficient set of FSTs $\mathcal{F} = \{T_1, T_2, \ldots, T_m\}$.

1. Let $\mathcal{F}$ be the $n-1$ edges of a minimum spanning tree for N. Let Γ_1 be the branches of size 1 (i.e., N).
2. **For** $k = 2$ to n **do** [Generate branches and FSTs of size k]
3. Let $\Gamma_k = \emptyset$;
4. **For** $l = 1$ to $\lfloor k/2 \rfloor$ and **foreach** $\mathcal{B}_1 \in \Gamma_l$ and $\mathcal{B}_2 \in \Gamma_{k-l}$ **do**
5. **If** $k < n$ **then**
6. Construct a new branch $\mathcal{B}$ from $\mathcal{B}_1$ and $\mathcal{B}_2$;
7. **If** $\mathcal{B}$ is feasible and passes pruning tests **then** add $\mathcal{B}$ to Γ_k
8. **If** $k > 2$ **then**
9. Construct a new FST T from $\mathcal{B}_1$ and $\mathcal{B}_2$
10. **If** T is feasible and passes pruning tests **then** add T to $\mathcal{F}$.

GeoSteiner FST generation algorithm

Full Steiner trees (FSTs) are constructed by combining two equilateral points/branches (lines 8-10 in thealgorithm). More specifically, a valid FST T is obtained if the Simpson line between the two equilateral points x_1 and x_2 (corresponding to branches $\mathcal{B}_1$ and $\mathcal{B}_2$, respectively) properly intersect their feasible subarcs.

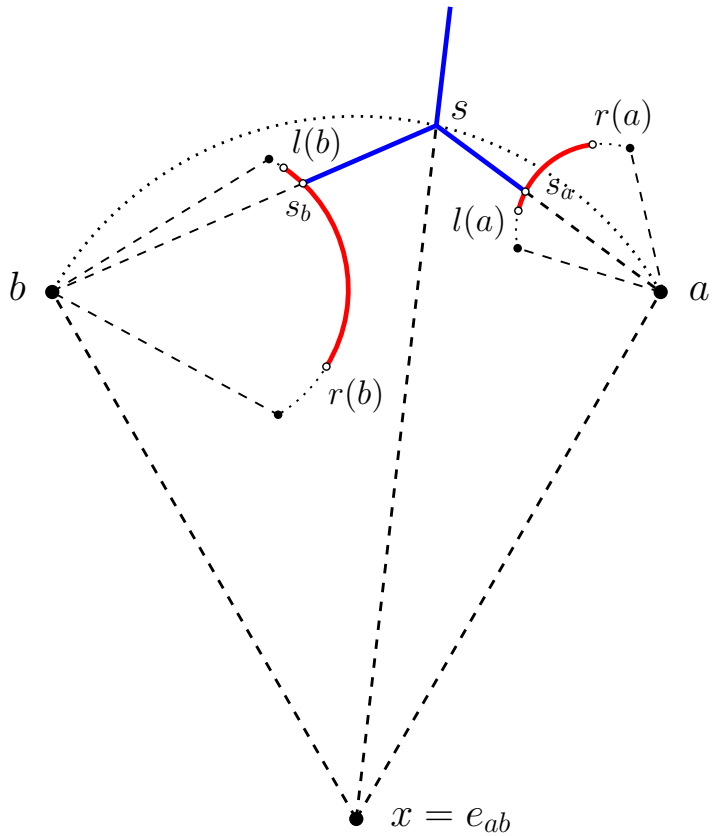
This, however, results in multiple generations of T — one for each edge of T. We may therefore arbitrarily select the edge of T that is incident to the terminal spanned by T that has the *highest index* in N. Thus, line 9 in the algorithm needs to first test if x_1 is a terminal t that has a higher index than all terminals in $N(x_2)$.

We now consider some of the pruning tests required for lines 7 and 10 of the algorithm.

Pruning Tests - Projections

Pruning tests for branches are based on reducing the feasible subarc of equilateral points. We begin with *projection tests* that basically use the fact that edges meet at angles of $2\pi/3$ at Steiner points.

