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

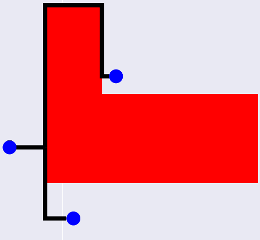
Stephan Held and Sophie Spirkl

Research Institute for Discrete Mathematics, University of Bonn

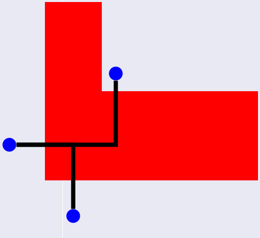
ISPD, March 30–April 2, 2014

Motivation

Example



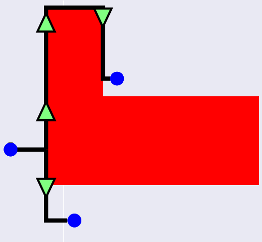
obstacle-avoiding



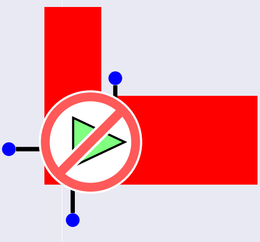
obstacle-unaware

Motivation

Example



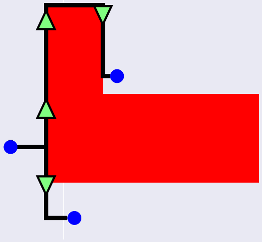
obstacle-avoiding



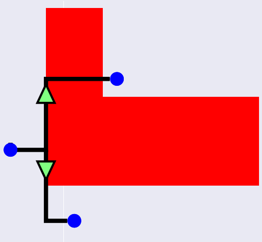
obstacle-unaware

Motivation

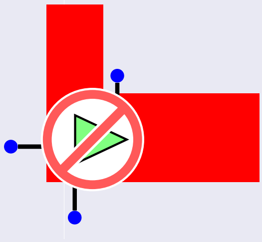
Example



obstacle-avoiding



reach-aware

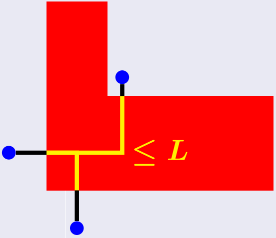


obstacle-unaware

Definition (Reach-aware Steiner tree)

Input:

- terminals T,
- rectilinear obstacles R,
- a reach length $L \in [0, \infty]$.



A Steiner tree Y connecting T is reach-aware if the length of each connected component in the intersection of Y with the interior of the blocked area $(\bigcup_{r \in R} r)^\circ$ is bounded by L.

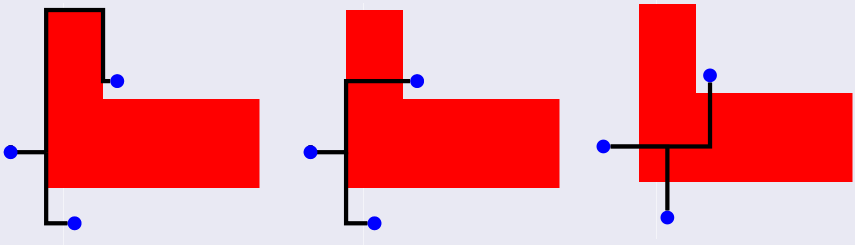
- All objects are considered to be in $\mathbb{R}^2$ with the ℓ_1-norm.
- This formulation does not depend on representation of blocked area, therefore we will assume R to be a set of rectangles.

Problem Formulation

Reach-aware Steiner tree problem

Find a reach-aware Steiner tree of minimum length.

Example



obstacle-avoiding ($L = 0$) | reach-aware ($0 < L < \infty$) | obstacle-unaware ($L = \infty$)

Problem Formulation

Reach-aware Steiner tree problem

Find a reach-aware Steiner tree of minimum length.

Previous Result

Müller-Hannemann and Peyer [2003]:

- Steiner tree algorithm on augmented Hanan grid
- 2-approximation with super-quadratic running time and space
- $\frac{2k}{2k-1}\alpha$-approximation for rectangles, where α is the approximation ratio in graphs

Main Result

Let $k = |T| + |R|$ denote the size of the input.

Theorem (Held and S. [2014])

A graph containing shortest reach-aware paths between all pairs of terminals of size $O(k^2 \log k)$ can be computed in $O(k^2 \log k)$ time.

Corollary (Held and S. [2014])

A 2-approximation for the minimum reach-aware Steiner tree problem can be computed in $O((k \log k)^2)$ time.

- If the number of corners of each rectilinear obstacle is bounded by a constant, the running time is $O(k(\log k)^2)$.

Main Result

Let $k = |T| + |R|$ denote the size of the input.

Theorem (Held and S. [2014])

A graph containing shortest reach-aware paths between all pairs of terminals of size $O(k^2 \log k)$ can be computed in $O(k^2 \log k)$ time.

Corollary (Held and S. [2014])

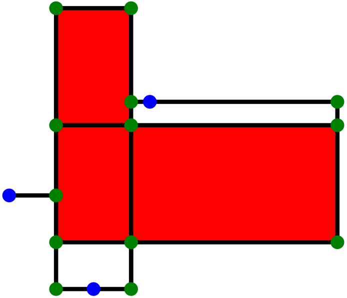
A 2-approximation for the minimum reach-aware Steiner tree problem can be computed in $O((k \log k)^2)$ time.

