Computing Low-Cost Convex Partitions for Planar Point Sets with Randomized Local Search and Constraint Programming

Da Wei Zheng
Department of Computer Science, University of British Columbia, Vancouver, Canada
zhengdw@cs.ubc.ca

Jack Spalding-Jamieson
Department of Computer Science, University of British Columbia, Vancouver, Canada
jacketsj@alumni.ubc.ca

Brandon Zhang
Department of Computer Science, University of British Columbia, Vancouver, Canada
brandon.zhang@alumni.ubc.ca

Abstract

The Minimum Convex Partition problem (MCP) is a problem in which a point-set is used as the vertices for a planar subdivision, whose number of edges is to be minimized. In this planar subdivision, the outer face is the convex hull of the point-set, and the interior faces are convex. In this paper, we discuss and implement the approach to this problem using randomized local search, and different initialization techniques based on maximizing collinearity. We also solve small instances optimally using a SAT formulation. We explored this as part of the 2020 Computational Geometry Challenge, where we placed first as Team UBC.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases convex partition, randomized local search, planar point sets

Digital Object Identifier 10.4230/LIPIcs.SoCG.2020.83

Category CG Challenge

Related Version A description of the 2020 CG:SHOP Challenge with related work and overall outcomes can be found at [4] `https://arxiv.org/abs/2004.04207`.

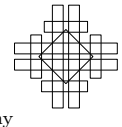
Acknowledgements We want to thank Sam Bayless for help with MonoSAT and constraint programming.

1 Introduction

For a point set P, a convex partition is a planar subdivison with vertex set P, including all the edges of the convex hull of P, such that all interior faces are convex. In the Minimum Convex Partition problem, the number of edges, or equivalently the number of faces, in a convex partition of P is minimized. When collinear points are allowed, this problem has been shown to be NP-hard [5].

One example of convex partitions is the family of triangulations. In fact this is the family of maximum convex partitions, so all other convex partitions can be compared to triangulations. We can also further conclude that any convex partition is a subset of some triangulation.

In this paper, we explore practical solutions to this problem, in the case of collinear points being allowed, and points having integer coordinates. We explored this during the 2020 Computational Geometry Challenge (CG:SHOP 2020), which was made to encourage the exploration of this problem [4]. We placed first in the challenge as Team UBC, and our experimental results in this paper are based on the instances provided.



We begin by briefly describing a simple constraint solving approach for small instances. We then discuss the details of a randomized local search approach. The details are divided into search operations, initialization techniques, and finally metaheuristics. Finally, we will discuss our experimental testing and results.

2 Algorithmic methods

2.1 Approaches for the Small Instances

We quickly solved the smallest instances (with ≤ 50 points) with a simple SAT formulation: For every pair of points in a point-set, create a variable representing whether the corresponding edge is enabled, i.e. it appears in the convex partition. In a convex partition, each interior vertex has what we refer to as the standard convexity constraint: For each edge coming into the vertex, and adjacent face, there should be another edge leaving the vertex touching the same face, such that the interior angle of the face at that vertex is at most 180 degrees. We can add this convexity constraint to the SAT formulation: For every enabled edge adjacent to an interior vertex, require that at least one of the edges within a 180 degree window in each direction is also enabled. Additionally, require that each vertex has degree at least 2, and that all convex hull edges are enabled. In addition to the encoded convexity constraint, we also require that for every two edges that intersect, at least one edge is disabled. Lastly, disable any edge that intersects a point in the point-set. The size of this formulation is $O(n^4)$ for a point-set with n vertices. The number of enabled edges in this formulation can be minimized using a MaxSAT solver. We used the MaxSAT solver UWrMaxSAT [2].

We also attempted a formulation with the geometry tools in MonoSAT [3], although it was not efficient enough to solve any instances.

2.2 Randomized local search

The main approach we used was a simple greedy local search starting from a valid convex partition. The search was designed for iterations to be as fast as possible. An edge that can be removed while preserving the convexity of its bordering faces is greedily removed. The removed edge is selected uniformly at random among all edges that can be removed. If no edge can be removed, a random edge is selected for a *rotation*. A rotation of an edge (u, v) around the point u can be performed when the edge is required to satisfy the vertex convexity constraint at u but not necessarily at v. This edge is removed and replaced with an edge (u, a) or (u, b) if these do not violate the convexity constraints at u (where a and b are the points adjacent to v on the same face as u). The direction of rotation is chosen at random. An edge for which this operation can be performed without violating the vertex convexity constraint is called *rotatable*.

