A Fast Algorithm for Rectilinear Steiner Trees with Length Restrictions on Obstacles

Stephan Held[1] and Sophie Theresa Spirkl[2]

[1] Research Institute for Discrete Mathematics, Bonn, Germany
held@or.uni-bonn.de
[2] Princeton University, Princeton NJ 08544, USA
sspirkl@princeton.edu

Abstract. We study the minimum rectilinear Steiner tree problem in the presence of obstacles. Traversing obstacles is not strictly forbidden, but the total length of each connected component in the intersection of the tree with the interior of the blocked area is bounded by a constant.
This problem is motivated by the layout of repeater tree topologies, a central task in chip design. Large blockages might be crossed by wires on higher layers, but repeaters may not be placed within the blocked area. A too long unbuffered piece of interconnect would lead to timing violations.
We present a 2-approximation algorithm with a worst case running time of $\mathcal{O}((k \log k)^2)$, where k is the number of terminals plus the number of obstacle corner points. Under mild assumptions on the obstacle structure, as they are prevalent in chip design, the running time is $\mathcal{O}(k(\log k)^2)$. Compared to strictly obstacle-avoiding trees, the algorithm provides significantly shorter solutions. It solves real world instances with 783 352 terminals within 126 seconds, proving its practical applicability.
This work has previously been published at ISPD'14, Petaluma, CA, USA.

1 Introduction

The computation of Steiner topologies for buffering is a central task in VLSI design [2, 3, 8]. Buffering is often restricted by macro cells, where no repeaters can be placed. This holds particularly for top-level layout that is dominated by big macros leaving only small gaps for buffers.

The wires, on the other hand, may well reach over the macros on higher routing layers. However, if an unbuffered component of wire gets too long, it will lead to capacitances, slew, or delay violations. This motivates bounding the length of tree components on top of macros.

Given a finite set of rectilinear obstacles in the plane and a reach-length $L \geq 0$, a set of axis-parallel segments is called *reach-aware*, if the total length of every connected component of its intersection with the interior of the *blocked area* (the closed union of all obstacles) has length at most L. This definition of reach-awareness is motivated by the assumption that a repeater can be placed arbitrarily close to the boundary of the blocked area, but not in its interior. We call a point *blocked* if it is in the interior of the blocked area.

The *reach-aware Steiner tree problem* (RASTP) is also known as the *length restricted Steiner tree problem*, introduced by Müller-Hannemann and Peyer [16]. The input consists of a finite set $S \subset \mathbb{R}^2$ of terminals, a finite set O of rectangular obstacles such that no terminal is placed in the interior of the blocked area ($s \cap (\bigcup_{o \in O} o)^\circ = \emptyset$ for all $s \in S$), and a parameter $L \in \mathbb{R}_{\geq 0}$. The goal is to find a shortest reach-aware rectilinear Steiner tree connecting these terminals.

For $L = \infty$ or $O = \emptyset$, the RASTP becomes the rectilinear Steiner tree problem, thus it is NP-hard. For $L = 0$, it is known as the *obstacle-avoiding rectilinear Steiner tree problem* [7]. Figure 1 shows examples of a minimum obstacle-unaware tree ($L = \infty$) with a large blocked

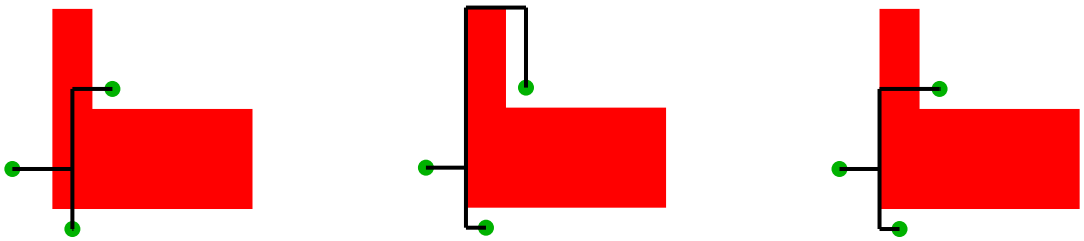


Fig. 1. Optimum solutions for different reach-lengths.

component, a minimum obstacle-avoiding tree ($L = 0$) taking a long detour, and a minimum reach-aware tree, where only the thin vertical obstacle can be passed by the choice of L.

There is extensive literature on fast algorithms for the case $L = 0$, which typically provide an approximation guarantee of two, the Steiner-ratio in graphs, [6, 12–14]. Our algorithm is based on the early work of Clarkson et al. [5], which introduced the notion of a visibility graph containing shortest paths between all pairs of terminals and obstacle corner points (for $L = 0$). Its size is bounded by $\mathcal{O}(k \log k)$ and it can be constructed in $\mathcal{O}(k(\log k)^2)$ time, where $k = |S| + |O|$. The currently most efficient exact algorithms for $L = 0$ can be found in [10].

For arbitrary L, [16] developed approximation algorithms based on an augmented Hanan grid. The size of this grid, bounded by $\mathcal{O}(k^2)$, is denoted by l. Using a minimum spanning tree heuristic, they present an $\mathcal{O}(l \log l)$ algorithm with an approximation ratio of 2. Under the additional assumption that all blockages are disjoint rectangles, they refine their construction further, obtaining approximation guarantees of $\frac{5}{4}\alpha$ and $\frac{2s}{2s-1}\alpha$ for $s \geq 4$ with graph size $\mathcal{O}(l)$ and $\mathcal{O}(l^{s-2})$, respectively, where α is the performance guarantee of a Steiner tree algorithm in graphs. However, their results are predominantly of theoretical interest, since even their 2-approximation algorithm would be very slow in practice due to the quadratic size of the Hanan grid.

