Computing Non-Obtuse Triangulations with Few Steiner Points

Mikkel Abrahamsen
University of Copenhagen, Denmark

Florestan Brunck
University of Copenhagen, Denmark

Jacobus Conradi
University of Bonn, Germany

Benedikt Kolbe
Hausdorff Center for Mathematics, Lamarr Institute for Machine Learning and Artificial Intelligence, University of Bonn, Germany

André Nusser
Université Côte d'Azur, CNRS, Inria, Sophia Antipolis, France

Abstract

We present the winning implementation of the Seventh Computational Geometry Challenge (CG:SHOP 2025). The task in this challenge was to find non-obtuse triangulations for given planar regions, respecting a given set of constraints consisting of extra vertices and edges that must be part of the triangulation. The goal was to minimize the number of introduced Steiner points. Our approach is to maintain a constrained Delaunay triangulation, for which we repeatedly remove, relocate, or add Steiner points. We use local search to choose the action that improves the triangulation the most, until the resulting triangulation is non-obtuse.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases non-obtuse triangulation, local search, competition

Digital Object Identifier 10.4230/LIPIcs.SoCG.2025.79

Category CG Challenge

Supplementary Material *Software (Source Code)*: `https://github.com/JacobusTheSecond/nonObtuseTri` [1], archived at `swh:1:dir:b6301970ab626c084f309f9b5e6b91e9acf24949`

Funding *Mikkel Abrahamsen*: Supported by Independent Research Fund Denmark, grant 1054-00032B, and by the Carlsberg Foundation, grant CF24-1929.
Florestan Brunck: Supported by a DNRF Chair grant from the Danish National Research Foundation and partially by the BARC grant of the Villum Foundation.
Jacobus Conradi: Funded by the iBehave Network: Sponsored by the Ministry of Culture and Science of the State of North Rhine-Westphalia.
Benedikt Kolbe: This research has partly been funded by the Federal Ministry of Education and Research of Germany and the state of North-Rhine Westphalia as part of the Lamarr-Institute for Machine Learning and Artificial Intelligence.
André Nusser: Supported by the France 2030 investment plan managed by the ANR as part of the Initiative of Excellence of Université Côte d'Azur with reference number ANR-15-IDEX-01.

Acknowledgements We thank the Challenge organizers and other competitors for their time, feedback, and making this whole event possible. We thank Max Kanold for invaluable input enabling high quality visualizations which made later discussions on how to improve our algorithm all the more fruitful. We gratefully acknowledge the access to the Marvin cluster of the University of Bonn.

1 Introduction

The Computational Geometry Challenge (CG:SHOP) is an annual competition on geometric optimization. The 2025 edition involved finding non-obtuse triangulations of a given planar straight-line graph (PSLG), a planar embedding of a graph where edges are represented as line segments; see Figure 1. The input consists of a PSLG $G = (V, E)$ and a simple polygon P with vertices in V. The goal is to output a non-obtuse triangulation $\mathcal{T}$ of P with vertices V' so that $V \subseteq V'$ and the edges of $\mathcal{T}$ cover E. The vertices $V' \setminus V$ are called *Steiner points* and the objective is to minimize them. We refer to the survey [10] for further details.

The problem of finding non-obtuse triangulations has a long history, dating back to the 1960s [6, 4, 3, 11, 13, 5]. In the 2-dimensional setting, the latest development is Bishop's algorithm – the only one to provably output a non-obtuse triangulation of any PSLG with a number of Steiner points polynomial in the vertices of the input. In general, the computational complexity of minimizing the number of Steiner points or triangles remains unknown, and the problem may be intractable already for simple polygons.

In this paper, we present our winning implementation. The instance set of the callenge consisted of 150 instances with $|V|$ and $|E|$ as large as 250. Our computed solutions were the best for 116 instances, while the runner-up team, "Gwamegi" [2], was best in 4. For the remaining 30 instances, we matched Gwamegi in number of Steiner points.

2 Algorithm

Let us first outline the idea behind our algorithm. Since a non-obtuse triangulation is a Delaunay triangulation (the local Delaunay condition is respected by the inscribed angle theorem), our algorithm maintains a (constrained) Delaunay triangulation (CDT) that gets modified successively. Whenever a obtuse triangle is present, the algorithm makes a modification by removing, relocating, or adding a Steiner point and updating the triangulation accordingly. More precisely, the algorithm operates in rounds, and in each round we consider a set of candidates for local changes and choose the most promising one. In each round, the input is the original instance and the current set of Steiner points, which have been added in the previous rounds. We compute a set of candidate actions, of which there are three types: (i) the insertion of a new Steiner point, (ii) the relocation of an existing Steiner point, and (iii) the replacement of two adjacent Steiner points with a single Steiner point. For

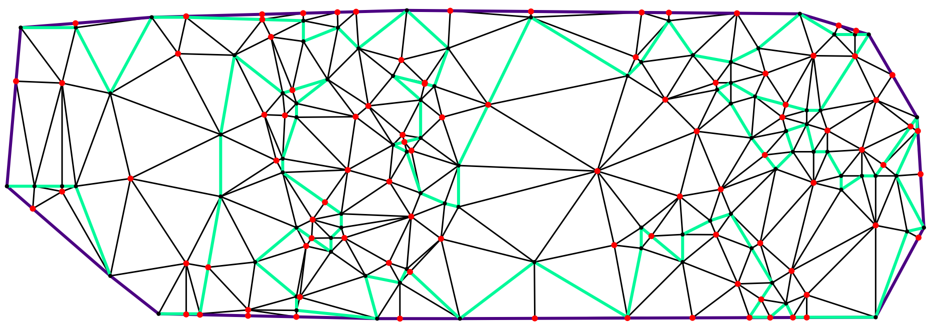


Figure 1 A non-obtuse triangulation produced by our implementation. Here, as in the rest of the paper, the region to be triangulated is enclosed by the fat curve, the given vertices are black, and the constraint edges are green. The Steiner points are red.

