

Computing Geometric Structures of Low Stabbing Number in the Plane.*

Joseph S. B. Mitchell[†]

Eli Packer[‡]

Abstract

The stabbing number of a geometric structure in \mathbb{R}^2 with respect to lines is the maximum number of times any line stabs (intersects) the structure. We present algorithms for constructing popular geometric data structures with small stabbing number and for computing lower bounds on an optimal solution. We evaluate our methods experimentally.

1 Introduction

Many geometric applications require constructing an “optimal” geometric network (e.g., spanning tree, matching, or triangulation) on an input set of points in the plane. A common objective function is to minimize the total length of the network. In this paper we study the objective function of minimizing the (line) *stabbing number*, SN , of the network – the maximum number of times any line is stabbed (intersected) by the network. Geometric structures of low stabbing number arise in various computational geometry problems, e.g., implicit point location, polygon containment, and hidden surface removal. While it is known that structures of low stabbing number exist (e.g., SN of stabbing number $O(\sqrt{n})$ for spanning trees), the problem of minimizing SN is known to be NP-hard, and little is known about provable approximation; see [1, 2].

Given a set of n points P in \mathbb{R}^2 and a specification of a type of geometric structure \mathcal{D} that is to be constructed on the points of P , the goal is to find an instance D of \mathcal{D} that minimizes SN . Let E denotes the edges in the output and L be the set of all lines in \mathbb{R}^2 . Formally, the goal is to optimize $\min_{D \in \mathcal{D}} \max_{l \in L} |\{e \in E : e \bowtie l\}|$ where \bowtie is the predicate that tests whether its arguments intersect.

In this work we consider three important geometric network structures: spanning trees, triangulations, and perfect matchings. In Section 2 we describe several heuristics to minimize the SN . In Section 3 we present a heuristic to compute lower bounds, based on the notion of “shattering”.

2 Heuristics for Upper Bounds

We build three geometric structures: spanning trees (denoted by \mathcal{S}), triangulations (\mathcal{T}) and perfect matchings (\mathcal{P}). We use a greedy strategy for each, with the objective to iteratively select edges to minimize the increase in SN . For computing \mathcal{S} we use a variant of the *Kruskal* algorithm, selecting edges that minimize the increase in SN (denoted by \mathcal{A}), instead of shortest edges. For computing \mathcal{T} and \mathcal{P} , the greedy heuristic is similar: we iteratively add the edge that minimizes the increase in SN , until the output is complete.

For any line $l \in L$ that does not intersect any point of P , let $P_r(l)$ and $P_l(l)$ be the sets of points in P that lie to the right and left of l respectively (for horizontal lines, $P_r(l)$ and $P_l(l)$ will be the points above and below). We say that two lines $l_1, l_2 \in L$ are *topologically equivalent* if $P_r(l_1) = P_r(l_2)$. Note that topologically equivalent lines stab the same subset of edges in any D , so in our heuristics we do not differentiate between them. Given n points in \mathbb{R}^2 , there are n^2 topologically equivalent congruence classes $\mathcal{G} = G_1 \dots G_{n^2}$. To discretize our algorithm, we select a representative from each and denote this set by \mathcal{G}' which will be the stabbing line candidate set. In order to construct \mathcal{G}' , we build the arrangement of lines in the dual plane corresponding to P , take the center of mass of each bounded face, and translate the centers back to the primal plane. Note that if a congruence class contains both lines with positive slope and lines with negative slope, then it will have two representatives in this method; we filter one to eliminate multiplicities.

Let $E' = \{[p_1, p_2] | p_1, p_2 \in P, p_1 \neq p_2\}$. We iteratively select edges to place in D , selecting the next edge as follows: For each edge $e \in E'$, let e_L be the set of lines in \mathcal{G}' that stab e . Let D' be the current structure at some step. For each $g \in \mathcal{G}'$, let $n(D', g)$ be the number of edges in D' that are stabbed by g . Let $e_{G'}$ be the set of lines that stab e . Then, $e_{G'}$ defines a histogram on e based on the values of the set $\{n(D', g) | g \in e_{G'}\}$. At each step of the algorithm, we select the edge e with the lexicographic least histogram value (bins with bigger index are more significant in this relation). After selecting the edge, we increment the corresponding values in the histograms of all of the edges that are stabbed by lines that stab e .

In the algorithms discussed above we used E' as the candidate set for building D , and, at each step, we choose one edge from E' . To improve the running time, we use the following heuristics to decrease the size of the candidate set E' .

*Work on this paper has been partially supported by the National Science Foundation (CCR-0098172, CCF-0431030)

[†]Department of Applied Mathematics and Statistics, Stony Brook University, Stony Brook, NY 11794-3600. jsbm@ams.sunysb.edu.

[‡]Department of Computer Science, Stony Brook University, Stony Brook, NY 11794-4400. epacker@cs.sunysb.edu.