Our problem is related to [9], who consider slew constraints instead of a reach-length L and provide an exact algorithm for this problem. In the RASTP, the reach-length can be used to bound the slew degradation on wires, though not as precisely as in [9]. However, we believe that it enables simpler and faster algorithms that are more suitable for a quick processing of the majority of nets.

A similar problem was also considered by [17], who incrementally update a rectilinear Steiner minimum tree to satisfy slew constraints. However, they do not give any performance guarantee for the approximation quality or running time.

Our algorithm produces short trees, but does not balance source-sink path lengths within the tree. However, in combination with the shallow light Steiner arborescence algorithm in [8] it can easily be extended to provide not only short but also fast reach-aware Steiner trees.

Note that for our definition, the restriction due to obstacles depends only on the blocked area and is independent of its representation by a union of rectangles. It differs from the one used in many recent publications on obstacle-avoiding rectilinear Steiner trees even for $L = 0$. For example, [10, 9] consider line segments on which two (polygonal) obstacles touch as not blocked and specifically only consider rectangular blockages. This definition depends on the particular representation of the blockages by polygons or rectangles. As no buffers can be placed on a line between two obstacles, we consider our definition more appropriate for our application.

In Section 3, we will see that one of the standard benchmarks for obstacle-avoiding trees contains a terminal isolated by a ring of obstacles and becomes infeasible with our definition and $L = 0$ (see Figure 6), which was also noted by [17], whereas in [10, 9] this instance is

reported as feasible due to their different definition of the blocked area. It should be noted that the approach in [10, 9] and many other publications would be adjustable to our definition (and $L = 0$).

In addition, our definition allows to compress or modify the representation of obstacles for the needs of the algorithm or its subroutines. In the following, we can assume that all obstacles are given as rectangles with pairwise disjoint interior and unblocked rectangle corners, e.g. by covering the blocked area with rectangles of maximal width and representing it as their union.

Finally, we can naturally model rectilinear polygons with rectilinear holes, which are often left out of big macros to serve particularly as buffer positions.

The remainder of the paper is organized as follows. In Section 2 we will give a detailed description of our algorithm. The performance on standard benchmarks as well as on practical instances is demonstrated in Section 3, followed by a conclusion in Section 4.

2 Algorithm

Our algorithm has two main phases: First, we construct a shortest-path-preserving graph (*visibility graph*) for the set of *endpoints*, by which we denote the union of the terminals S and the corner points of obstacles. In this graph, we use a Dijkstra-Kruskal approach [13] to compute a Steiner tree for the terminal set S. Simple local search heuristics are used as a post-optimization step.

The approximation ratio of this algorithm is at most two, since the computed Steiner tree is as most as long as a minimum terminal spanning tree in the visibility graph.

2.1 Reach-Aware Visibility Graph

The concept of a visibility graph for dealing with paths among polygonal obstacles was first introduced by Clarkson et. al. [5] for the obstacle-avoiding case. It is based on the key idea that every shortest path can be modified (while preserving the length) in such a away that it consists of shortest paths from one endpoint (terminal or blockage corner point) to another and the interior of the bounding box of two consecutive endpoints intersects neither blockages nor terminals. We will prove similar results for the reach-aware case, and based on that, give a precise construction of our visibility graph in the next section.

The construction then ensures that between each pair of endpoints, there is a so called *median line*, a vertical line to which both endpoints are connected by a horizontal segment (if possible). If the bounding box is completely unblocked, this is always possible. Using the argument above, this construction is indeed shortest-path-preserving. Note that horizontal instead of vertical median lines could be used equivalently.

In the remainder of the paper, we will often use the notion of empty bounding boxes.

Definition 1 (Empty Bounding Box). *Given points $s, t \in \mathbb{R}^2$, their bounding box is called* empty *if it does not contain any endpoints except for (potentially) s or t.*

Note that empty bounding boxes may intersect blockages whose endpoints are located outside the box. The emptiness of a bounding box also depends on its boundary, thus it depends on the choice of its spanning corners s and t.

In the obstacle-avoiding case as in [5], it is always possible to insert a median line and connecting segments if the bounding box of two endpoints is empty (and such a connection does not cross blocked area). Here, this does not hold and we have to develop a more elaborate construction.

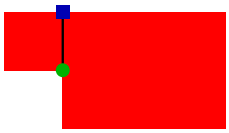
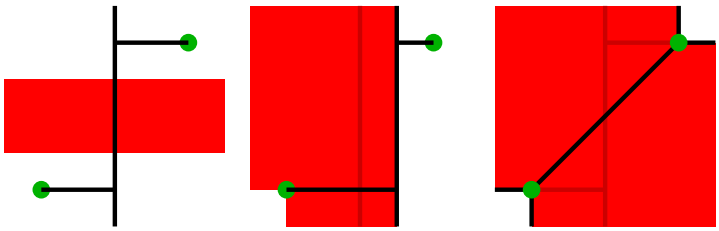


Fig. 2. Obstacle corner (green disk) and its vertical mirror point (blue square).



(a) Case 1 (b) Case 2 (c) Case 3

Fig. 3. Configurations in an empty bounding box.

First we extend the set of endpoints. As endpoints are connected to median lines via horizontal lines only, we provide vertical edges across blocked area in the following way.