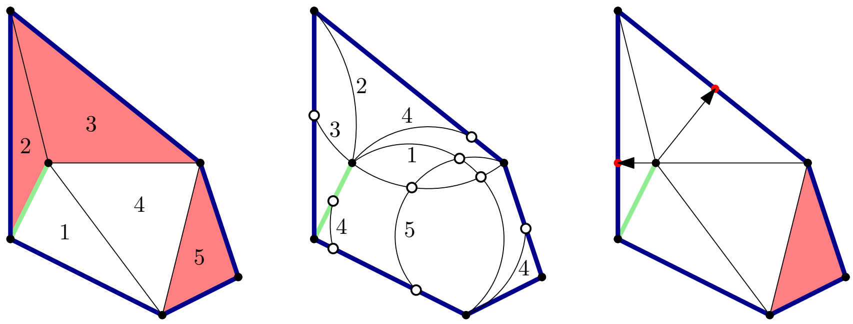


Figure 2 *left:* CDT with obtuse triangles marked in red; *middle:* its `circleArrangement`; *right:* the two `altitudeDrop` points of the two upper obtuse triangles and its resulting CDT.

each action, we compute the resulting triangulation after applying the action. Each of the resulting triangulations is evaluated using a cost function (given in Section 2.3, Equation (1)), which involves the number of Steiner points and the number of obtuse triangles of different types. The output produced by the round is the triangulation that minimizes the cost. The process terminates when the CDT has no obtuse triangles left. In particular, our algorithm produces a non-obtuse triangulation whenever it terminates.

The way in which we insert a new Steiner point is inspired by the work of Erten and Ungör [9, 8, 7]. They described a general approach to finding triangulations avoiding small or large angles, where new Steiner points are chosen as optimal points along the Voronoi diagram (see [7, Section 2.1]). We extend this idea to a visibility-constrained Voronoi diagram [14] which takes into account the constraints of the PSLG.

2.1 Primitives

We first describe the geometric primitives that we are basing our algorithm on:

- `altitudeDrop`: Given an obtuse triangle, compute the unique point on the longest side that splits the obtuse triangle into two right-angled triangles (Figure 2, right).
- `polygonCenter`: Given a simple polygon, compute a center point such that connecting each vertex of the polygon with the center point results in a non-obtuse triangulation of the polygon, or decide no such point exists. Such a point can be computed as follows: Observe that for any two points p and q, a point x forms a non-obtuse triangle pqx if and only if x lies inside the slab defined by the two hyperplanes rooted at p and q with normal $q-p$ and outside the disk centered at $\frac{p+q}{2}$ with radius $\frac{\|q-p\|}{2}$. Pick any point in the intersection of these sets defined by all consecutive pairs of vertices p and q on the boundary of the polygon (Figure 3).
- `visibilityVoronoi`: Given a CDT, compute the visibility-constrained Voronoi diagram (sometimes called bounded Voronoi diagram) [14]. (Figure 4, top right).
- `clippedCircumcircle`: Let T be a triangle in a CDT and C its circumcircle. The disk bounded by C may be partitioned into multiple pieces by the constraints of the CDT. Compute the piece of the partition containing T (Figure 4, bottom left).
- `circleArrangement`: Given a CDT, compute first the `clippedCircumcircle` for each triangle, then the arrangement of these clipped circumcircles (Figure 2, middle).

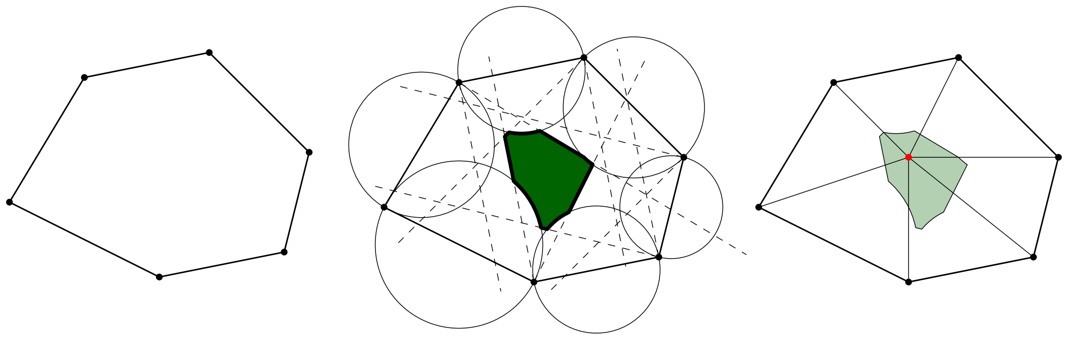


Figure 3 Illustration of the computation of the `polygonCenter` primitive.

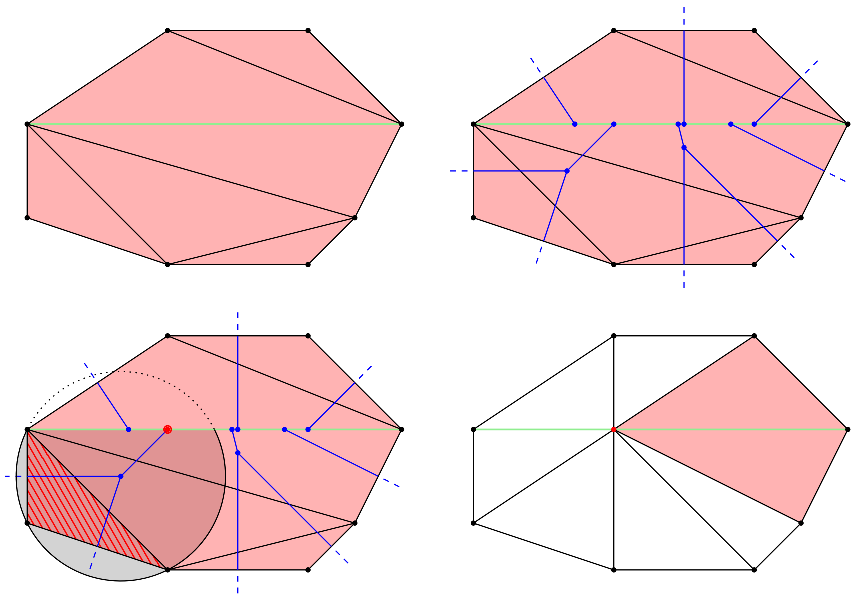


Figure 4 Center identification in the style of Erten and Ungör [9, 8, 7], from the visibility-constrained Voronoi diagram [14].