- If the number of corners of each rectilinear obstacle is bounded by a constant, the running time is $O(k(\log k)^2)$.

Reach-Aware Visibility Graph

We construct the reach-aware visibility graph with the following properties:

- There is a reach-aware shortest path between every pair of terminals.
- Every subset of the edge set is reach-aware.



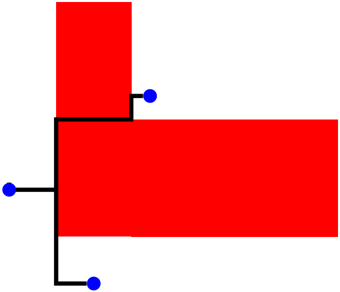
Lemma

A minimum terminal spanning tree is a 2-approximation.

Reach-Aware Visibility Graph

We construct the reach-aware visibility graph with the following properties:

- There is a reach-aware shortest path between every pair of terminals.
- Every subset of the edge set is reach-aware.



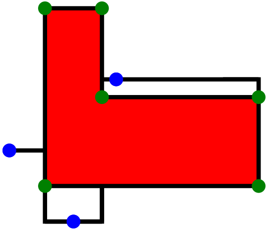
Lemma

A minimum terminal spanning tree is a 2-approximation.

Reach-Aware Visibility Graph

For $L = 0$, Clarkson et al. [1987] proved that a graph containing shortest paths between all terminals of size $\mathcal{O}(k \log k)$ can be computed in $\mathcal{O}(k(\log k)^2)$ time.

We generalized their construction.



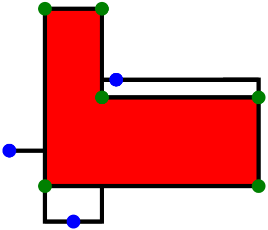
Clarkson graph

Other previous results include:

- PTAS by Min et al. [2003]
- 2-approximations by Lin et al. [2008], Long et al. [2008], Liu et al. [2009]
- Exact algorithm by Huang et al. [2013]

Reach-Aware Visibility Graph

For $L = 0$, Clarkson et al. [1987] proved that a graph containing shortest paths between all terminals of size $\mathcal{O}(k \log k)$ can be computed in $\mathcal{O}(k(\log k)^2)$ time.



Clarkson graph

We generalized their construction.

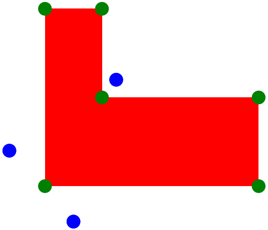
Other previous results include:

- PTAS by Min et al. [2003]
- 2-approximations by Lin et al. [2008], Long et al. [2008], Liu et al. [2009]
- Exact algorithm by Huang et al. [2013]

Path Decomposition Lemma

The set of endpoints $\mathcal{E}$ contains all terminals and obstacle corners.

The bounding box of two endpoints is empty, if it intersects no other endpoint.



Endpoints

Lemma (Clarkson et al. [1987])

A shortest obstacle-avoiding path between two endpoints can be modified s. t.

- *the bounding box of two consecutive endpoints is empty, and*
- *its restriction to that bounding box is an ℓ_1-shortest path.*

This modification preserves length and obstacle-avoidance.

Path Decomposition Lemma

Goal

A shortest reach-aware path between two endpoints can be modified s. t.

- the bounding box of two consecutive endpoints is empty, and
- its restriction to that bounding box is an ℓ_1-shortest path.

This modification preserves length and reach-awareness.

- The lemma does not hold in the reach-aware case:

Path Decomposition Lemma

Goal

A shortest reach-aware path between two endpoints can be modified s. t.

- the bounding box of two consecutive endpoints is empty, and
- its restriction to that bounding box is an ℓ_1-shortest path.

This modification preserves length and reach-awareness.

- The lemma does not hold in the reach-aware case:

Path Decomposition Lemma

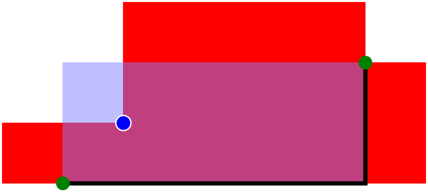
Goal

A shortest reach-aware path between two endpoints can be modified s. t.

- the bounding box of two consecutive endpoints is empty, and
- its restriction to that bounding box is an ℓ_1-shortest path.

This modification preserves length and reach-awareness.

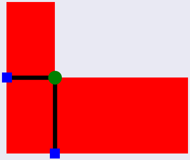
- The lemma does not hold in the reach-aware case:



Mirror Points

Definition

A mirror point (blue square) is the endpoint of an axis-parallel connection across an obstacle at a non-convex corner (green disk).



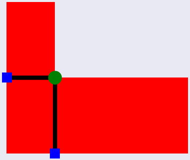
Endpoints $\mathcal{E}$

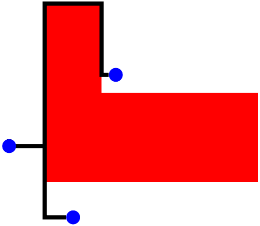
- From now on, we only consider the extended set of endpoints $\mathcal{E}$, which contains terminals, obstacle corners and mirror points.

Mirror Points

Definition

A mirror point (blue square) is the endpoint of an axis-parallel connection across an obstacle at a non-convex corner (green disk).