If an obstacle corner is on the lower or upper boundary of the blocked area and the vertical connection across the blocked area to the next unblocked point has length bounded by L, we add that point and connect the two (see Figure 2). Analogously, if it is on left or right boundary of the blocked area, we add a horizontal connection and mirror point across the obstacle. In the following we call such a connection point a *mirror point*, and add all mirror points to the (extended) set of endpoints

$$\mathcal{E} := \{ \text{ endpoints and mirror points } \}.$$

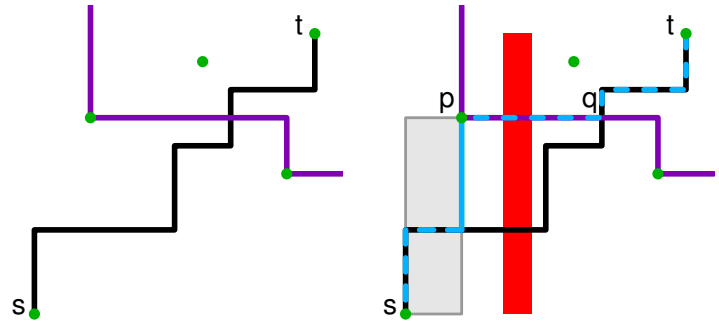
Each obstacle corner has at most one vertical and one horizontal mirror point, because it cannot be blocked from both above and below. The same is true for horizontal connections.

Now, assume that there is an ℓ_1-shortest reach-aware path between two points in $\mathcal{E}$ with empty bounding box (w.r.t. $\mathcal{E}$), then three cases can occur:

1. If the median line is unblocked at the y-coordinate of one endpoint, it must be unblocked at the y-coordinate of the other endpoint (by the insertion of the mirror points). We simply connect the endpoints to the median (Figure 3(a)).
2. If the median is blocked at the y-coordinates of the endpoints and the two spanning corners are not the only unblocked points in their (closed) bounding box, we must have a situation as in Figure 3(b). If the connections shown in Figure 3(b) are reach-aware, we insert them. Steiner points added along obstacle boundaries will be connected in a post-processing step.
3. If the entire bounding box except for the spanning corners in blocked, the endpoints will be connected diagonally if they can see each other, i.e. their ℓ_1-distance is less than L (Figure 3(c)).

The black lines in Figure 3 show the new construction, the shaded black lines are the median and connecting lines that would be inserted without blockages.

Note that mirror points are determined regardless of the Steiner tree and are added to $\mathcal{E}$, whereas Steiner points added on median lines such as in Figure 3(a) or obstacles boundaries as in Figure 3(b) are not considered endpoints.



(a) Staircase between s and t for given endpoints (green); P is drawn schematically.

(b) Construction of p. New path preserves ℓ_1-length.

Fig. 4. Subdivision of P for the proof of Lemma 1.

For the reach-aware case, we can now prove a result similar to Clarkson's about ℓ_1-shortest paths, i.e. paths whose length equals the ℓ_1-distance of their endpoints:

Lemma 1. *Given a set of rectilinear obstacles, a set of terminals in the plane, and a parameter L, any shortest path between two points w.r.t. the ℓ_1-norm that is reach-aware can be modified so that the bounding box of two consecutive endpoints on the path does not contain another endpoint. This modification preserves length and reach-awareness.*

Proof (of Lemma 1). Let s and t be two points with non-empty bounding box and P an ℓ_1-shortest s-t-path that is reach-aware; without loss of generality let $s = (0, 0)$ and $t = (x, y)$ with $x, y > 0$. Consider the set of endpoints in the bounding box of s and t. For these points, we define a staircase as the boundary of the (not necessarily closed) region of all $r \in \mathbb{R}^2_{\geq 0}$ such that the bounding box of s and r is empty (see Figure 4(a), staircase shown in purple).

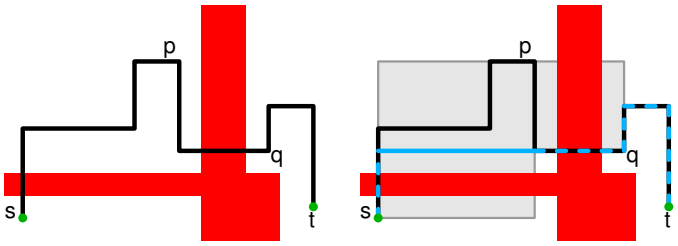
Let q be the first intersection point (starting from s) of P (black path) with this staircase as in Figure 4(b). Consider the bounding box of q and s. By the choice of q, there is at least one endpoint on its boundary. Among these let p be the one closest to s. Modify P to pass through this point (light blue path) and note that the length remains the same. Furthermore, the bounding box of p and s is empty.

P remains reach-aware, because blockages intersecting the bounding box of s and q have a very simple structure; in fact, – because there can be no corner in its interior – all of them have to intersect the entire bounding box either in a vertical (as shown) or horizontal rectangle whose endpoints lie on the outside. Either way, the blockages must have intersected P before, and the intersected length of the new path with the blockage is minimal; since P was reach-aware, so is the new path. Inductive application of this modification to the rest of P between p and t transforms P into the desired form in at most $\mathcal{O}(|S| + |O|)$ steps.

This can be generalized to reach-aware shortest paths:

Theorem 1. *Given a set of rectilinear obstacles, a set of terminals in the plane, and a parameter L, any path P between two endpoints that is reach-aware and shortest possible can be modified so that the bounding box of two consecutive endpoints on the path is empty, and P restricted to that bounding box is an ℓ_1-shortest path. This modification preserves length and reach-awareness.*

Proof ((Sketch)). We begin by applying Lemma 1 to all path segments of P that are ℓ_1-shortest paths. Now consider two consecutive endpoints s and t such that the path between them is not an ℓ_1-shortest path. We will show that P is not shortest possible. As a result, if we know



(a) s and t are consecutive endpoints. Their bounding box contains no reach-aware s-t-path.

(b) P can be shortened

Fig. 5. Configuration in the proof of Theorem 1.

that the path between two consecutive endpoints is always ℓ_1-shortest, the theorem follows from Lemma 1.