2.2 Action Generation

In this section we build on the geometric primitives we just introduced, and describe in more detail the three types of candidate actions that the algorithm evaluates for a provided CDT $\mathcal{D}$, which together form the action set $\mathcal{A}(\mathcal{D})$ of $\mathcal{D}$. Every Steiner point, triangle, and edge of $\mathcal{D}$ and every cell in the `circleArrangement` of $\mathcal{D}$ has at most one action associated to it. Every action includes two steps: the insertion, relocation or deletion of Steiner points and ensuring that the maintained triangulation is a CDT. We now describe the types of actions.

Insertion. Let T be a triangle in $\mathcal{D}$. If T is non-obtuse already, then it has no action associated to it. Hence, let T be obtuse instead. If the longest side of T (opposite the unique obtuse angle in T) is a constrained edge of $\mathcal{D}$, the candidate action associated to T adds the unique point described by the `altitudeDrop` primitive. If instead the longest side of T is not constrained, the candidate action associated to T adds the point v in the `clippedCircumcircle` of T that lies on an edge e of the `visibilityVoronoi` of $\mathcal{D}$ and maximizes the distance to the points of $\mathcal{D}$ defining e (see Figure 4). This point v

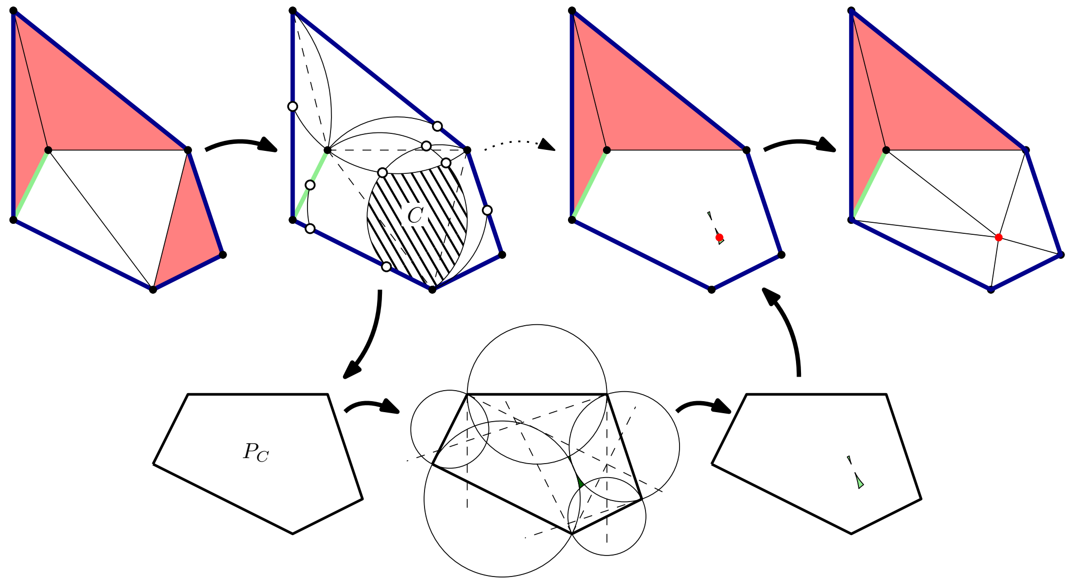


Figure 5 Center identification by choosing a cell C from the `circleArrangement` and computing the `polygonCenter` of its corresponding polygon P_C.

can efficiently be identified via a local search in the `visibilityVoronoi` diagram. We observe that v maximizes the distance to all other points in the CDT while still lying in the `clippedCircumcircle` of T. This guarantees that its insertion into the CDT results in the removal of T from the CDT, accompanied by an intuitively increased likelihood that only few angles of triangles incident to v are obtuse (c.f. [9, 8, 7]).

Next, let C be a cell in the `circleArrangement`. Let $\{T_1, \ldots\}$ be the set of all triangles whose `clippedCircumcircle` contains C. By classical Delaunay arguments, the union of triangles in $\{T_1, \ldots\}$ defines a polygon P_C. We note that the CDT resulting from adding a point $v \in C$ is obtained by replacing the edges of $\mathcal{D}$ in P_C with the segments vp for every vertex p of P_C. In particular, if at least one triangle in $\{T_1, \ldots\}$ is obtuse and the `polygonCenter` v of P_C lies in C, then adding v to the CDT ensures that the number of obtuse triangles in the CDT strictly decreases. Thus, if at least one T_i is obtuse and the `polygonCenter` of P_C exists, the candidate action associated to C adds this `polygonCenter` (see Figure 5), otherwise no action is associated to C.

Relocation. Let v be a Steiner point of $\mathcal{D}$. If v is not part of an obtuse triangle, v has no action associated to it. Hence, let v be part of some obtuse triangle instead. Let P be the polygon resulting from the union of all triangles that v is a part of. Assume the `polygonCenter` of P exists, otherwise v has no action associated to it. The action associated to v moves v to the `polygonCenter` of P.

Deletion. Let e be an edge in $\mathcal{D}$ defined by two Steiner points v and w. The union of all triangles that v or w are a part of forms a polygon P. Assume the `polygonCenter` of P exists, otherwise e has no action associated to it. The action associated to e removes the two points v and w from $\mathcal{D}$ and adds the `polygonCenter` of P.

2.3 Evaluation

The evaluation function of a CDT $\mathcal{D}$ consists of two parts: A naive evaluation function and an algorithm that improves the accuracy of the naive evaluation.