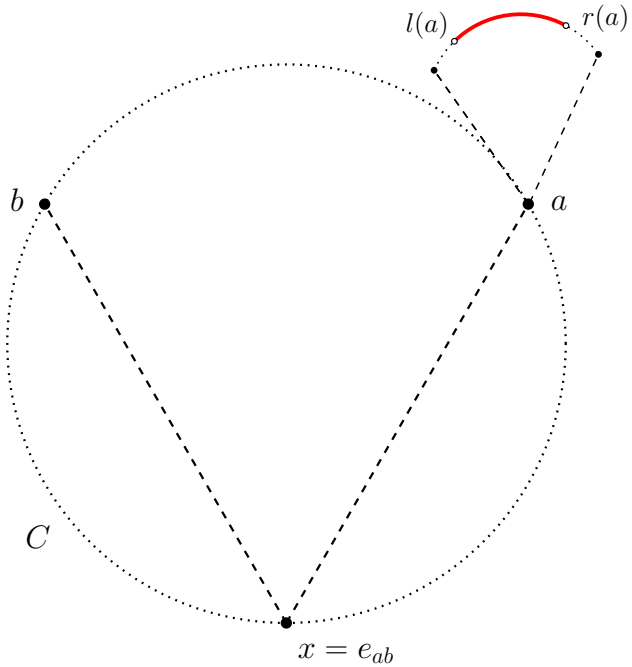
Consider an equilateral point $x = e_{ab}$ with base points a and b. For x to exist, there must be a Steiner point s on $\widehat{ab}$ such that the ray from a through s intersects the feasible subarc $\widehat{r(a)l(a)}$ in a point s_a strictly between a and s; similarly for b. Note that $s_a s$ and $s_b s$ are possible edges in an FST.

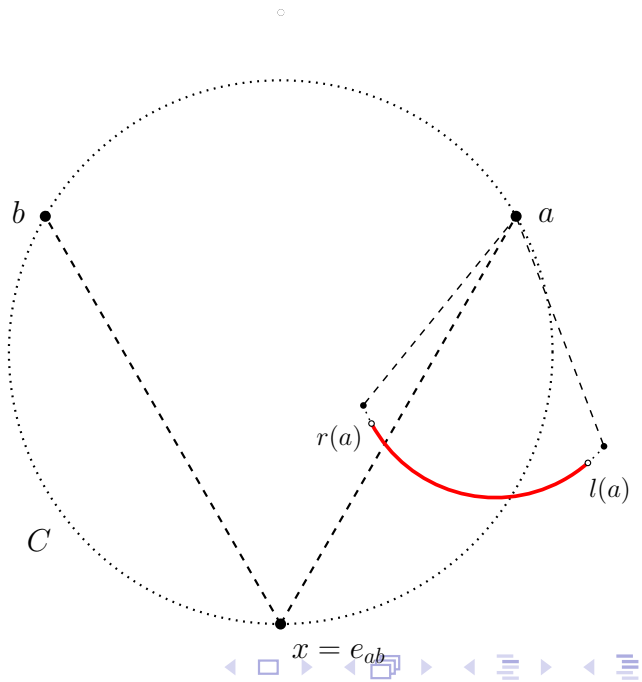


Pruning Tests - Projections

The Steiner point s can be viewed as a *projection* of s_a (or s_b) onto Steiner arc $\overset{\frown}{ab}$. The set of feasible projections is the intersection of the two projection sets for s_a ands_b. If this intersection set is empty, the equilateral point x can be pruned.