Let p be the last point on P from s to t such that the path from s to p is an ℓ_1-shortest path; let q be the last point or t (whichever comes first) from p to t on P such that pq is an ℓ_1-shortest path. Note that $p \neq t$ is not an endpoint, but $q = t$ is possible.

By our initial application of the previous lemma, we know that the bounding boxes of s and p and p and q are empty.

From the proof of Lemma 1, it is clear that between s and any point in the bounding box of s and p there is a reach-aware ℓ_1-shortest path, so q cannot be in that bounding box, because by choice of p and q, spq is not an ℓ_1-shortest path and replacing spq by sq would make P shorter and preserve reach-awareness, a contradiction.

If the bounding boxes of s and p and p and q intersect only in p, then sq is an ℓ_1-shortest path, which is a contradiction to our choice of p. Hence, the bounding boxes must intersect in a line segment as in Figure 5.

As in the previous lemma, since we know that the gray shaded bounding boxes are empty, any blockage intersected by the modified path must be intersected by spq for at least as long a segment. The new path (shown in light blue) is therefore reach-aware, and an ℓ_1-shortest s-q-path; it is strictly shorter than spq. Therefore, P is not shortest possible, which concludes the proof.

With this theorem, a construction similar to Clarkson et al. [5] can produce a reach-aware visibility graph. However, further caution is required in this case, because we must guarantee that any Steiner tree extracted from the visibility graph is reach-aware. This entails more than ensuring that any edge is reach-aware, because several edges may belong to the same connected component when intersected with the interior of the blocked area.

A simple solution to this is forbidding blocked Steiner points, and making sure that every edge is reach-aware. Together with the condition that there is a shortest path (if it exists) between any two vertices with empty bounding box in the visibility graph, this implies the correctness of the second part of our algorithm.

2.2 Visibility Graph Construction

Now we will show how to efficiently find a graph with the properties stated in the previous section. The main idea in [5] is to introduce the median lines recursively, i.e. to insert a median line at the median of all x-coordinates of the endpoints (and thus between every pair of endpoints such that one of them is to the left and one is to the right of the line), add a horizontal line from

Function 1 Preprocessing

1: Compute connected components of blocked area.
2: Compute $\mathcal{E}$.
3: For every $p \in \mathcal{E}$, compute its visible interval.

each endpoint to the median and introduce a Steiner point on the median line, if it is obstacle-avoiding, connect consecutive Steiner points on the median if possible, and then continue with the sets of endpoints left and right of the line separately. We will use a similar construction here.

In order to insert median lines efficiently, we need to know which endpoints can be connected to the median line by a horizontal segment, i.e. which endpoints can "see" the median. In the reach-aware case, there are further complications when the median is blocked by a (small) obstacle – even if an endpoint can see the median, the horizontal segment from the endpoint to the median line might intersect it in a blocked point, thus we cannot introduce a Steiner point there.

Much of the required information can be computed in preprocessing. We define the *visible interval* of an endpoint $p \in \mathcal{E}$ as the maximum horizontal read-aware line through p, which is well-defined as p is unblocked as an endpoint. This allows us to check if an endpoint can see a median line in constant time, independently of the size of the visible interval or the number of obstacles intersecting it.

Therefore, the steps in Function 1 are done as preprocessing using a sweepline algorithm:

In the obstacle-avoiding case, a horizontal sweepline that stores the obstacles in a binary search tree is sufficient. The reach-aware case is more complicated; here we have two binary search trees, one storing all blocked intervals and one only storing blocked intervals of width at least L. This allows insertion and deletion in $\mathcal{O}(\log |O|)$ time, as well as detecting visibility, and makes it possible to precompute visible intervals (horizontal sweepline) and mirror points (vertical sweepline) in $\mathcal{O}((|S|+|O|)\log |O|)$ time. This data structure, but with a vertical sweepline, is also used for determining visibility on the median later in the algorithm, i.e. in Function 2.

At the core of the algorithm is the INSERT_MEDIAN function (Function 2), which deals with the construction discussed in Section 2.1. The recursive insertion of median lines is implemented from left to right, so that our sweepline can keep track of the obstacles on the median. The set M in Function 2 represents the set of points on the current obstacle on the median that cannot see across horizontally, i.e. the points that are relevant for the case in Figure 3(c).

The overall construction of the visibility graph is given in Algorithm 3, where in a post-processing step vertices along a common obstacle boundary are connected.

2.3 Steiner Tree Construction

Given a visibility graph $G = (V, E)$, we are now interested in extracting a Steiner tree. Since G is shortest-path-preserving, any minimum terminal spanning tree routine will provide an approximation guarantee of two. In our implementation, Steiner trees are found in $\mathcal{O}(|E| \log |E|)$ time using a Dijkstra-Kruskal approach from [13]. We also applied Mehlhorn's algorithm [15], with a faster running time of $\mathcal{O}(|E| + |V| \log |V|)$, but found it slower in practice.

This Steiner tree is a 2-approximation of an optimal reach-aware Steiner tree, because it is at most as long as a reach-aware minimum spanning tree. In fact, since Steiner points on obstacles are forbidden, is it a 2-approximation algorithm for a less restrictive version of the problem wherein the longest path across blocked area has length $\leq L$, since both problems coincide for minimum spanning trees and our solution is feasible for both.