Naïve evaluation. We say an obtuse triangle T in $\mathcal{D}$ *supersedes* another obtuse triangle T' in $\mathcal{D}$ if the longest side of T is either of the shorter two sides of T' (Figure 6 shows superseded-superseding triangle pairs). We expect superseded triangles to be influenced by whatever action is associated to its superseding triangle. An example of this phenomenon can be seen in Figure 4. Hence, we argue that handling non-superseded obtuse triangles should be prioritized and therefore the cost of non-superseded obtuse triangles should be higher than that of superseded (obtuse) triangles. The evaluation function we use is

$$\mathrm{eval}(\mathcal{D}) \coloneqq \#\text{Steiner points} + 1.1 \cdot \#\text{superseded } \triangle\text{'s} + 3.1 \cdot \#\text{non-superseded } \triangle\text{'s}. \tag{1}$$

Figure 6 shows a run of the algorithm, including the actions that were applied and the value of the evaluation function. Note that we did not pursue optimizing the constants in (1).

Probabilistic state-graph evaluation. To improve the rather naïve evaluation function, we endow the evaluation function with a depth parameter $k \geq 0$, akin to the notion of minimax searching [12] over a game-state graph of a single player game, and define

$$\mathrm{eval}_k(\mathcal{D}) \coloneqq \begin{cases} \min_{A \in \mathcal{A}(\mathcal{D})} \mathrm{eval}_{k-1}(\mathcal{D}_A) & \text{if } k > 0 \\ \mathrm{eval}(\mathcal{D}) & \text{if } k = 0 \end{cases} \tag{2}$$

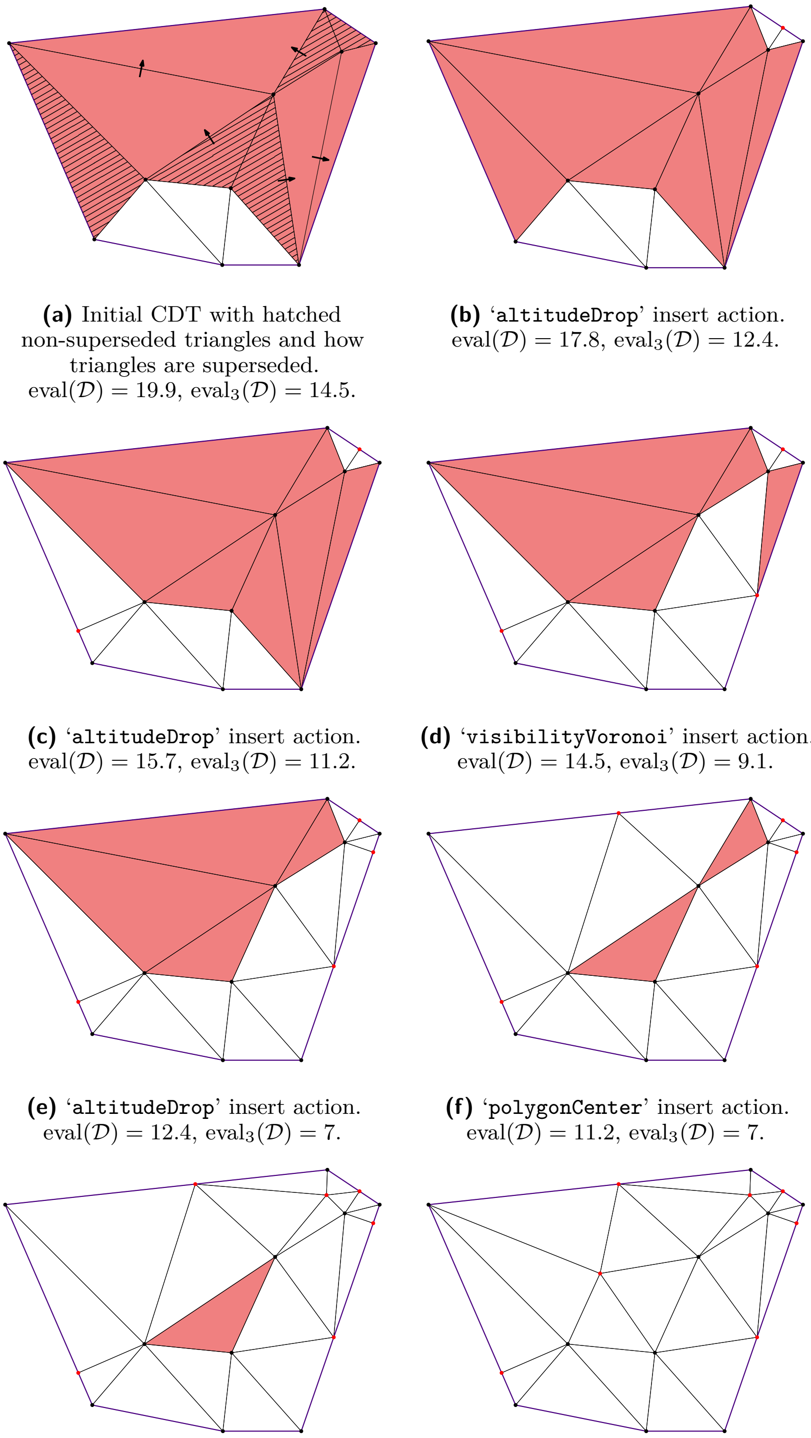
where $\mathcal{D}_A$ is the CDT resulting from $\mathcal{D}$ when applying the action A from the action set $\mathcal{A}(\mathcal{D})$. Even after extensive optimization efforts, we found that even with k as small as 3 the execution time was quite slow. Hence, instead of evaluating $\mathrm{eval}_{k-1}(\mathcal{D}_A)$ for every A in the action set of $\mathcal{D}$, we randomly draw a sample S of actions from $\mathcal{A}(\mathcal{D})$ (with a bias towards more promising moves determined by $\mathrm{eval}_0(\mathcal{D}_A)$) computing $\min_{A \in S} \mathrm{eval}_{k-1}(\mathcal{D}_A)$.

2.4 Solution Merging

We observed that, due to the probabilistic nature of the evaluation function, different solutions to the same instance often had a similar number of Steiner points but looked vastly different locally. This motivates merging different solutions by sampling a circle and replacing every Steiner point inside this circle from one solution with the Steiner points of the other solution. Note that merging two solutions might result in a CDT that has obtuse triangles. Thus, we apply our algorithm to these CDTs obtaining a new solution with potentially less Steiner points. Repeated application of this merging step reduced the number of Steiner points by as much as 30% for some instances (see Figure 7).