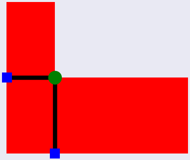
Endpoints $\mathcal{E}$

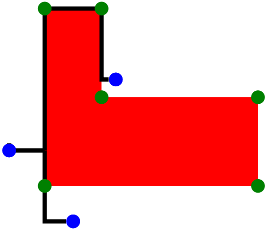
- From now on, we only consider the extended set of endpoints $\mathcal{E}$, which contains terminals, obstacle corners and mirror points.

Mirror Points

Definition

A mirror point (blue square) is the endpoint of an axis-parallel connection across an obstacle at a non-convex corner (green disk).





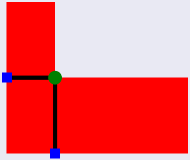
Endpoints $\mathcal{E}$

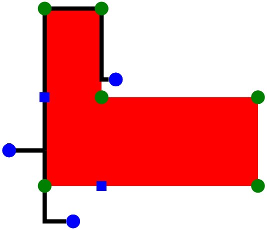
- From now on, we only consider the extended set of endpoints $\mathcal{E}$, which contains terminals, obstacle corners and mirror points.

Mirror Points

Definition

A mirror point (blue square) is the endpoint of an axis-parallel connection across an obstacle at a non-convex corner (green disk).





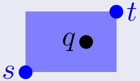
Endpoints $\mathcal{E}$

- From now on, we only consider the extended set of endpoints $\mathcal{E}$, which contains terminals, obstacle corners and mirror points.

Path Decomposition Lemma

Definition

For two points s and t, their closed bounding box is empty, if it contains no endpoints except for s and t.



not empty

Lemma (Held and S. [2014])

A shortest reach-aware path between two endpoints can be modified s. t.

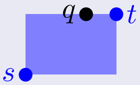
- *the bounding box of two consecutive endpoints is empty, and*
- *its restriction to that bounding box is an ℓ_1-shortest path.*

This modification preserves length and reach-awareness.

Path Decomposition Lemma

Definition

For two points s and t, their closed bounding box is empty, if it contains no endpoints except for s and t.



still not empty

Lemma (Held and S. [2014])

A shortest reach-aware path between two endpoints can be modified s. t.

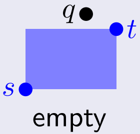
- *the bounding box of two consecutive endpoints is empty, and*
- *its restriction to that bounding box is an ℓ_1-shortest path.*

This modification preserves length and reach-awareness.

Path Decomposition Lemma

Definition

For two points s and t, their closed bounding box is empty, if it contains no endpoints except for s and t.



Lemma (Held and S. [2014])

A shortest reach-aware path between two endpoints can be modified s. t.

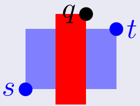
- *the bounding box of two consecutive endpoints is empty, and*
- *its restriction to that bounding box is an ℓ_1-shortest path.*

This modification preserves length and reach-awareness.

Path Decomposition Lemma

Definition

For two points s and t, their closed bounding box is empty, if it contains no endpoints except for s and t.



still empty

Lemma (Held and S. [2014])

A shortest reach-aware path between two endpoints can be modified s. t.

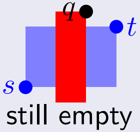
- *the bounding box of two consecutive endpoints is empty, and*
- *its restriction to that bounding box is an ℓ_1-shortest path.*

This modification preserves length and reach-awareness.

Path Decomposition Lemma

Definition

For two points s and t, their closed bounding box is empty, if it contains no endpoints except for s and t.



Lemma (Held and S. [2014])

A shortest reach-aware path between two endpoints can be modified s. t.

- *the bounding box of two consecutive endpoints is empty, and*
- *its restriction to that bounding box is an ℓ_1-shortest path.*

This modification preserves length and reach-awareness.

Path Decomposition Lemma

Lemma (Held and S. [2014])

A shortest reach-aware path between two endpoints can be modified s. t.

- *the bounding box of two consecutive endpoints is empty, and*
- *its restriction to that bounding box is an ℓ_1-shortest path.*

This modification preserves length and reach-awareness.

- This fixes the earlier example:

Path Decomposition Lemma

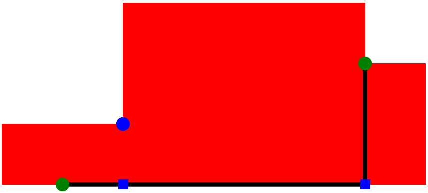
Lemma (Held and S. [2014])

A shortest reach-aware path between two endpoints can be modified s. t.

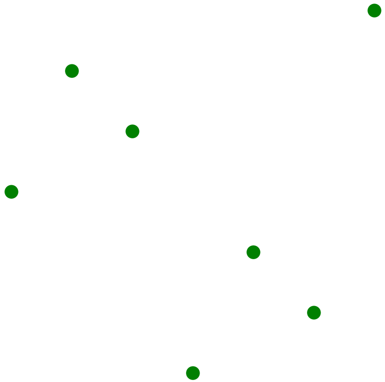
- *the bounding box of two consecutive endpoints is empty, and*
- *its restriction to that bounding box is an ℓ_1-shortest path.*

This modification preserves length and reach-awareness.