Algorithm 2 INSERT_MEDIAN($\mathcal{E}$)

1: Take the median x_m of the x-coordinates of $\mathcal{E}$.
2: INSERT_MEDIAN($\{(x, y) \in \mathcal{E} : x < x_m\}$)
3: Introduce a median line $L_m := \{(x_m, y) \ : \ y \in \mathbb{R}\}$.
4: $M = \emptyset$.
5: **for** every $p = (x, y) \in \mathcal{E}$ in non-incr. order of y **do**
6: **if** (x_m, y) is not blocked and visible from p **then**
7: Add new vertex (x_m, y) and edge $\{(x_m, y), p\}$.
8: On the median line, connect (x_m, y) to the previous one ((x_m, y')) if existing and possible.
9: $M = \emptyset$.
10: **else**
11: Let $[x_l, x_r] \ni x_m$ be the maximum interval s.t. the point (x', y) is blocked for all $x' \in [x_l, x_r]$.
12: $r := \arg\max\{||p - p'||_1 \ : \ p' \in \{(x_l, y), (x_r, y)\}\}$.
13: $q := \arg\min\{||p - p'||_1 \ : \ p' \in \{(x_l, y), (x_r, y)\}\}$.
14: **if** p can see q **then**
15: Add q and connect p with q.
16: **end if**
17: **if** p can (also) see r **then**
18: Add r and connect q with r.
19: **else if** $x = x_l$ or $x = x_r$ is corner **then**
20: Connect p to all $p' \in M$ with $||p - p'||_1 \leq L$
21: Add p to M
22: **end if**
23: **end if**
24: **end for**
25: INSERT_MEDIAN($\{(x, y) \in \mathcal{E} : x > x_m\}$)

Algorithm 3 Visibility graph construction

1: Preprocessing (Function 1).
2: Sort obstacles and initialize empty sweepline of obstacles on the median
3: INSERT_MEDIAN($\mathcal{E}$)
4: Connect points along obstacle boundaries

2.4 Running Time

In the following, n will denote the number of terminals and m will denote the number of rectangles. For $L = 0$, this algorithm has a running time of $\mathcal{O}((n + m)(\log(m + n))^2)$ [5]. For $L > 0$, the running time can be bounded as follows: Let $k = n + m$. The preprocessing in Function 1 is done by our sweepline in $\mathcal{O}(k \log k)$.

The INSERT_MEDIAN function is called k times; each point is in the current set $\mathcal{E}$ for at most $\log_2 k$ medians. Thus, the for loop in lines 5–24 of this function is executed an amortized $\mathcal{O}(k \log k)$ times. Every step in the for loop takes at most $\mathcal{O}(\log k)$, except for line 20. This line takes $\mathcal{O}(n+m)$, which is an upper bound on the size of M. If M has size at most l, the running time of the visibility graph construction is bounded by $\mathcal{O}((k \log k)(l + \log k))$.

The number of vertices of the visibility graph is bounded by $\mathcal{O}(k \log k)$, because the number of mirror points is linear and the number of Steiner points is $\mathcal{O}(k \log k)$ – for each endpoint, a constant number of Steiner points is added every time it is in the set $\mathcal{E}$. The number of edges is bounded by $\mathcal{O}(kl \log k)$.

Constructing the Steiner tree from the visibility graph therefore takes $\mathcal{O}(kl \log k(\log k + \log l))$ time. In the worst case, with k the only upper bound on l, the algorithm has a running time of $\mathcal{O}(k^2(\log k)^2)$. However, this is rarely the case in practice (as is evident from the running times in the next section): If the complexity of each rectilinear polygon is bounded by a constant, the set M in INSERT_MEDIAN has constant size, hence there are $\mathcal{O}(k \log k)$ edges and vertices in the visibility graph and the running time is $\mathcal{O}(k(\log k)^2)$ as in the obstacle-avoiding case.

2.5 Preprocessing of Obstacles

In practice, the running time of the algorithm can be decreased significantly, especially for large L, by preprocessing the obstacles and ignoring those that can never make our solution infeasible. If the diameter of an isolated rectangular obstacle is bounded by L, it could be ignored and a post-processing routine could replace non-reach-aware tree components on that obstacle by segments of its boundary.

We propose a different approach: If half the diameter of a rectangular obstacle is at most L, then a shortest path across this obstacle is always reach-aware. Therefore, we can ignore these obstacles during the construction of the visibility graph, and forbid Steiner points with a degree of more than three on obstacles during the MTST construction. This is slower in theory, but more efficient in practice, because the visibility graph construction takes up the majority of the running time.

The restriction that no Steiner points are allowed on obstacles does not affect the theoretical approximation guarantee and actually yields very useful solutions for applications in buffering: Buffers are often necessary at bifurcations to shield uncritical capacity, which is impossible if these occur on top of an obstacle.

Using these observations, many small rectangular obstacles can be ignored for the visibility graph construction. For sufficiently large L, constructing an ℓ_1-Steiner tree built by an obstacle-unaware heuristic initially and checking feasibility leads to better results and running times of our algorithm.

2.6 Post-Processing

Since the visibility graph only contains certain shortest paths, we found that a combination of simple local search heuristics can significantly improve the result for most instances.

Especially on practical instances, we often encountered large clusters of terminals in an unblocked area. For those, any (obstacle-unaware) Steiner tree algorithm could be used instead. Therefore, in post-processing, we collect maximal components of the constructed tree with unblocked bounding box and reconnect them using the exact FLUTE algorithm [4] for up to 9 terminals and a Prim heuristic in the Delaunay triangulation for larger terminal sets.

Furthermore, there some non-optimal local configurations, such as trunks with more branches on one side and L-shapes that can be mirrored to decrease the length, can be found and processed efficiently for the entire tree by changing the edge structure locally if the resulting tree is reach-aware.

This procedure can be iterated for better results; in our experience, a good trade-off of running time and solution quality is achieved by one iteration for chip instances and three iterations for benchmark instances.