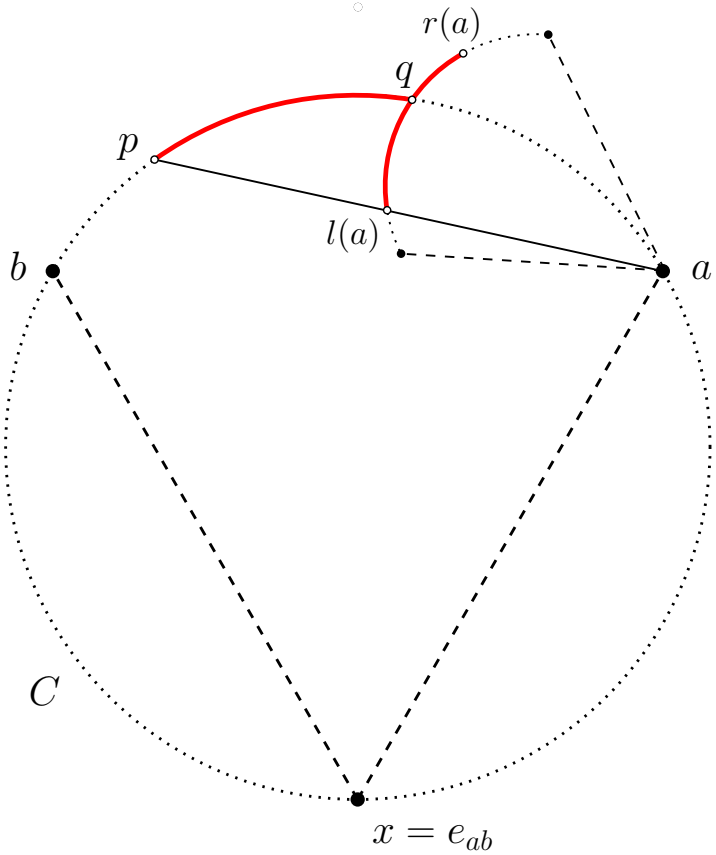
Let C be the circle circumscribing a, b and x. There are 3 cases for a.
Case 1: $l(a)$ is outside C. The equilateral point x can be pruned.