3 Implementation, Experiments, and Examples

Code and Hardware. Our implementation is in python. In the later stages of the competition we ran our solver on ten nodes of the Marvin cluster of the University of Bonn, each with 1024GB of RAM and 96 threads at 2.10GHz using around 300 000 CPU hours. We did not find any improved solution in the last two weeks of the competition.



(a) Initial CDT with hatched non-superseded triangles and how triangles are superseded. $\mathrm{eval}(\mathcal{D}) = 19.9$, $\mathrm{eval}_3(\mathcal{D}) = 14.5$.

(b) '`altitudeDrop`' insert action. $\mathrm{eval}(\mathcal{D}) = 17.8$, $\mathrm{eval}_3(\mathcal{D}) = 12.4$.

(c) '`altitudeDrop`' insert action. $\mathrm{eval}(\mathcal{D}) = 15.7$, $\mathrm{eval}_3(\mathcal{D}) = 11.2$.

(d) '`visibilityVoronoi`' insert action. $\mathrm{eval}(\mathcal{D}) = 14.5$, $\mathrm{eval}_3(\mathcal{D}) = 9.1$.

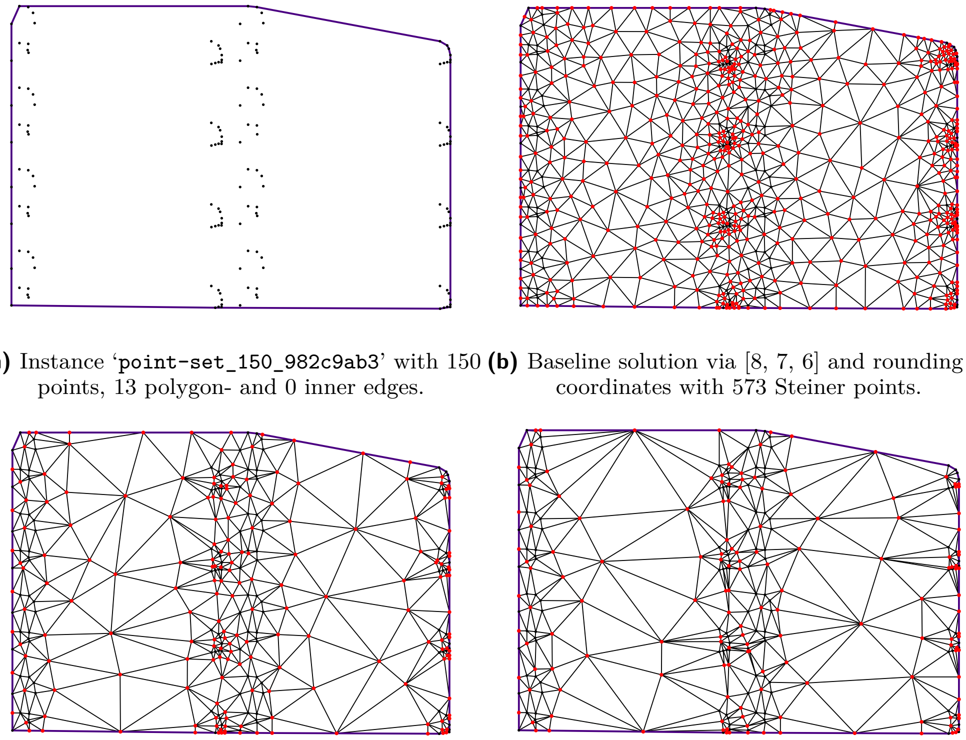
(e) '`altitudeDrop`' insert action. $\mathrm{eval}(\mathcal{D}) = 12.4$, $\mathrm{eval}_3(\mathcal{D}) = 7$.

(f) '`polygonCenter`' insert action. $\mathrm{eval}(\mathcal{D}) = 11.2$, $\mathrm{eval}_3(\mathcal{D}) = 7$.

(g) '`polygonCenter`' insert action. $\mathrm{eval}(\mathcal{D}) = 9.1$, $\mathrm{eval}_3(\mathcal{D}) = 7$.

(h) Terminal '`polygonCenter`' insert action. $\mathrm{eval}(\mathcal{D}) = 7$, $\mathrm{eval}_3(\mathcal{D}) = 7$.

Figure 6 Maintained CDT of instance '`point-set_10_c04b0024`' (consisting of 10 points, 6 polygon edges and 0 inner edges) throughout the execution of the local search with depth 0, with all obtuse triangles marked in red. Execution time: 10 seconds. Final number of Steiner points: 7.



(a) Instance '`point-set_150_982c9ab3`' with 150 points, 13 polygon- and 0 inner edges.

(b) Baseline solution via [8, 7, 6] and rounding of coordinates with 573 Steiner points.

(c) Best found solution via probabilistic local search and depth 3 with 190 Steiner points.

(d) Solution via merging and re-solving of existing solutions with 151 Steiner points.

Figure 7 Solution comparison of instance '`point-set_150_982c9ab3`'.

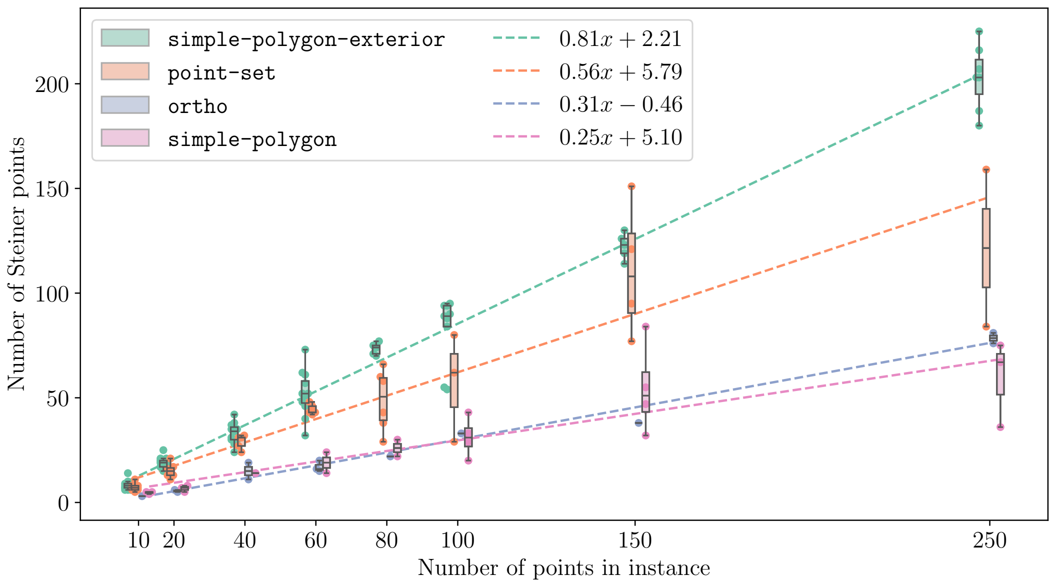


Figure 8 Relationship between solution size and input complexity over different instance types.