2.3 Fast operations for high number of iterations

For this randomized local search strategy to be successful, we needed to be able to run many iterations quickly.

At each iteration, we sample at random a half-edge (an edge and a pivot vertex) that is rotatable in one or both directions. To sample valid edges quickly, we maintain a list of rotatable half-edges. After each rotation of an edge or deletion of an edge, we only change the rotation status of the neighbouring edges (sharing both a vertex and a face) of the rotation or deletion. This is because this operation only changes the local angular properties of the edges that are adjacent to the rotatable edge on the two faces that the edge was on.

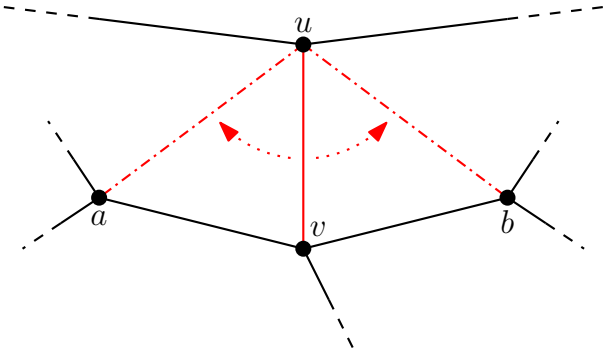


Figure 1 The edge (u, v) can rotate to either (u, a) or (u, b) in one iteration.

We managed to do an average of 20000 iterations per second on large instances. In our implementation, one iteration took $O(\log n)$ operations. It could have been written in expected $O(1)$ operations per iteration by making use of the expected (and practically) low degree of each vertex, but we did not explore this during the competition.

2.4 Initialization

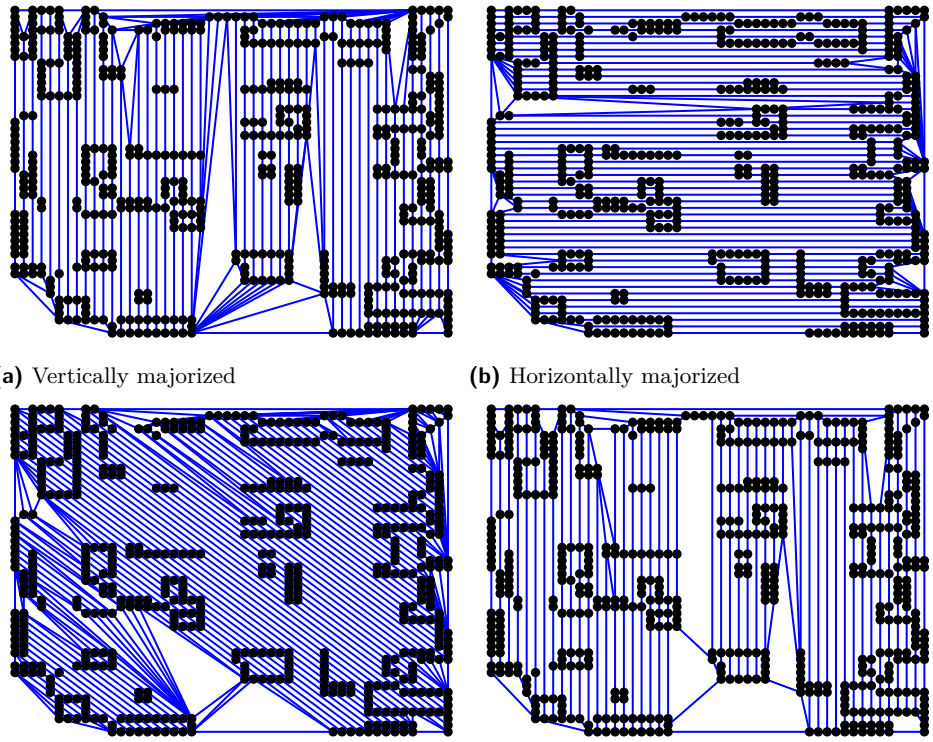
The local search strategy we employed is fairly sensitive to initialization and to the random choices that were made. To mitigate this, we spent some effort finding different initializations. We found that the best solutions for most instances were obtained by starting the local search from the Delaunay triangulation.