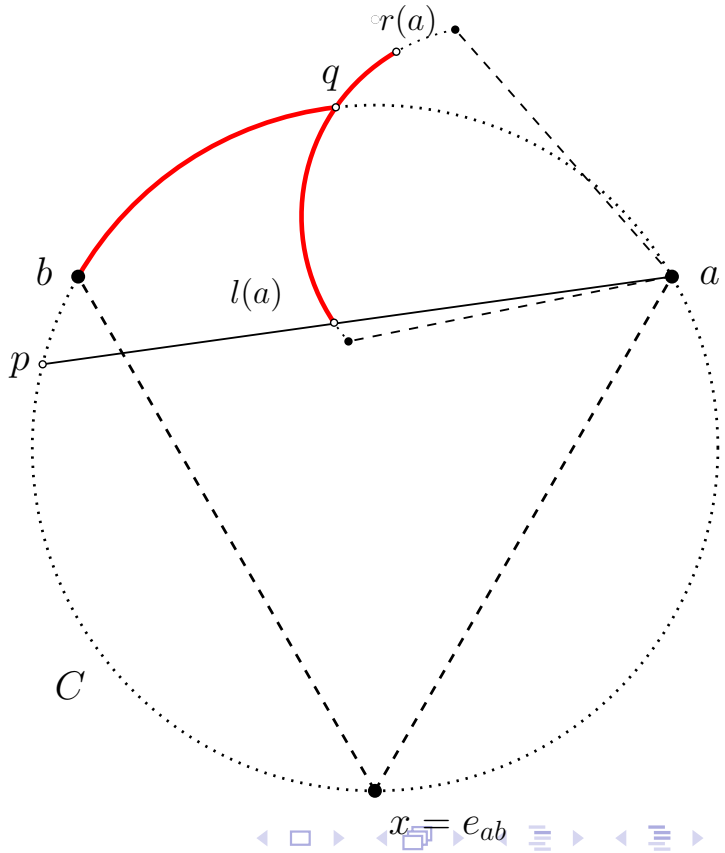




Pruning Tests - Projections

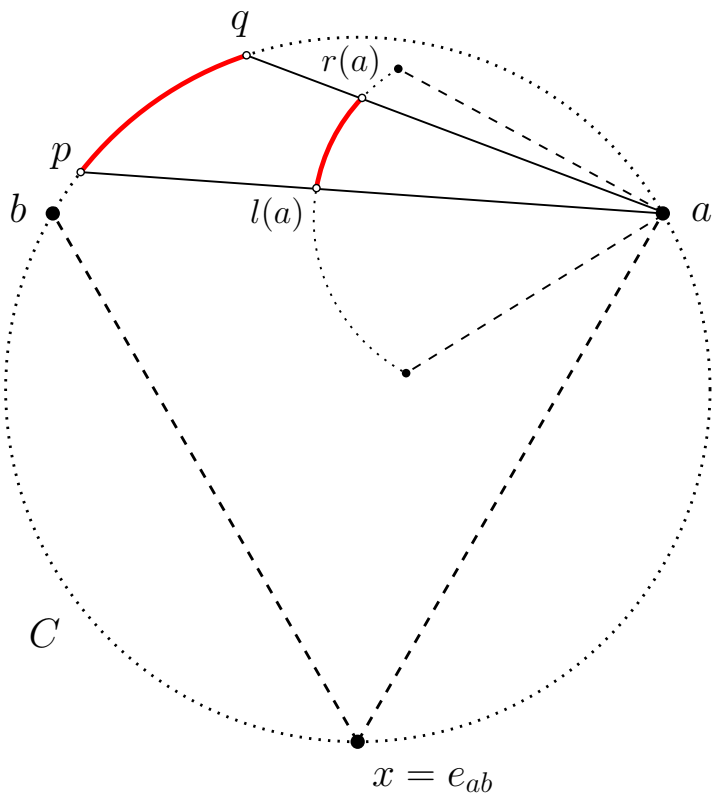
Case 2: $r(a)$ is outside C, $l(a)$ is inside C. Let p be the projection of $l(a)$ onto C, and let q be the intersection of $\widehat{r(a)l(a)}$ with $\widehat{ab}$ (if q does not exist, x can be pruned). If p is on $\widehat{ab}$, then $\widehat{pq}$ forms a feasible subarc for x; otherwise, $\widehat{bq}$ forms a feasible subarc for x.

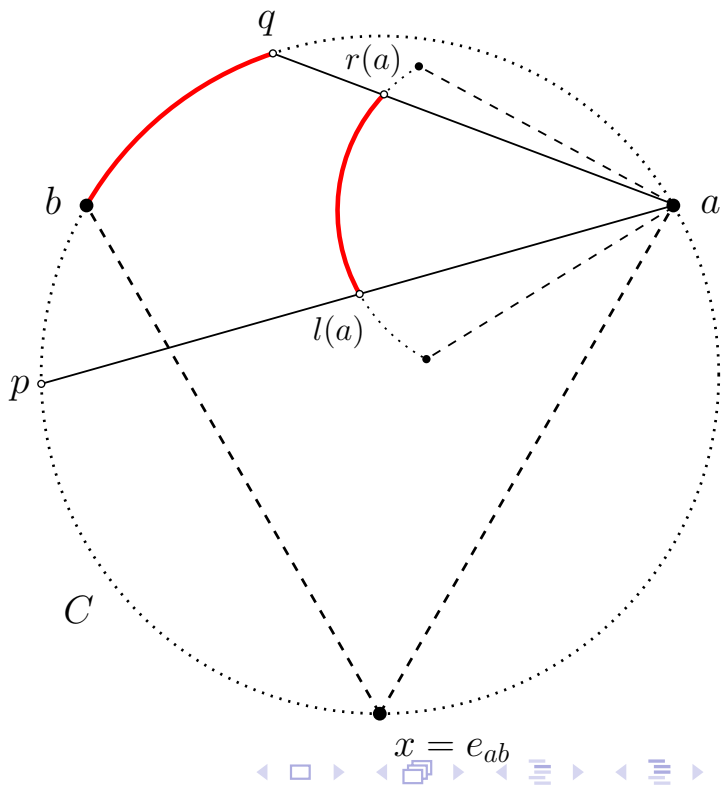




Pruning Tests - Projections

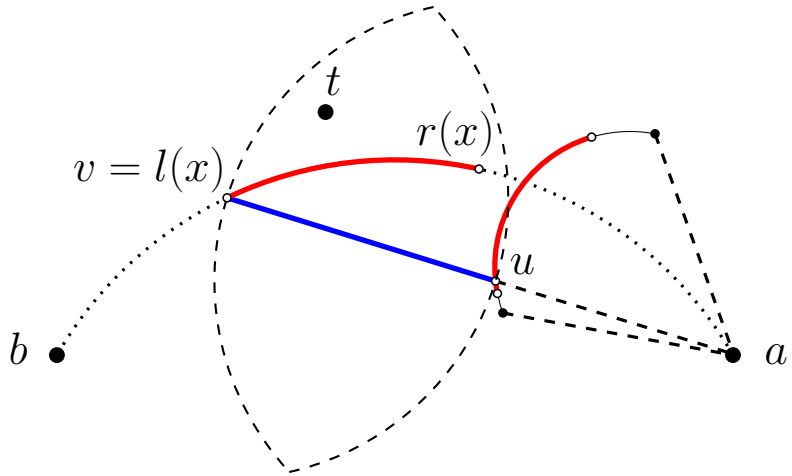
Case 3: Both $r(a)$ and $l(a)$ are inside C. Let p be the projection of $l(a)$ onto C, and let q be the projection of $r(a)$ onto C. If both p and q are on $\widehat{ab}$, then $\widehat{pq}$ forms a feasible subarc for x. If only q is on $\widehat{ab}$, then $\widehat{bq}$ forms a feasible subarc for x. In all other cases, x can be pruned.