- This fixes the earlier example:



Medians

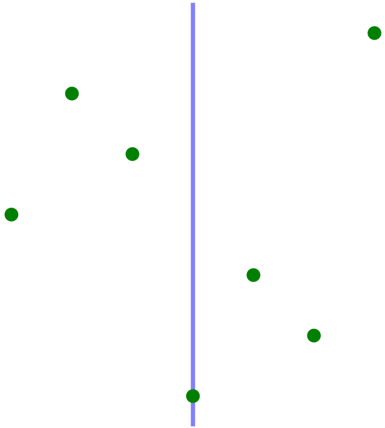


Algorithm

- Take a set of points
- Insert vertical line at median of x-coordinates
- Connect all points to median line
- Proceed recursively left and right

- Size $\mathcal{O}(k \log k)$
- Contains shortest paths between terminals

Medians

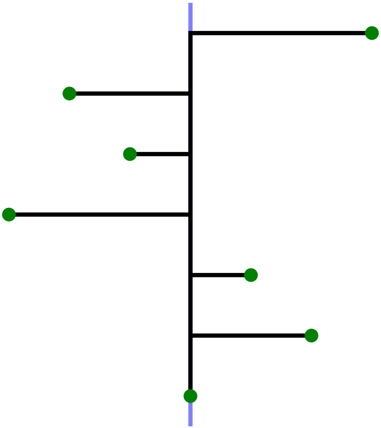


Algorithm

- Take a set of points
- Insert vertical line at median of x-coordinates
- Connect all points to median line
- Proceed recursively left and right

- Size $\mathcal{O}(k \log k)$
- Contains shortest paths between terminals

Medians

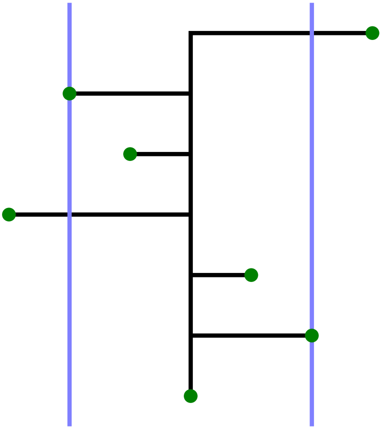


Algorithm

- Take a set of points
- Insert vertical line at median of x-coordinates
- Connect all points to median line
- Proceed recursively left and right

- Size $\mathcal{O}(k \log k)$
- Contains shortest paths between terminals

Medians

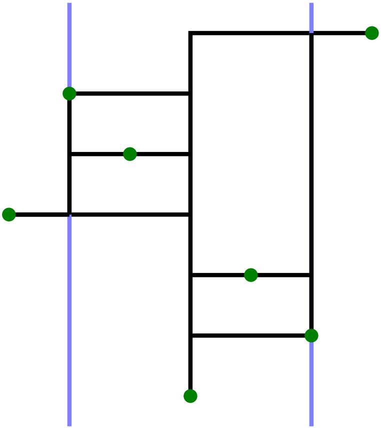


Algorithm

- Take a set of points
- Insert vertical line at median of x-coordinates
- Connect all points to median line
- Proceed recursively left and right

- Size $\mathcal{O}(k \log k)$
- Contains shortest paths between terminals

Medians

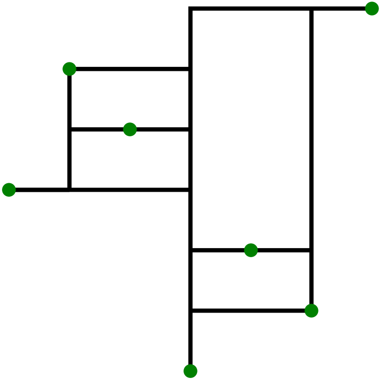


Algorithm

- Take a set of points
- Insert vertical line at median of x-coordinates
- Connect all points to median line
- Proceed recursively left and right

- Size $\mathcal{O}(k \log k)$
- Contains shortest paths between terminals

Medians



Algorithm

- Take a set of points
- Insert vertical line at median of x-coordinates
- Connect all points to median line
- Proceed recursively left and right

- Size $\mathcal{O}(k \log k)$
- Contains shortest paths between terminals

Medians for Reach-Aware Visibility Graph

- Insert medians lines recursively
- Connect endpoints on opposite sides by shortest path

If the bounding box of the two endpoints is empty, 3 cases can occur:

Case 1:
median unblocked

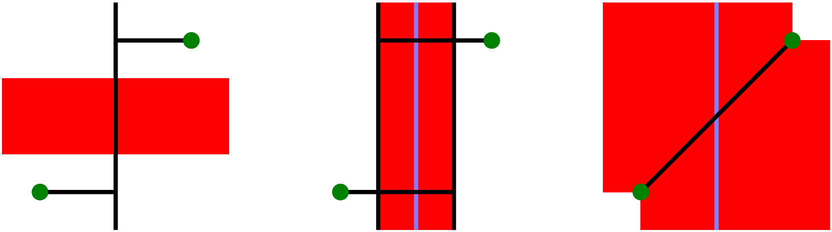
Case 2a:
blocked, can cross

Case 2b:
cannot cross

Medians for Reach-Aware Visibility Graph

- Insert medians lines recursively
- Connect endpoints on opposite sides by shortest path

If the bounding box of the two endpoints is empty, 3 cases can occur:



Case 1:
median unblocked

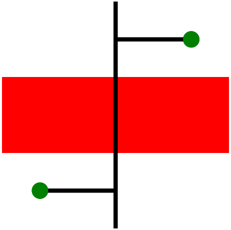
Case 2a:
blocked, can cross

Case 2b:
cannot cross

Medians for Reach-Aware Visibility Graph

- Insert medians lines recursively
- Connect endpoints on opposite sides by shortest path

If the bounding box of the two endpoints is empty, 3 cases can occur:



Case 1:
median unblocked

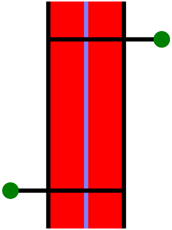
Case 1

- Mirror points ensure that if unblocked for one point, then for both
- Add path as shown if reach-aware
- If ℓ_1-shortest reach-aware path exists, then this path is one

Medians for Reach-Aware Visibility Graph

- Insert medians lines recursively
- Connect endpoints on opposite sides by shortest path

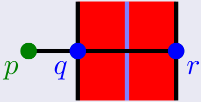
If the bounding box of the two endpoints is empty, 3 cases can occur:



Case 2a:
blocked, can cross

Case 2a

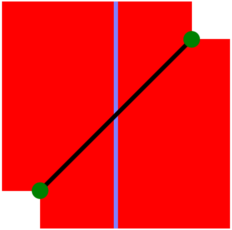
- If connection to median reach-aware, add pq and qr, if possible
- Connect points along obstacle boundaries



Medians for Reach-Aware Visibility Graph

- Insert medians lines recursively
- Connect endpoints on opposite sides by shortest path

If the bounding box of the two endpoints is empty, 3 cases can occur:



Case 2b:
cannot cross

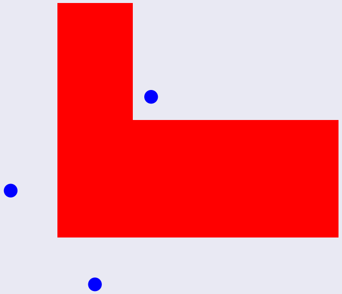
Case 2b

- Pairs of non-convex obstacle corners of the same obstacle
- Connect diagonally

There are few such connections in practice.

Example

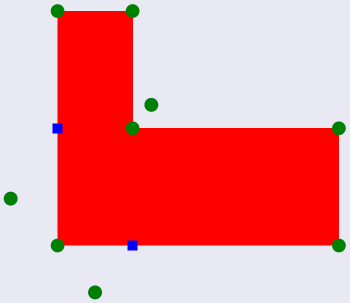
Algorithm



- Instance
- Collect endpoints and compute mirror points
- Insert medians recursively
- Connect points along obstacle boundaries
- Extract Steiner tree

Example

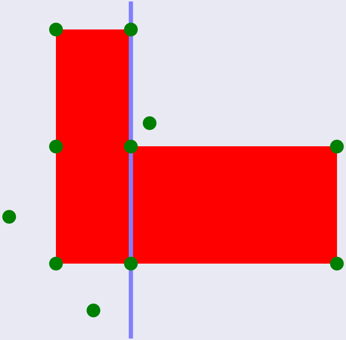
Algorithm



- Instance
- Collect endpoints and compute mirror points
- Insert medians recursively
- Connect points along obstacle boundaries
- Extract Steiner tree

Example

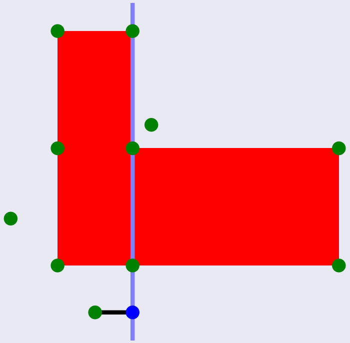
Algorithm



- Instance
- Collect endpoints and compute mirror points
- Insert medians recursively
- Connect points along obstacle boundaries
- Extract Steiner tree

Example

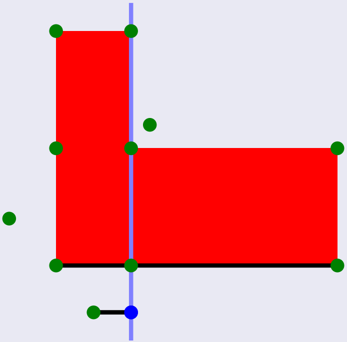
Algorithm



- Instance
- Collect endpoints and compute mirror points
- Insert medians recursively
- Connect points along obstacle boundaries
- Extract Steiner tree

Example

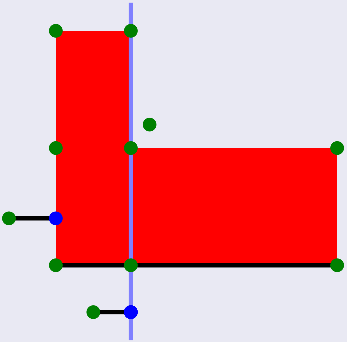
Algorithm



- Instance
- Collect endpoints and compute mirror points
- Insert medians recursively
- Connect points along obstacle boundaries
- Extract Steiner tree