The second set of instances included dense point sets in a small grid, and had a large number of collinear points. With collinear points, it is possible to obtain convex partitions with a large number of degree two vertices that have edges on opposite sides, which is what we did. To do well on these instances we joined all vertices with the same x-coordinate in a chain. For the top and bottom ends of each chain we joined them with an algorithm similar to the monotone chain algorithm [1] for convex hulls where we add every edge the monotone chain algorithm considers to create a valid convex partition. Figure 2(a) is an example of such an instance.

We also generalized this to arbitrary slopes, by rotating the points before running the monotone chain subroutine. 2(b) and (c) are examples of this. We call this *majorization* with respect to a slope. We tried to initialize the local search with majorized initializations using commonly occuring slopes we found by sampling random pairs of points. However these other initializations did not improve our results.

2.5 Metaheuristics

To mitigate the sensitivity of the solver to the random choices it makes, we restarted an instance from a randomly chosen initial configuration if the objective value of the solution did not improve for $8n \log^2 n$ iterations, where n is the number of points in the instance. We chose a value that was $\Omega(n \log^2 n)$ to ensure that every edge would be moved at least once with high probability, as we sample the $O(n)$ rotatable half-edges with uniform probability.



(a) Vertically majorized

(b) Horizontally majorized

(c) Slope -1 majorized

(d) Final result after local search

Figure 2 Images of convex partitions of the `rop0000548` instance.

3 Practical computation

3.1 Computational envorinment

We performed computations on one of the shared UBC undergrad CS servers, which has two 32-core Intel Xeon E5-2698 v3 CPUs running at 2.30 GHz. After some initial testing of the algorithm, we ran the local search continuously on all instances for about 16 days, for a total of approximately 2.8 years of CPU time.

3.2 Experimental results

We tested our implementations using the instances provided for CG:SHOP 2020. Here we provide some analysis on the following groups of instances:

- `euro-night` and `us-night`: Randomly sampled points from an illumination map of Europe/the US at night.
- `uniform`: Uniformly randomly generated points.
- `rop`: Instances with many orthogonally collinear points in a bounded grid.

For each convex partition of an instance, a score can be computed according to the following formula:

$$\text{score} = \frac{\text{number of edges in convex partition}}{\text{number of edges in triangulation}}$$

As was mentioned in the introduction, the number of edges in the triangulation is maximum, so the score will be a value between 0 and 1. Smaller scores are better.

3.3 Performance on small instances

All the instances up to size 100 were run using the MaxSAT solver UWrMaxSAT [2], which allows the instances to be solved optimally. The largest instance to be completely solved within a 600 second time limit had 45 points. Most of the instances with at most 45 points completed in under a second. During the contest, there was no strict time limit for the MaxSAT solver, and a total of 70 instances were solved optimally with the SAT formulation before they were later matched by our local search methods. The largest of these had 100 points.

Our local search methods eventually matched the answers we obtained from running MaxSAT. The longest any of these took to match the answers was 65142 iterations, with 20 restarts. Most took under a second and less than 20000 iterations, and used no restarts.

3.4 Performance on uniform instances

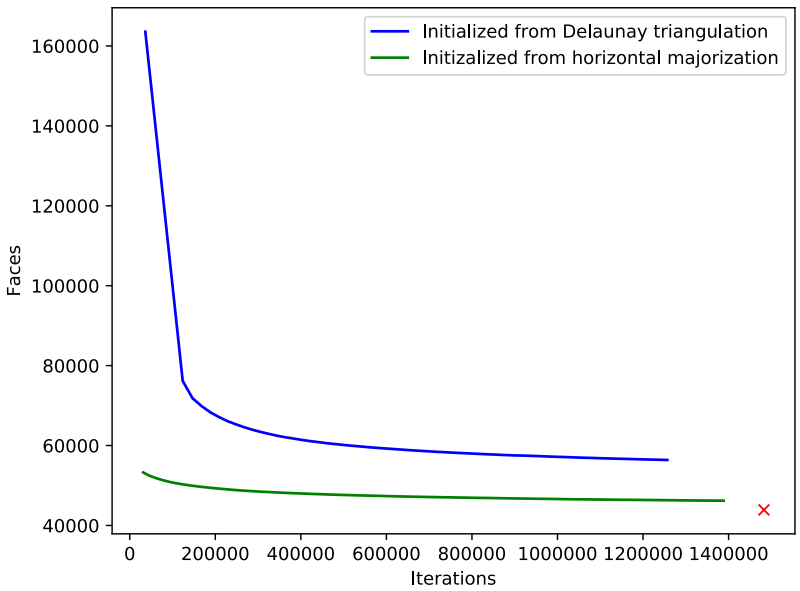


Figure 3 Performance of our algorithm with different initializations on the instance `euro-night-0100000`. Both lines depict 1 minute of runtime of our implementation. The red x-mark denotes our final score on the instance.