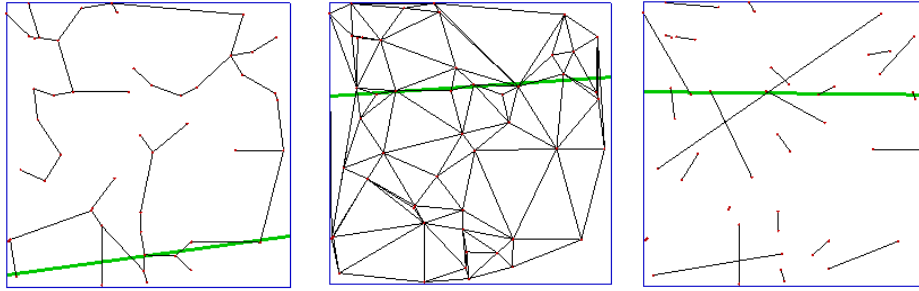


Figure 1: Results of our implementation on a set of 50 points: spanning tree, triangulation, and matching (left to right). The thick line in each subfigure is a witness to the stabbing number.

Localization. Intuitively, short edges should usually be better candidates than long ones since, usually, fewer lines stab them. Let $R(P)$ be the maximum distance between any point in P and its nearest neighbor, and let $r(P) = c * R(P)$ for some constant c (we chose $c = 5$). We modify E' as follows: $E' = \{(p_1, p_2) | p_1, p_2 \in P, |(p_1, p_2)| \leq r(P)\}$. In the case that the new E' is not sufficient to build D (e.g., if the input consists of two distant clusters, and we want to construct \mathcal{M}), we double $r(P)$ and update $E'(P)$ accordingly. Our experiments have shown that this heuristic decreases the size of E' (and the processing time) significantly, without affecting the results substantially. We denote this optimization heuristic by OPT_l .

Divide and Conquer. Using a quadtree technique, we recursively partition the space into four quadrants, compute D recursively in each, and then connect the instances of the four quadrants. We use the same ideas we discussed above for selecting edges. This idea also decreased the processing time significantly without affecting the output quality substantially. We denote this heuristic optimization by OPT_{dq} .

Another time-saving heuristic we performed was to consider a random sample of \mathcal{G}' when building D . We denote this heuristic by $OPT_s(\beta)$, for sampling probability β .

Finally, we compared our results to standard methods to compute structures on P , without regards to stabbing number: minimum (Euclidean) spanning tree, Delaunay triangulation, and greedy (Euclidean) perfect matching.

Preliminary results with the above techniques are presented in Table 1.

3 Heuristic for Lower Bound

Let $L' \subset L$ and $\mathcal{A}(L')$ be an arrangement induced by L' . Let f be any face of $\mathcal{A}(L')$ and suppose that the number of points in P that lie inside f is at most 1 (we say in this case that L' shatters P [3]). Consider the edges of any geometric network D . Since L' shatters P , each $e \in D$ is stabbed by at least one edge of L' . It follows from the Pigeonhole Principal that there is a line $l \in L'$ that stabs D at least $\lfloor \frac{n-1}{|L'|} \rfloor$ times. (Note that $n - 1$ is the number of edges in D .) Finding the min-

imum shattering set was shown to be NP-complete [3]. We compute a lower bound by finding a shattering set as follows. We greedily select lines from G' , where the greedy choice is to select a line that separates as many pairs of points in P that are still not separated by previously selected lines.

Table 2 presents preliminary results with our implementation for the minimum spanning tree. It shows that the different optimizations we performed did not harm the quality of the result much and that our heuristics outperformed the Euclidian minimum spanning tree. The table also shows that there is a gap between our lower and upper bounds. Further experimentation will attempt to estimate the growth rate of the upper/lower bound ratio.

	10	20	50	100	200
\mathcal{A} (general algorithm)	3.6	5	7.2	9.6	13
$OPT_s(0.1)$	4.8	6.8	9.4	11.2	15.2
$OPT_s(0.2)$	4.6	6.6	8.6	10.6	14.8
$OPT_s(0.3)$	4.4	6	8	10.2	14.6
$OPT_s(0.5)$	4.2	5.8	7.6	10.2	14.2
$OPT_s(0.75)$	4	6.2	7.6	10	13.8
$OPT_s(0.9)$	4	6	7.4	9.8	13.4
OPT_l	3.8	5.2	7.4	10	13.2
OPT_{dq}	3.6	5	7.6	10.4	13
MST	4	6.6	9.2	12	16.4
Lower Bound	2	2	3	4	5

Table 1: Average upper and lower bounds on the stabbing number obtained with our heuristics for spanning trees. The experiments were performed with random set of points. MST denotes the Euclidean minimum spanning tree.

References

- [1] P. K. Agarwal. Ray shooting and other applications of spanning trees with low stabbing number. *SIAM J. Comput.*, 21:540–570, 1992.
- [2] S. Fekete, M. Lbbecke, and H. Meijer. Minimizing the stabbing number of matchings, trees, and triangulations. *15th Ann. ACM-SIAM Sympos. Discrete Algorithms*, pages 430–439, 2004.
- [3] R. Freimer, J. S. B. Mitchell, and C. D. Piatko. On the complexity of shattering using arrangements. In *Proc. 2nd Canad. Conf. Comput. Geom.*, pages 218–222, Aug. 1990.

	Greedy	D and Q	Random s.t	x/y monotone	Onion
Random	8	9	11	28	15
Convex	2	2	7	23	2
X shape	4	4	8	32	26
Grid	8	9	15	29	13
Clustures	7	7	12	14	13

Table 2: Average upper and lower bounds on the stabbing number obtained with our heuristics for spanning trees. The experiments were performed with random set of points. MST denotes the Euclidean minimum spanning tree.