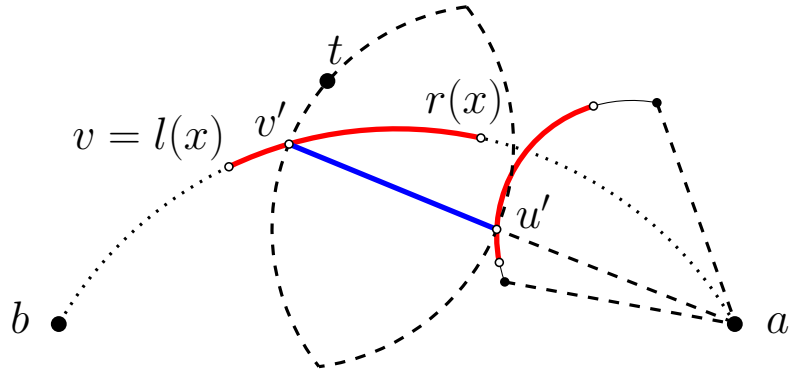




Pruning Tests - Empty lunes

Assume that a feasible subarc $\widehat{l(x)r(x)}$ for the equilateral point $x = e_{ab}$ has been identified using the projection test. If a is of non-zero order, let u be the point on $\widehat{l(a)r(a)}$ that is projected onto point $v = l(x)$ (otherwise $u = a$).





If there is a terminal t inside $\mathcal{L}(u, v)$ we move $v = l(x)$ toward $r(x)$ on $\widehat{ab}$ until the corresponding lune does not contain t. If v moves all the way to $r(x)$, then x can be pruned. We repeat this for b.

Pruning Tests - Bottleneck Steiner distance bound

Consider the equilateral point $x = e_{ab}$; let $N(a)$ and $N(b)$ be the terminals involved in the construction of a and b, respectively. A Steiner point s on $\widehat{ab}$ is the root of a branch tree B_s with two children (also branch trees) that interconnect the terminals in $N(a)$ and $N(b)$, respectively. Any path in B_s between terminals $t_a \in N(a)$ and $t_b \in N(b)$ includes the edges $s_a s$ and $s_b s$.

The bottleneck Steiner distance $\mathrm{BSD}(t_a, t_b)$ bounds the length of each edge on a Steiner tree path between t_a and t_b (Lemma 9). Let

$$B = \min_{t_a \in N(a),\ t_b \in N(b)} \mathrm{BSD}(t_a, t_b)$$

be the minimum pairwise bottleneck Steiner distance between a terminal in $N(a)$ and a terminal in $N(b)$. Then $|s_a s| \leq B$ and $|s_b s| \leq B$.

This edge-length bound results in a powerful pruning test, that can be implemented in a similar manner to the empty lune test.

Pruning Tests - Final remarks

Each of these three types of tests can be performed using elementary geometric constructions and computations that do not involve trigonometric functions. The running times are as follows:

Projections Constant time for a given equilateral point.

Empty lune Constant time for each move of $l(x)$ or $r(x)$, plus the time needed to identify a terminal inside the corresponding lune (if any).

Bottleneck Steiner distance Constant time, plus the time needed to compute B (which depends on the data structure used).

There are other possible pruning test (for example those based on upper bounds using efficient heuristics solutions to the Steiner problem) however these can be quite time consuming, so it is important to find the right balance between running time and pruning efficiency.

Performance of FST generation algorithm

The practical performance of the GeoSteiner FST generation algorithm for the Euclidean Steiner tree problem has been carefully studied. Using the current version of the code on a modern computer, the FSTs for 1000 uniformly distributed terminals can be generated in less than 10 seconds. This amazing performance comes in particular from an efficient implementation of the lune property and bottleneck Steiner distance tests.