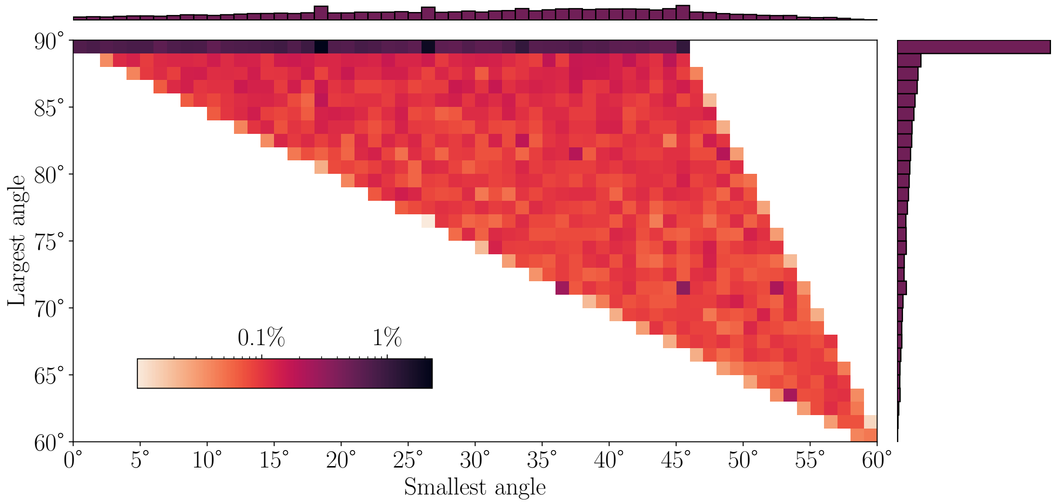


Figure 9 Average distribution of smallest and largest angles, of all triangles in all solutions.

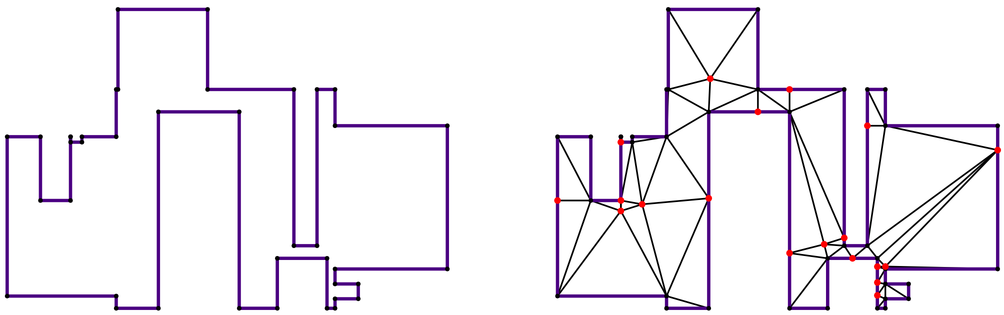


Figure 10 Instance '`ortho_40_56a6f463`' and a solution with 19 Steiner points.

Solutions. The provided instances fall into various types: (i) `ortho`, consisting of rectilinear simple polygons (Figure 10), (ii) `simple-polygon`, consisting of simple polygons (Figure 11), (iii) `point-set`, consisting of the convex polygons with points in its interior (Figures 13 and 14), and (iv) `simple-polygon-exterior`, consisting of convex polygons with points and additional constraints in its interior (Figures 12 and 15). Our algorithm performs well on all instances types; no manual intervention or distinction is necessary for our approach. The number of Steiner points in our solutions compared to the number of input points in each type suggests that the total number of Steiner points is roughly linear in the input complexity and increases depending on the number of points and constraints in the interior of the PSLG, see Figure 8. We observe that a large fraction of triangles computed by our algorithm are right triangles (see Figure 9, right). All other angles in our solutions (determined by largest and smallest angle) appear to be roughly equally distributed (see Figure 9).

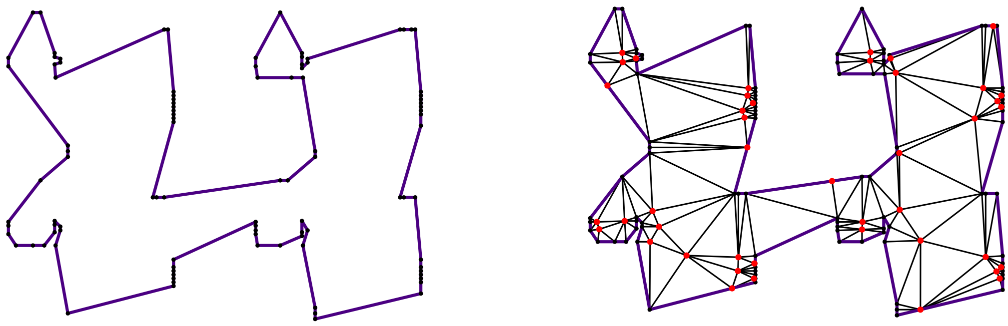


Figure 11 Instance ‘`simple-polygon_100_cb23308c`’ and a solution with 43 Steiner points.