Example

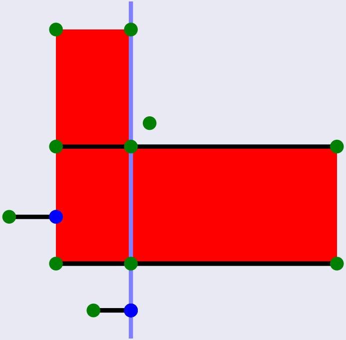
Algorithm



- Instance
- Collect endpoints and compute mirror points
- Insert medians recursively
- Connect points along obstacle boundaries
- Extract Steiner tree

Example

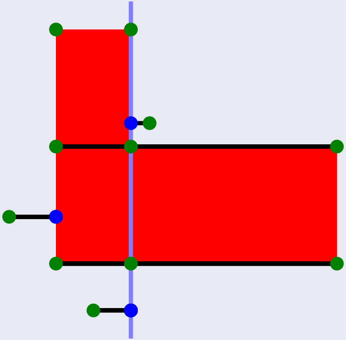
Algorithm



- Instance
- Collect endpoints and compute mirror points
- Insert medians recursively
- Connect points along obstacle boundaries
- Extract Steiner tree

Example

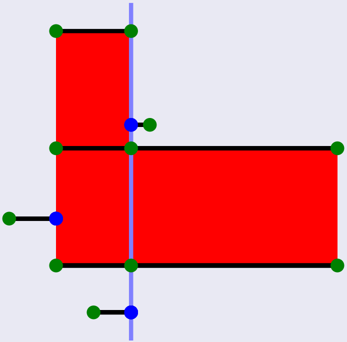
Algorithm



- Instance
- Collect endpoints and compute mirror points
- Insert medians recursively
- Connect points along obstacle boundaries
- Extract Steiner tree

Example

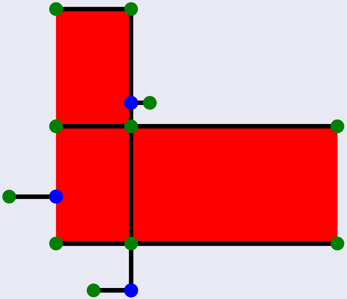
Algorithm



- Instance
- Collect endpoints and compute mirror points
- Insert medians recursively
- Connect points along obstacle boundaries
- Extract Steiner tree

Example

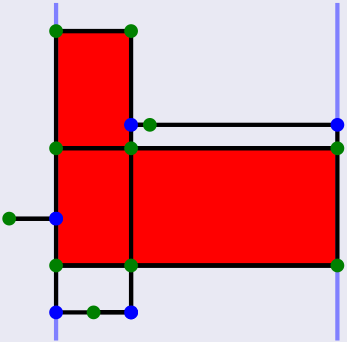
Algorithm



- Instance
- Collect endpoints and compute mirror points
- Insert medians recursively
- Connect points along obstacle boundaries
- Extract Steiner tree

Example

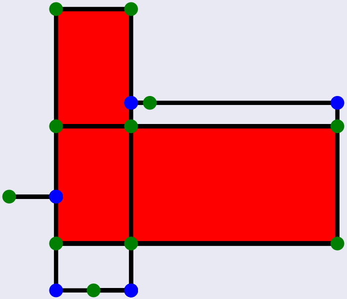
Algorithm



- Instance
- Collect endpoints and compute mirror points
- Insert medians recursively
- Connect points along obstacle boundaries
- Extract Steiner tree

Example

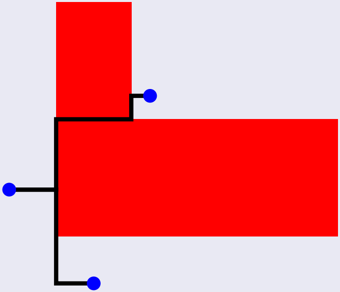
Algorithm



- Instance
- Collect endpoints and compute mirror points
- Insert medians recursively
- Connect points along obstacle boundaries
- Extract Steiner tree

Example

Algorithm



- Instance
- Collect endpoints and compute mirror points
- Insert medians recursively
- Connect points along obstacle boundaries
- Extract Steiner tree

Analysis of Visibility Graph Construction

Let $k = |T| + |R|$ denote the size of the input, l the maximum number of corners of an obstacle.

- There are $\mathcal{O}(k)$ endpoints in $\mathcal{E}$
- Each endpoint is connected to $\mathcal{O}(\log k)$ medians
- Including diagonal edges, such a connection increases the graph size by $\mathcal{O}(l)$

Theorem (Held and S. [2014])

A graph containing shortest reach-aware paths between all pairs of terminals of size $O(kl \log k)$ can be computed in $O(k \log k \cdot (l + \log k))$ time.

Analysis of Visibility Graph Construction

Let $k = |T| + |R|$ denote the size of the input, l the maximum number of corners of an obstacle.

- There are $\mathcal{O}(k)$ endpoints in $\mathcal{E}$
- Each endpoint is connected to $\mathcal{O}(\log k)$ medians
- Including diagonal edges, such a connection increases the graph size by $\mathcal{O}(l)$

Theorem (Held and S. [2014])

A graph containing shortest reach-aware paths between all pairs of terminals of size $O(kl \log k)$ can be computed in $O(k \log k \cdot (l + \log k))$ time.

Analysis of Visibility Graph Construction

Let $k = |T| + |R|$ denote the size of the input, l the maximum number of corners of an obstacle.