2.7 Multiple Nets

In chip design, there is usually a persistent set of obstacles, but thousands or millions of nets. Many nets consist of only two or three terminals and it would be too time-consuming to compute

the visibility graph from scratch for each net. Therefore, we proceed as follows when processing all nets of a chip: Some preprocessing steps are independent of the terminal set and can be precomputed for all nets. Using this information, we construct an obstacle visibility graph, i.e. a visibility graph for an empty set of terminals. This obstacle graph can be extended to a visibility graph for a given set of terminals by inserting a new median line through every terminal and connecting all obstacle endpoints and other terminals to this line (if possible). This preserves the visibility graph invariant of having a median line between each pair of endpoints with empty bounding box containing an ℓ_1-shortest path between them.

This construction is most useful if a net has few terminals and L is small. In our experience, the number of terminals for which this construction is applied should be less than logarithmic in the number of obstacles and linear (with a very small slope) in L.

3 Experimental Results

We carried out experiments on standard benchmarks from the literature for the obstacle-avoiding Steiner tree problem, as well as some very big industrial instances. Furthermore, for three industrial chips we computed Steiner trees for all existing nets. All tests were carried out on an Intel® Xeon® CPU X5690 @ 3.47GHz, 192 GB RAM with 12 cores.

3.1 Standard Benchmarks

The standard benchmarks are composed as follows. RC01-RC12 are randomly generated instances by Feng et. al. [6], IND1-IND5 are industrial test cases by Synopsys, RT01-RT05 have a fixed ratio of terminals to obstacles of 5, 10 and 50 and were introduced by Lin et. al. [12], and RL01-RL05 are large random instances by Long et. al. [14].

Table 3 shows the lengths of reach-aware Steiner trees found by our algorithm for different values of L, relative to the length of the longer side of the bounding box of the instance including obstacle corners. In most of these benchmarks, no obstacle has both width and height exceeding 10% of the size of the bounding box.

For these instances, the obstacles are given as a set of rectangles whose union represents the blocked area. Here, Opt* denotes the optimum solution (if known) for the obstacle-avoiding Steiner tree problem according to Huang and Young [10]. We added them as a reference value, but recall that they have a slightly different definition of obstacle-avoiding. Edges between two rectangles that share a boundary may be used, whereas with our definition, they would pass through the interior of the blocked area and not be obstacle-avoiding.

Thus, their Steiner trees can be strictly shorter than the optimum according to our definition for any $L < \infty$. For some of the larger instances (marked by $\leq$), the optimum solution is actually unknown; here we present the best value from literature [1, 11]. For two instances, we improve on the best known value from literature for $L = 0$ despite our stricter definition.

With our definition, IND5 becomes infeasible for small L ($\leq 1\%$ of bounding box), as seen in Figure 6 – there is a vertex isolated by a ring of obstacles in the upper right corner. The picture shows our solution for $L = 10\%$ of the instance width, i.e. the difference of the maximum and minimum x-coordinates in the input.

Except for the RL instances, one can see that the tree length gradually decreases with increasing value of L. The RL instances have many terminals that are spread uniformly across the unblocked area. For such instances, the obstacles do not affect the length of a minimum Steiner tree significantly and the length variation is dominated by the Steiner tree approximation. For $L = 0$, our results on three of the RL instances are better than the best previous upper bounds.

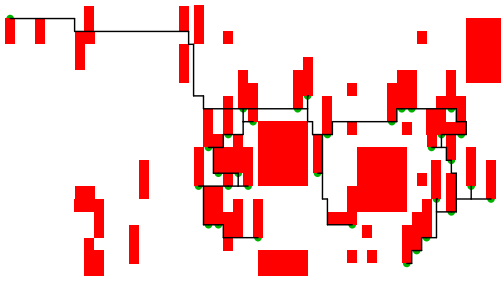


Fig. 6. Our solution for IND5 with $L = 10\%$ instance width.

The reported running times are generally fast. For some instances, the running time first increases and then decreases with growing L. The reason is that we first add more and more edges reaching over obstacles, thereby increasing the size of the visibility graph. Later, increasingly many obstacles can be pruned according to Section 2.5 and the running time decreases. This effect will become even more evident when routing all nets on a chip in Section 3.3.

3.2 Big Chip Instances

Eight industrial test instances arise from a cooperation with IBM. They have between 109 and 783 352 terminals. The bigger ones represent reset trees with low performance requirements, where short length is a major focus. These instances are published as the "BONN" instances as part of the 11th DIMACS benchmark suite on Steiner trees:

`http://dimacs11.cs.princeton.edu/instances`.

Instance	$\|S\|$	$\|O\|$	$L^\star$	**Length**			**RT**
				$L = 0$	$L = L^\star$	$L = \infty$	**sec.**
BIG1	109	101	90	31695	31566	28485	¡1
BIG2	23292	54	2400000	364338561	363004401	361726146	1
BIG3	35574	158	1500000	746523861	746495841	735059181	2
BIG4	46269	127	1500000	1071883920	1071827520	1068448860	4
BIG5	108500	141	4200000	1973406390	1964154690	1957120800	10
BIG6	129399	210	1500000	infeasible	2608227090	2616871950	14
BIG7	639639	382	4200000	3060914728	3028456768	3013106038	99
BIG8	783352	175	1200000	1948056132	1944546732	1931964162	126

Table 1. Results for the BIG instances.

Table 1 shows results for eight real-world instances. The first three columns show the number of terminals, the number of rectangular obstacles, and the reach-length $L^\star$, which depends on the technology and the metal stack of the underlying chip. We then report the lengths for obstacle-avoiding trees ($L = 0$), for the given reach-length $L^\star$ and for ignoring all obstacles ($L = \infty$). Again, one can observe that the lengths gradually decrease with growing L, even though large parts of the length are incurred by unblocked clusters of terminals.