Uniform point sets should approximate general position point sets. For general position point sets it can be shown with Euler's characteristic that the maximum score achievable is asymptotically $\frac{1}{2}$. On large point sets, we are approaching this limit.

3.5 Performance on `rop`

We were able to achieve much better performance on `rop` instances by first horizontally or vertically majorizing the instance as initialization to the randomized local search. This took advantage of the many collinear points in the instance, as most of them had very small ranges of x and y coordinates.

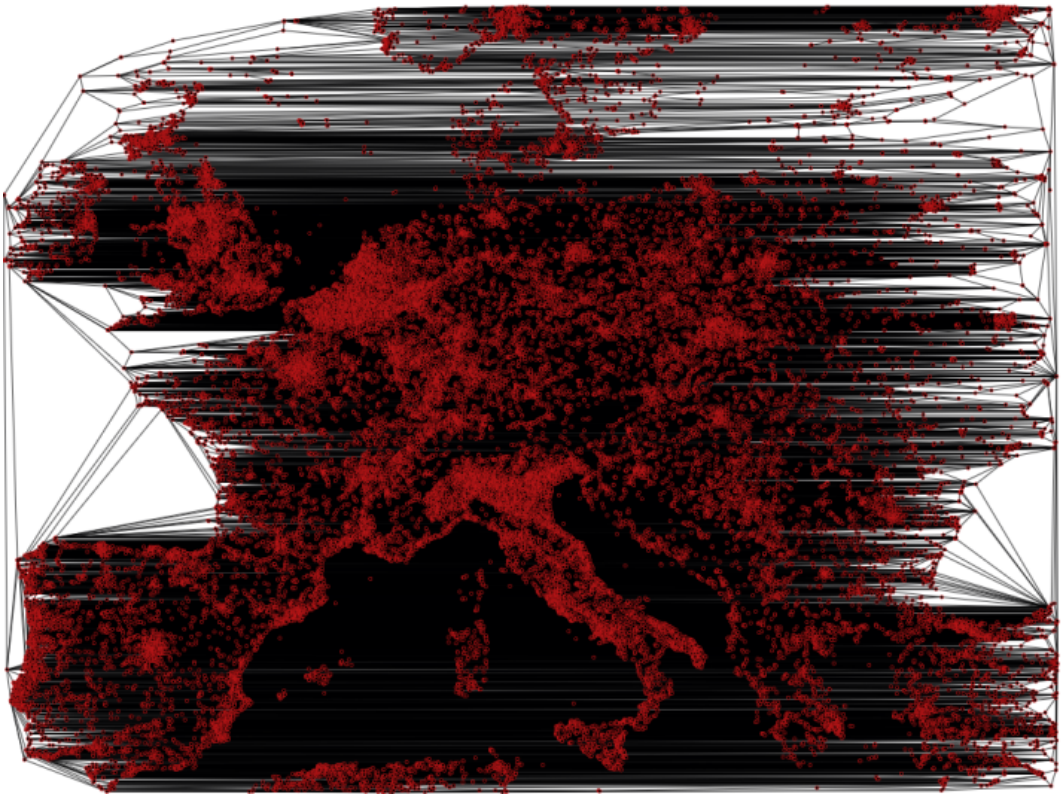


Figure 4 Our final solution to `euro-night-0040000` found by running randomized local search from horizontally majorized initialization.

3.6 Improved performance on large `euro-night` and `us-night`

We obtained better scores on large `euro-night` and `us-night` instances than on large `uniform` instances by initializing our randomized local search with horizontal majorization. This can be seen in Figure 5. This is an artifact of how the test data for the contest was created: The uniform instances have coordinates sampled from the range $[0, 6000000]$. However, the `us-night` and `euro-night` instances have coordinates in much smaller ranges: The `us-night-0100000` instance has y coordinates in the range $[0, 76956]$, and x coordinates in the range $[4, 136766]$, while the largest `euro-night` instance of 100000 points has y coordinates in the range $[0, 57598]$, and x coordinates in the range $[8, 102392]$. These instances were points sampled from a distribution based on an illumination map. As a result of the restriction to integer points, and these very small bounds for the integers, the large `us-night` and `euro-night` instances had lots of points with the same x or y coordinate, and hence lots of collinear points. By majorizing these instances, we were able to successfully take advantage of this. For example, our solution for `euro-night-0100000` has 36820 vertices with degree 2, for a total of 143883 edges and 43884 faces. We believe that with more clever methods to leverage collinear points, these solutions can improved even further.