- There are $\mathcal{O}(k)$ endpoints in $\mathcal{E}$
- Each endpoint is connected to $\mathcal{O}(\log k)$ medians
- Diagonal edges take time $\mathcal{O}(1)$, others need to check reach-awareness in $\mathcal{O}(\log k)$

Theorem (Held and S. [2014])

A graph containing shortest reach-aware paths between all pairs of terminals of size $O(kl \log k)$ can be computed in $O(k \log k \cdot (l + \log k))$ time.

Analysis of Steiner Tree Construction

Let $k = |T| + |R|$ denote the size of the input, l the maximum number of corners of an obstacle.

- The visibility graph contains reach-aware shortest paths between all terminals
- There are no Steiner points on obstacles
- Any Steiner tree in the visibility graph is reach-aware

Corollary (Held and S. [2014])

A 2-approximation for the minimum reach-aware Steiner tree problem can be computed in $O(kl \log k(\log l + \log k))$ time.

We used a Dijkstra-Kruskal approach of Liu et al. [2009] with running time $\mathcal{O}(m \log m)$ for m edges.

Analysis of Steiner Tree Construction

Let $k = |T| + |R|$ denote the size of the input, l the maximum number of corners of an obstacle.

- The visibility graph contains reach-aware shortest paths between all terminals
- There are no Steiner points on obstacles
- Any Steiner tree in the visibility graph is reach-aware

Corollary (Held and S. [2014])

A 2-approximation for the minimum reach-aware Steiner tree problem can be computed in $O(kl \log k(\log l + \log k))$ time.

We used a Dijkstra-Kruskal approach of Liu et al. [2009] with running time $\mathcal{O}(m \log m)$ for m edges.

Analysis of Steiner Tree Construction

Let $k = |T| + |R|$ denote the size of the input, l the maximum number of corners of an obstacle.

- The visibility graph contains reach-aware shortest paths between all terminals
- There are no Steiner points on obstacles
- Any Steiner tree in the visibility graph is reach-aware

Corollary (Held and S. [2014])

A 2-approximation for the minimum reach-aware Steiner tree problem can be computed in $O(k(\log k)^2)$ time, if l is constant.

We used a Dijkstra-Kruskal approach of Liu et al. [2009] with running time $\mathcal{O}(m \log m)$ for m edges.

Post-Optimization

Unblocked optimization

Rebuild subtrees whose bounding box is unblocked:

- Replace maximal subtrees by 1.5-approximation of RSMT
- Build subtrees for up to 9 terminals optimally using FLUTE

- Local optimizations:

Flip L's

Shift segments

Post-Optimization

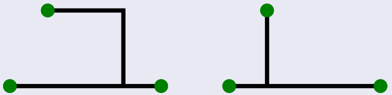
Unblocked optimization

Rebuild subtrees whose bounding box is unblocked:

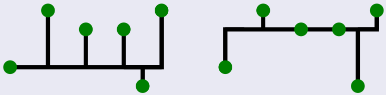
- Replace maximal subtrees by 1.5-approximation of RSMT
- Build subtrees for up to 9 terminals optimally using FLUTE

- Local optimizations:

Flip L's



Shift segments

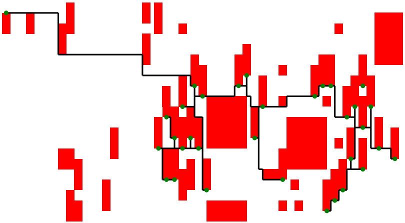


Standard Benchmarks

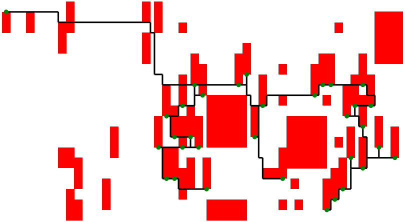
Name	$\|S\|$	$\|O\|$	Best	Lengths				
				$L=0$	1%	5%	10%	∞
RL01	5000	5000	481813	493372	486836	490658	491565	472780
RL02	9999	500	637753	638206	638151	638276	638612	634187
RL03	9999	100	640902	639495	639314	639195	638851	636566
RL04	10000	10	697125	694654	694654	691612	691612	691660
RL05	10000	0	728438	723102	723102	723102	723102	723102
RT01	10	500	2146	2283	2012	1817	1817	1817
RT02	50	500	45852	49500	46762	45772	45772	45747
RT03	100	500	7964	8380	8034	8092	8046	7697
RT04	100	1000	9693	10616	8160	7788	7788	7788
RT05	200	2000	51313	55507	45479	45581	46101	43099
IND1	10	32	604	629	629	609	609	609
IND2	10	43	9500	10600	10600	9100	9100	9100
IND3	10	50	600	678	678	600	587	587
IND4	25	79	1086	1160	1160	1137	1121	1092
IND5	33	71	1341	infeas.	infeas.	1364	1343	1312
Σ RT				3.62	4.13	2.56	2.56	1.29

Best: best published for $L = 0$ with relaxed definition of obstacles; opt. on RT, IND

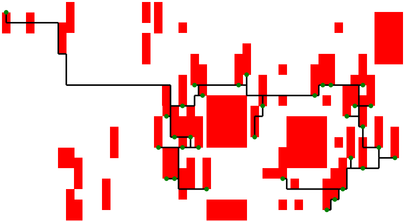
IND5 (Standard Benchmark Instance)