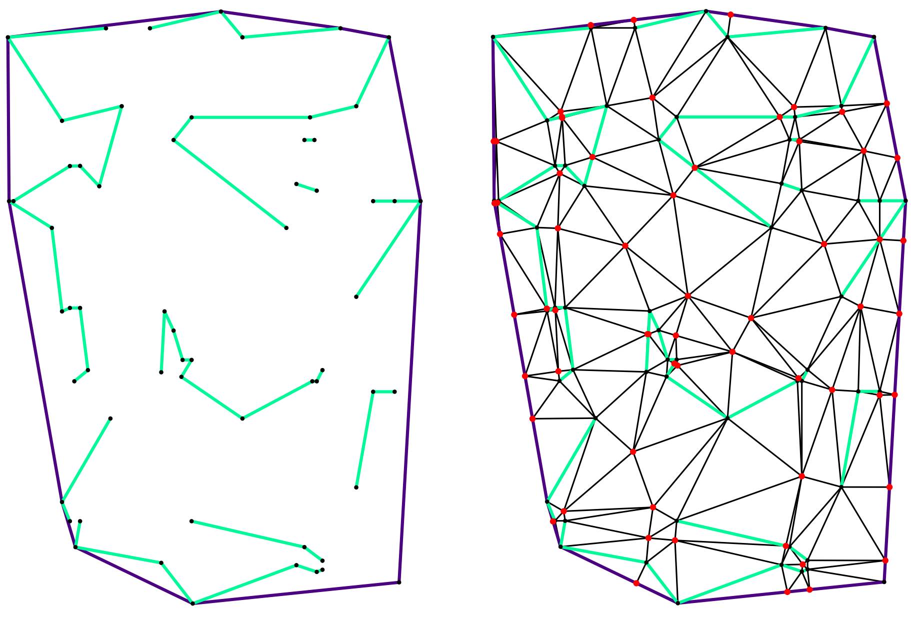


Figure 12 Solution to ‘`simple-polygon-exterior-20_60_8221c868`’ with 62 Steiner points.

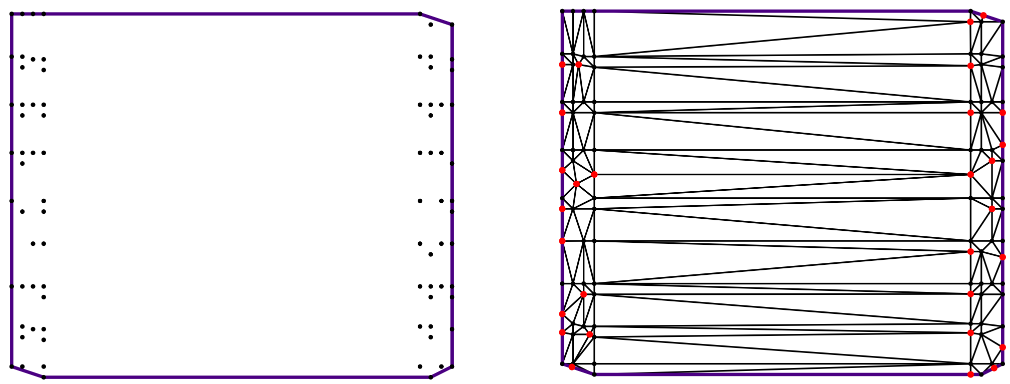


Figure 13 Instance '`point-set_80_9a8373fb`' and a solution with 29 Steiner points.

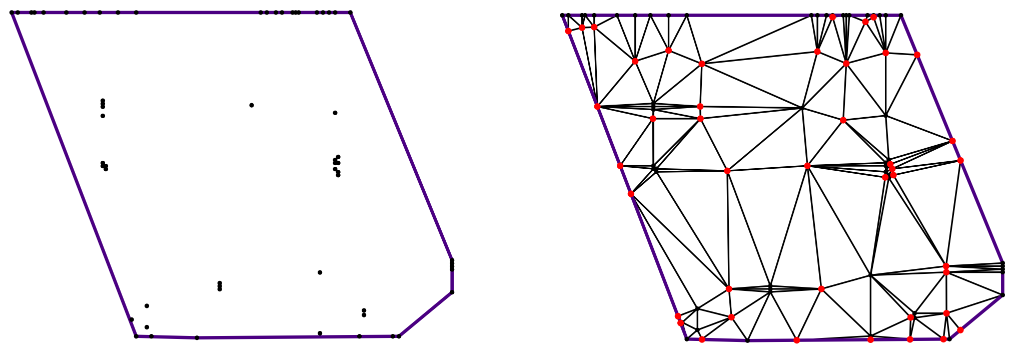
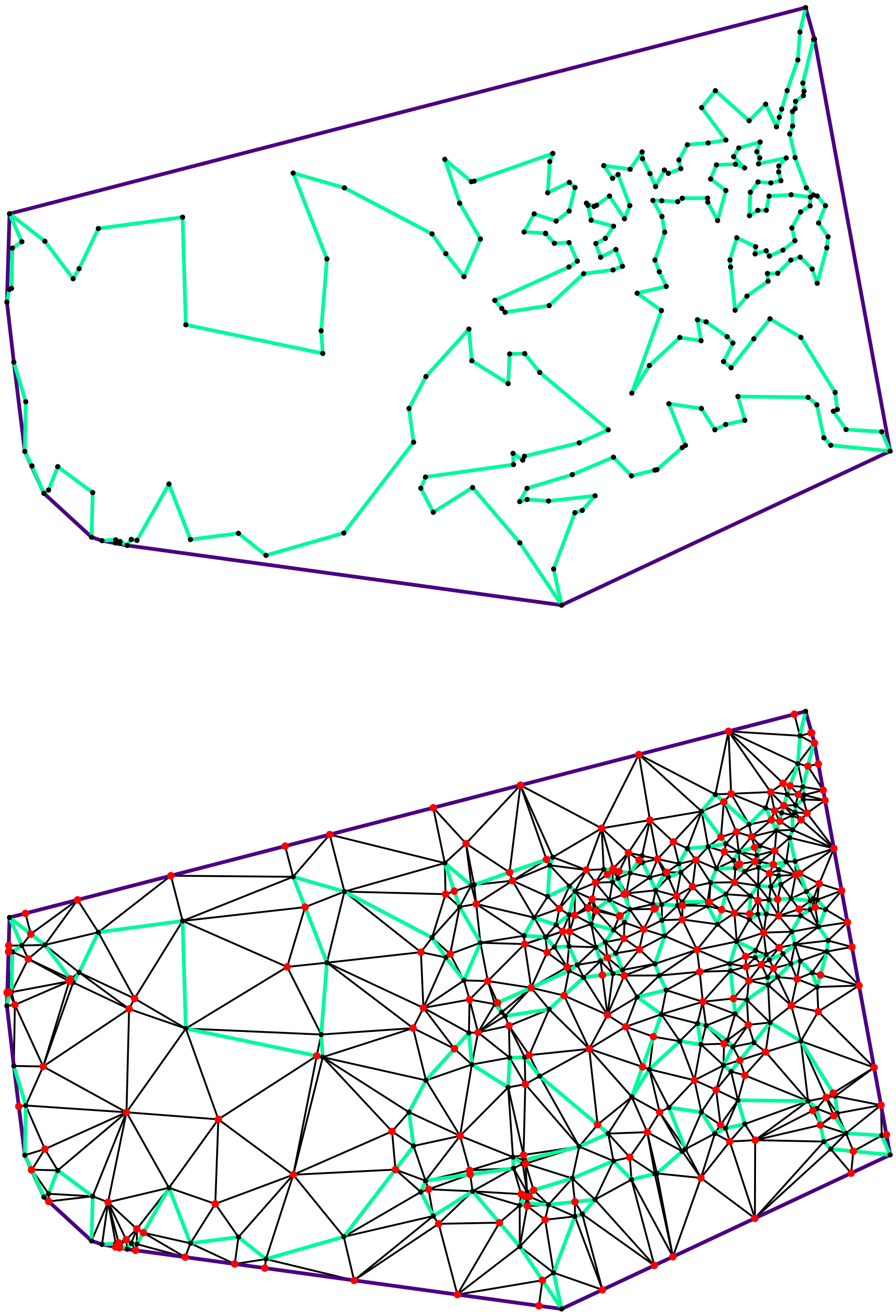