4 Open questions

Are there other modifications of our local search strategy that can perform well, or even better? During the competition, we tried various methods of adding edges and simulated annealing based on total length of the edges in the convex partition, but we were unable to do any better with these methods.

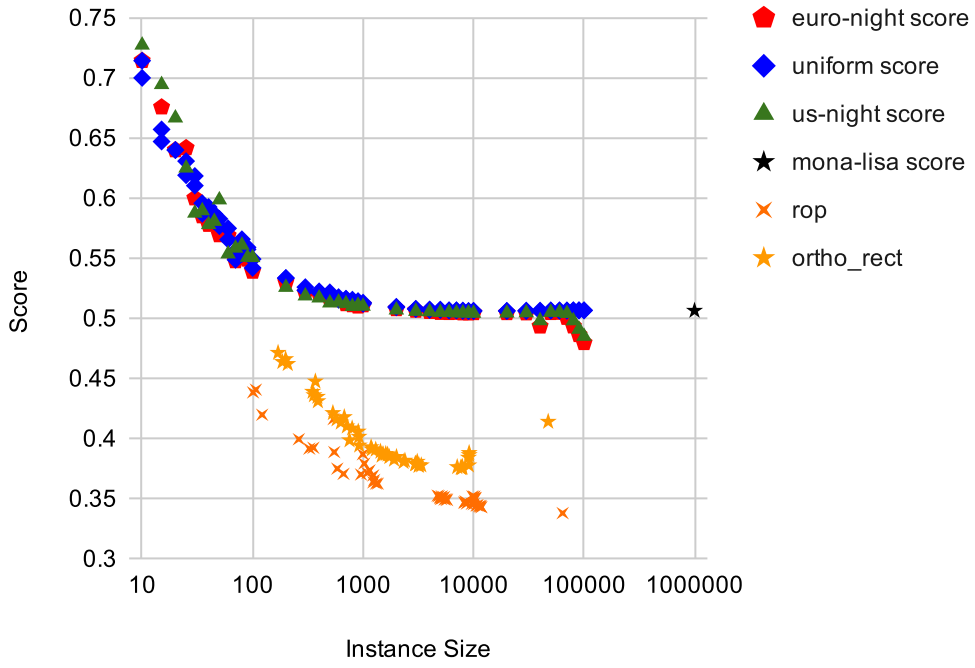


Figure 5 A plot of score vs number of points in the instance, where the score is defined as the number of edges over the number of edges in a triangulation.

Our algorithm local search strategy was very simple, yet it was able to achieve very good results on all instances. On almost uniform point sets, it was able to approach the theoretical limit of $\frac{1}{2}$. Is it possible to prove theoretical guarantees about the performance of this algorithm on random point sets or in general? We conjecture that on large random point sets, the performance of our local search algorithm approaches 0.5 in score.

References

1 A.M. Andrew. Another efficient algorithm for convex hulls in two dimensions. *Information Processing Letters*, 9(5):216–219, 1979. doi:10.1016/0020-0190(79)90072-3.

2 Fahiem Bacchus, Matti Järvisalo, and Ruben Martins, editors. *MaxSAT Evaluation 2019: Solver and Benchmark Descriptions*. Department of Computer Science Report Series B. Department of Computer Science, University of Helsinki, Finland, 2019.

3 Sam Bayless, Noah Bayless, Holger H. Hoos, and Alan J. Hu. SAT Modulo Monotonic Theories. *Proceedings of the 29th AAAI Conference on Artificial Intelligence*, 2015.

4 Erik D. Demaine, Sándor P. Fekete, Phillip Keldenich, Joseph S. B. Mitchell, and Dominik Krupke. Computing convex partitions for point sets in the plane: The cg:shop challenge 2020, 2020. arXiv:2004.04207.

5 Nicolas Grelier. Minimum convex partition of point sets is np-hard, 2019. arXiv:1911.07697.