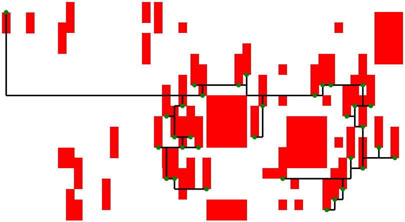
$L = 0$, infeasible



$L = 10$, length $= 1364$

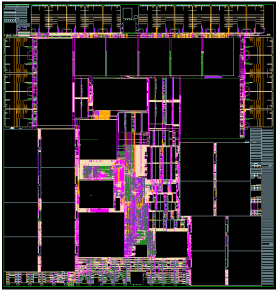


$L = 50$, length $= 1343$

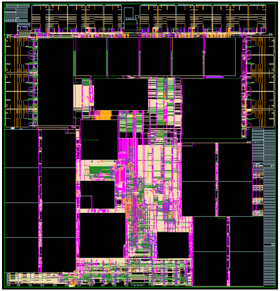


$L = \infty$, length $= 1312$

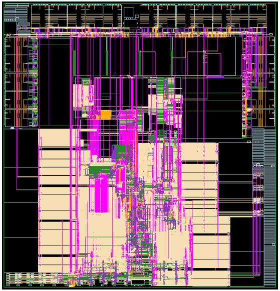
Results on Chips



LeonardTop
obstacle-avoiding
$L = 0$



LeonardTop
reach-aware
$L = 1mm$

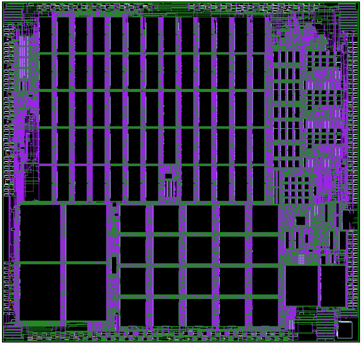


LeonardTop
obstacle-unaware
$L = \infty$

Results on Chips

AndreTop, 3 899 379 nets

L	**Length**	**#inf.**	**CPU**	**Wall**
0	562 032	0	11:23	5:45
0.5	535 453	0	21:47	7:21
1	469 175	0	15:22	6:21
2.5	440 680	0	10:17	5:54
∞	440 537	0	08:18	5:12

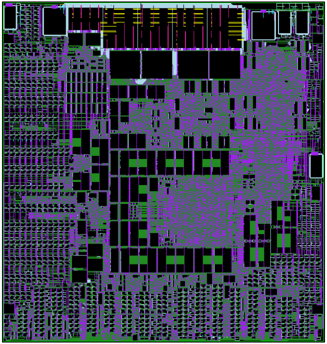


Choices of L and total net lengths reported in mm, running times in mm:ss using 8 threads. Lengths marked by * include infeasible nets with opens.

Results on Chips

AlexTop, 2 674 754 nets

L	**Length**	**#inf.**	**CPU**	**Wall**
0	580 318*	1 955	21:58	6:10
0.5	536 358*	1	24:52	6:29
1	532 307	0	21:46	6:06
2.5	530 284	0	17:58	5:55
∞	529 301	0	07:07	4:38

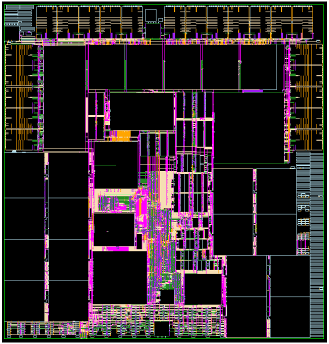


Choices of L and total net lengths reported in mm, running times in mm:ss using 8 threads. Lengths marked by * include infeasible nets with opens.

Results on Chips

LeonardTop, 525 498 nets

L	**Length**	**#inf.**	**CPU**	**Wall**
0	201 127*	6 669	13:33	2:42
0.5	249 067*	40	16:54	3:11
1	246 862	0	17:41	3:24
2.5	203 378	0	11:31	2:32
∞	199 216	0	01:52	1:24



Choices of L and total net lengths reported in mm, running times in mm:ss using 8 threads. Lengths marked by * include infeasible nets with opens.

Some of our instances are part of the
11th DIMACS implementation challenge:
`http://dimacs11.cs.princeton.edu/home.html`
Organizers:

D. Johnson, T. Koch, R.F. Werneck, M. Zachariasen

Instance	$\|T\|$	$\|O\|$	$L^\star$	**Length**			**RT**
				$L = 0$	$L = L^\star$	$L = \infty$	**sec.**
Bonn_23292_54	23292	54	2400	364338	363004	361726	1
Bonn_35574_158	35574	158	1500	746523	746495	735059	2
Bonn_46269_127	46269	127	1500	1071883	1071827	1068448	4
Bonn_108500_141	108500	141	4200	1973406	1964154	1957120	10
Bonn_129399_210	129399	210	1500	infeas.	2608227	2616871	14
Bonn_639639_382	639639	382	4200	3060914	3028456	3013106	99
Bonn_783352_175	783352	175	1200	1948056	1944546	1931964	126

All lengths scaled by 10^{-3}.

Applications

Steiner trees constructed by our algorithm can be used as initial solutions:

Timing

- Cong et al. [1992]
- Khuller et al. [1995]
- Held et al. [2013]

Routing

- Incorporated in BonnTools (BonnRoute Global) to generate starting solutions quickly for majority of nets