For $n = 1000$ uniformly distributed terminals, in total around 60000 equilateral points are generated, the largest having around 10–15 terminals, and around 2500 FSTs are generated. For $n = 10000$, the running time on a modern computer is less than 500 seconds; the number of surviving FSTs is 26000.

The running time for uniformly distributed terminals is less than quadratic in practice.

The GeoSteiner Algorithm — Concaternation Phase

The output of the FST generation phase is a set of FSTs $\mathcal{F} = \{T_1, T_2, \ldots, T_m\}$ that is guaranteed to contain the full components of at least one minimum Steiner tree for N. In the FST concatenation problem, we need to identify a subset $\mathcal{F}^* \subseteq \mathcal{F}$ such that $\mathcal{F}^*$ interconnects N and has minimum total length.

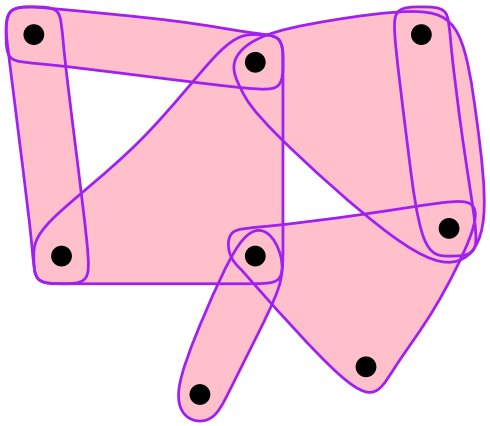
This problem can be solved in numerous ways. Here we briefly look at how it can be solved as a *minimum spanning tree problem in hypergraphs*.

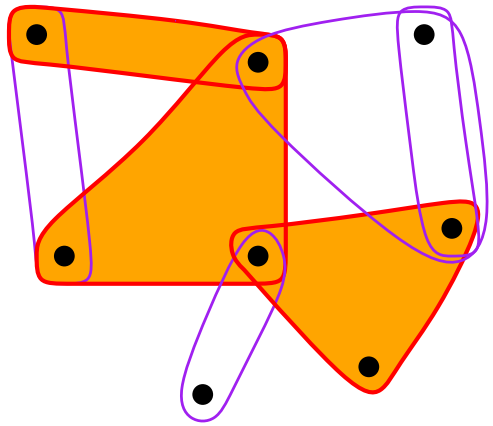
Spanning trees in hypergraphs

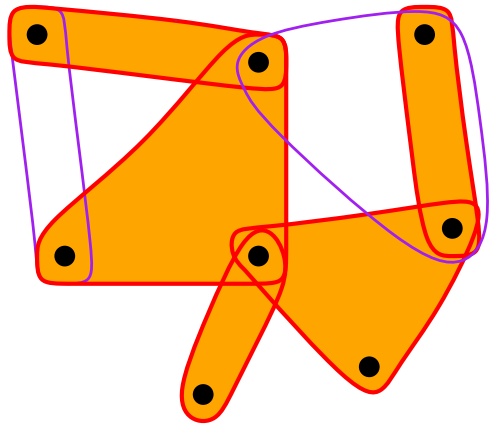
Definitions: Hyperedge, hypergraph, chain

Given a finite set V, a *hyperedge* $\mathbf{e} \subseteq V$ is a subset of V of cardinality $|\mathbf{e}| \geq 2$. A *hypergraph* $\mathbf{G} = (V, \mathbf{E})$ consists of a set of vertices V and a set $\mathbf{E}$ of hyperedges of V.

A *chain* in a hypergraph $\mathbf{G} = (V, \mathbf{E})$ from $v_0 \in V$ to $v_k \in V$ is an alternating sequence of vertices and hyperedges $v_0, \mathbf{e}_1, v_1, \mathbf{e}_2, v_2, \ldots, \mathbf{e}_k, v_k$ such that all vertices and hyperedges are *distinct* and $v_{i-1}, v_i \in \mathbf{e}_i$ for $i = 1, 2, \ldots, k$.