Figure 14 Instance '`point-set_60_27bc003d`' and a solution with 43 Steiner points.



■ **Figure 15** Solution with 225 Steiner points to ‘simple-polygon-exterior_250_c0a19392’.

References

1 Mikkel Abrahamsen, Florestan Brunck, Jacobus Conradi, Benedikt Kolbe, and André Nusser. nonObtuseTri. Software, version v3.0., swhId: `swh:1:dir:b6301970ab626c084f309f9b5e6b91e9acf24949` (visited on 2025-06-02). URL: `https://github.com/JacobusTheSecond/nonObtuseTri`, `doi:10.4230/artifacts.23288`.

2 Taehoon Ahn, Jaegun Lee, Byeonguk Kang, and Hwi Kim. Incremental algorithm and local search for minimum non-obtuse triangulations (CG challenge). In Oswin Aichholzer and Haitao Wang, editors, *41st International Symposium on Computational Geometry (SoCG 2025)*, 2025. These proceedings. `doi:10.4230/LIPIcs.SoCG.2025.80`.

3 Brenda S. Baker, Eric Grosse, and Conor S. Rafferty. Nonobtuse triangulation of polygons. *Discret. Comput. Geom.*, 3:147–168, 1988. `doi:10.1007/BF02187904`.

4 Marshall W. Bern, Scott A. Mitchell, and Jim Ruppert. Linear-size nonobtuse triangulation of polygons. *Discret. Comput. Geom.*, 14(4):411–428, 1995. `doi:10.1007/BF02570715`.

5 Jan Brandts, Sergey Korotov, Michal Krízek, and Jakub Solc. On nonobtuse simplicial partitions. *SIAM Rev.*, 51(2):317–335, 2009. `doi:10.1137/060669073`.

6 Y. Burago and V. Zalgaller. Polyhedral embedding of a net. *Vestnik Leningrad. Univ*, 15(7):66–80, 1960.

7 Hale Erten and Alper Üngör. Computing acute and non-obtuse triangulations. In Prosenjit Bose, editor, *Proceedings of the 19th Annual Canadian Conference on Computational Geometry, CCCG 2007, August 20-22, 2007, Carleton University, Ottawa, Canada*, pages 205–208. Carleton University, Ottawa, Canada, 2007. URL: `http://cccg.ca/proceedings/2007/09a2.pdf`.

8 Hale Erten and Alper Üngör. Computing triangulations without small and large angles. In Francois Anton, editor, *Sixth International Symposium on Voronoi Diagrams, ISVD 2009, Copenhagen, Denmark, June 23-26, 2009*, pages 192–201. IEEE Computer Society, 2009. `doi:10.1109/ISVD.2009.32`.

9 Hale Erten and Alper Üngör. Quality triangulations with locally optimal steiner points. *SIAM J. Sci. Comput.*, 31(3):2103–2130, 2009. `doi:10.1137/080716748`.

10 Sándor P. Fekete, Phillip Keldenich, Dominik Krupke, and Stefan Schirra. Minimum non-obtuse triangulations: The CG:SHOP Challenge 2025, 2025. URL: `https://arxiv.org/abs/2504.04412`.

11 Hiroshi Maehara. Acute triangulations of polygons. *European Journal of Combinatorics*, 23(1):45–55, 2002. `doi:10.1006/EUJC.2001.0531`.

12 M. Maschler, S. Zamir, and E. Solan. *Game Theory*. Cambridge University Press, 2013. URL: `https://books.google.de/books?id=lqwzqgvhwXsC`.

13 S. Saraf. Acute and nonobtuse triangulations of polyhedral surfaces. *European Journal of Combinatorics*, 30(4):833–840, 2009. `doi:10.1016/J.EJC.2008.08.004`.

14 Jane Tournois, Pierre Alliez, and Olivier Devillers. Interleaving delaunay refinement and optimization for 2d triangle mesh generation. In Michael L. Brewer and David L. Marcum, editors, *Proceedings of the 16th International Meshing Roundtable, October 14-17, 2007, Seattle, Washington, USA, Proceedings*, pages 83–101. Springer, 2007. `doi:10.1007/978-3-540-75103-8_5`.