The running times (given for $L = L^\star$) demonstrate that our algorithm is capable of handling even largest instances efficiently.

Figure 7 shows a plot of BIG5 for $L = L^\star$. The reach-length does not allow to pass over the biggest obstacles, but some wires cross smaller obstacles in the center of the chip.

(a) AndreTop, 3 899 379 nets

L	Length	#inf.	CPU	Wall
0	562 032	0	11:23	05:45
0.5	535 453	0	21:47	07:21
1	469 175	0	15:22	06:21
2.5	440 680	0	10:17	05:54
∞	440 537	0	08:18	05:12

(b) AlexTop, 2 674 754 nets

L	Length	#inf.	CPU	Wall
0	580 318*	1 955	21:58	06:10
0.5	536 358*	1	24:52	06:29
1	532 307	0	21:46	06:06
2.5	530 284	0	17:58	05:55
∞	529 301	0	07:07	04:38

(c) LeonardTop, 525 498 nets

L	Length	#inf.	CPU	Wall
0	201 127*	6 669	13:33	02:42
0.5	249 067*	40	16:54	03:11
1	246 862	0	17:41	03:24
2.5	203 378	0	11:31	02:32
∞	199 216	0	01:52	01:24

Choices of L and total net lengths are reported in mm, running times in mm:ss using 8 threads. Total lengths marked by * include infeasible nets with (large) opens!

Table 2. Results on entire chips.

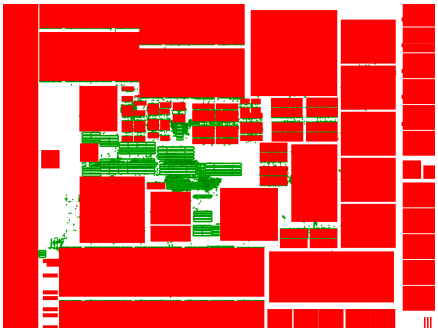


Fig. 7. Industrial instance with 108500 terminals.

3.3 Computations for Entire Chips

The benefit of allowing components to reach over obstacles becomes very evident when computing reach-aware Steiner trees for all nets on a chip. We tested our algorithm on three chips in 65 nm technology, also provided by IBM.

Table 2 shows the total net lengths for $L = 0$, $L = 0.5\,mm$, $L = 1\,mm$, $L = 2.5\,mm$ and $L = \infty$. For small L, some nets are infeasible. For those nets, the solution consists of a reach-aware forest with minimum number of components and only the length of this forest is included in the total length. Therefore, on LeonardTop the total net length for $L = 0$ is lower than for larger reach-lengths, where all nets become feasible. The number of infeasible nets is listed in the "*#inf.*"-columns.

Compared to obstacle-avoiding trees, the reduction in total net length is substantial, e.g. 2.5%, 14.8%, 19.8% and 20% on AndreTop or 8.7%, 9.5%, 9.8% and 10% on AlexTop, here even in presence of infeasible nets for $L = 0$.

LeonardTop contains 26 macros with width and height above $2\,mm$. Thus, a significant length reduction only occurs when raising L to $2.5\,mm$.

Again, the running times first rise with growing L, because the size of the visibility graph increases, and then fall due to the pruning of obstacles. The wall times were obtained using 8 threads.

4 Conclusion

We have proposed a new algorithm for computing reach-aware Steiner trees that is fast in theory and on real-world instances from chip design. It provides a 2-approximation for minimum reach-

Name	$\|S\|$	$\|O\|$	Opt^*	**Lengths**					**Running times in seconds**				
				$L=0$	1%	5%	10%	∞	$L=0$	1%	5%	10%	∞
RL01	5000	5000	$\leq$ 481813	493372	486836	490658	491565	472780	0.65	1.02	0.26	0.28	0.13
RL02	9999	500	$\leq$ 637753	638206	638151	638276	638612	634187	0.66	0.68	0.66	0.65	0.25
RL03	9999	100	$\leq$ 640902	639495	639314	639195	638851	636566	0.72	0.73	0.73	0.72	0.25
RL04	10000	10	$\leq$ 697125	694654	694654	691612	691612	691660	0.76	0.76	0.27	0.27	0.24
RL05	10000	0	$\leq$ 728438	723102	723102	723102	723102	723102	0.27	0.26	0.26	0.26	0.24
RC01	10	10	25980	27360	27360	25290	25290	25290	0.00	0.00	0.00	0.01	0.00
RC02	30	10	41350	43010	43010	42540	41460	41330	0.00	0.00	0.00	0.01	0.00
RC03	50	10	54160	55080	55080	54650	55660	52470	0.01	0.00	0.00	0.00	0.00
RC04	70	10	59070	60300	60300	57410	56120	55330	0.00	0.01	0.00	0.00	0.00
RC05	100	10	74070	75060	75060	73330	73460	71610	0.01	0.00	0.00	0.01	0.00
RC06	100	500	79714	85133	84200	81983	82145	77472	0.03	0.03	0.01	0.01	0.01
RC07	200	500	108740	114225	112168	111249	110343	107190	0.03	0.03	0.03	0.01	0.00
RC08	200	800	112564	120394	116649	113778	115090	109589	0.05	0.05	0.03	0.02	0.00
RC09	200	1000	111005	118116	115169	112665	113571	107561	0.07	0.06	0.02	0.03	0.01
RC10	500	100	164150	168350	168350	166910	166330	164600	0.03	0.02	0.03	0.02	0.00
RC11	1000	100	230873	235424	234930	234827	235407	230620	0.06	0.06	0.06	0.05	0.01
RC12	1000	10000	$\leq$ 756998	792417	785857	785857	785857	754414	0.82	0.11	0.11	0.11	0.08
RT01	10	500	2146	2283	2012	1817	1817	1817	0.01	0.03	0.00	0.01	0.01
RT02	50	500	45852	49500	46762	45772	45772	45747	0.02	0.03	0.00	0.01	0.00
RT03	100	500	7964	8380	8034	8092	8046	7697	0.03	0.03	0.01	0.01	0.01
RT04	100	1000	9693	10616	8160	7788	7788	7788	0.05	0.07	0.01	0.02	0.02
RT05	200	2000	51313	55507	45479	45581	46101	43099	0.12	0.15	0.06	0.04	0.02
IND1	10	32	604	629	629	609	609	609	0.01	0.00	0.00	0.00	0.00
IND2	10	43	9500	10600	10600	9100	9100	9100	0.00	0.00	0.01	0.00	0.00
IND3	10	50	600	678	678	600	587	587	0.00	0.00	0.00	0.00	0.00
IND4	25	79	1086	1160	1160	1137	1121	1092	0.01	0.00	0.00	0.00	0.01
IND5	33	71	1341	infeas.	infeas.	1364	1343	1312	0.00	0.00	0.00	0.01	0.00
Sum									3.62	4.13	2.56	2.56	1.29