Spanning trees in hypergraphs

Definitions: Spanning tree in hypergraph

A *spanning tree* $\mathbf{T} = (V(\mathbf{T}), \mathbf{E}(\mathbf{T}))$ in a hypergraph $\mathbf{G} = (V, \mathbf{E})$ is a set of hyperedges $\mathbf{E}(\mathbf{T}) \subseteq \mathbf{E}$ spanning V such that there is a *unique* chain using vertices and hyperedges from $\mathbf{E}(\mathbf{T})$ between *every pair* of vertices $u, v \in V$.

A spanning tree $\mathbf{T}$ in a hypergraph has the same key properties as as in an ordinary graph: any two distinct hyperedges in $\mathbf{T}$ contain at most one common vertex; and there is a unique path in $\mathbf{T}$ between every pair of vertices.

MINIMUM SPANNING TREE PROBLEM IN HYPERGRAPHS

Given: An edge-weighted hypergraph $\mathbf{G} = (V, \mathbf{E})$, where $c(\mathbf{e}) > 0$ denotes the weight of each hyperedge $\mathbf{e} \in \mathbf{E}$..
Find: A spanning tree $\mathbf{T} = (V, \mathbf{E}(\mathbf{T}))$ in $\mathbf{G}$ such that $\sum_{\mathbf{e} \in \mathbf{E}(\mathbf{T})} c(\mathbf{e})$ is minimised.

Spanning trees in hypergraphs

The main motivation for studying the minimum spanning tree problem in hypergraphs is that it can be used to solve the full Steiner tree (FST) concatenation problem. Recall that the output of the FST generation phase is a set of FSTs $\mathcal{F} = \{T_1, T_2, \ldots, T_m\}$ that is guaranteed to contain the full components of at least one minimum Steiner tree for terminal set N. In the FST concatenation problem, we need to identify a subset $\mathcal{F}^* \subseteq \mathcal{F}$ such that $\mathcal{F}^*$ interconnects N and has minimum total length.

Clearly, the FST concatenation problem can be formulated as a minimum spanning tree problem in a hypergraph, where the terminal set N forms the vertex set; for each FST T_i the associated terminal subset $N(T_i) \subseteq N$ is a hyperedge with weight equal to the geometric length $|T_i|$ of the FST.

The best method, to date, for solving the hypergraph spanning tree problem in this context is using *integer programming*.

Integer programming formulation

A minimum spanning tree is represented as an incidence vector x, where $x_{\mathbf{e}} = 1$ if edge $\mathbf{e} \in \mathbf{E}$ is part of the minimum spanning tree, and otherwise $x_{\mathbf{e}} = 0$. We denote by $|\mathbf{e}|$ the number of vertices spanned by hyperedge $\mathbf{e} \in \mathbf{E}$.

$$\text{minimise} \quad \sum_{\mathbf{e} \in \mathbf{E}} c(\mathbf{e}) x_{\mathbf{e}} \qquad \boxed{IP_{\text{hmspt}}} \qquad (1)$$

$$\text{subject to} \quad \sum_{\mathbf{e} \in \mathbf{E}} (|\mathbf{e}| - 1) x_{\mathbf{e}} = |V| - 1 \qquad (2)$$

$$\sum_{\substack{\mathbf{e} \in \mathbf{E}: \\ \mathbf{e} \cap W \neq \emptyset}} (|\mathbf{e} \cap W| - 1) x_{\mathbf{e}} \leq |W| - 1, \quad \emptyset \neq W \subset V \qquad (3)$$

$$x_{\mathbf{e}} \in \{0, 1\}, \qquad \mathbf{e} \in \mathbf{E}. \qquad (4)$$

Integer programming formulation

The objective (1) is to minimise the total weight of the chosen hyperedges subject to the following constraints:

- Equation (2) enforces the correct number and cardinality of hyperedges to construct a spanning tree.
- Constraints (3) eliminate cycles by extending the standard notion of cycle elimination in graphs. For a given non-empty subset $W \subset V$, the total number of 2-edges in the subset (viewing hyperedges as a set of 2-edges) can be at most $|W| - 1$, otherwise a cycle is created. The number of 2-edges contributed by a hyperedge $\mathbf{e} \in \mathbf{E}$ that intersects W is $|\mathbf{e} \cap W| - 1$.

Although IP_{hmspt} is an NP-hard problem it can in practice be solved efficiently via *branch-and-cut* methods. This formulation was one of the great breakthroughs in speeding up GeoSteiner.