Opt^* denotes the optimum solution for $L = 0$ from [10] w.r.t. a slightly more relaxed interpretation of obstacles. Running times of 0.00 were positive but less than 0.01 seconds.

Table 3. Solution length and running times for different choices of L.

aware Steiner trees and is, in contrast to existing 2-approximation algorithms [16], fast enough for practical computations.

The computational results, in particular on entire chips, demonstrate the big length reductions compared to obstacle-avoiding Steiner trees for the purpose of computing short and reach-aware buffer tree topologies.

References

1. G. Ajwani, C. Chu and W.-K. Mak. *FOARS: FLUTE Based Obstacle-Avoiding Rectilinear Steiner Tree Construction*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 30 (2011), 194–204.
2. C. J. Alpert, A. B. Kahng, C. N. Sze and Q. Wang. *Timing-driven Steiner trees are (practically) free*, Proceedings of the Design Automation Conference (2006), 389–392.
3. C. Bartoschek, S. Held, J. Maberg, D. Rautenbach and J. Vygen. *The repeater tree construction problem*, Information Processing Letters 110 (2010), 1079–1083.
4. C. Chu and Y.-C. Wong. *FLUTE: Fast Lookup Table Based Rectilinear Steiner Minimal Tree Algorithm for VLSI Design*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 27 (2008), 70–83.
5. K. Clarkson, S. Kapoor and P. Vaidya. *Rectilinear shortest paths through polygonal obstacles in* $\mathcal{O}(n(\log n)^2)$ *time*, Proceedings of the Symposium on Computational Geometry (1987), 251–257.
6. Z. Feng, Y. Hu, T. Jing, X. Hong, X. Hu and G. Yan. *An* $\mathcal{O}(n \log n)$ *algorithm for obstacle-avoiding routing tree construction in the* λ*-geometry plane*, Proceedings of the International Symposium on Physical Design (2006), 48–55.
7. J. L. Ganley and J. P. Cohoon. *Routing a multi-terminal critical net: Steiner tree construction in the presence of obstacles*, Proceedings of the IEEE International Symposium on Circuits and Systems vol.1 (1994), 113–116.
8. S. Held and D. Rotter. *Shallow Light Steiner Arborescences with Vertex Delays*, Proceedings of the International Conference on Integer Programming and Combinatorial Optimization (2013), 229–241.
9. T. Huang and E. F. Y. Young. *Construction of rectilinear Steiner minimum trees with slew constraints over obstacles*, Proceedings of the International Conference on Computer-Aided Design (2012), 144–151.
10. T. Huang and E. F. Y. Young. *ObSteiner: An Exact Algorithm for the Construction of Rectilinear Steiner Minimum Trees in the Presence of Complex Rectilinear Obstacles*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 32 (2013), 882–893.
11. L. Li and E. F. Y. Young. *Obstacle-avoiding Rectilinear Steiner Tree Construction*, Proceedings of the International Conference on Computer-Aided Design (2008), 523–528.
12. C.-W. Lin, S. Y. Chen, C.-F. Li, Y.-W. Chang and C.-L. Yang. *Obstacle-Avoiding Rectilinear Steiner Tree Construction Based on Spanning Graphs*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 27 (2008), 643–653.
13. C.-H. Liu, S.-Y. Yuan, S.-Y. Kuo and S.-C. Wang. *High-performance obstacle-avoiding rectilinear Steiner tree construction* ACM Transactions on Design Automation of Electronic Systems 14 (2009), article 45.
14. J. Long, H. Zhou and S. O. Memik. *An* $\mathcal{O}(n \log n)$ *edge-based algorithm for obstacle-avoiding rectilinear Steiner tree construction*, Proceedings of the International Symposium on Physical Design (2008), 126–133.
15. K. Mehlhorn. *A faster approximation algorithm for the Steiner problem in graphs*, Information Processing Letters 27 (1988), 125–128.
16. M. Müller-Hannemann and S. Peyer. *Approximation of Rectilinear Steiner Trees with Length Restrictions on Obstacles*, Proceedings of the Workshop on Algorithms and Data Structures (2003), LNCS 2748, 207–218.
17. Y. Zhang, A. Chakraborty, S. Chowdhury and D. Z. Pan. *Reclaiming over-the-IP-block routing resources with buffering-aware rectilinear Steiner minimum tree construction.* Computer-Aided Design (ICCAD), 2012 IEEE/ACM International Conference on, pp. 137-143. IEEE, 2012.