Computing Low-Cost Convex Partitions for Planar Point Sets Based on Tailored Decompositions

Günther Eder
Universität Salzburg, FB Computerwissenschaften, Austria
geder@cs.sbg.ac.at

Martin Held
Universität Salzburg, FB Computerwissenschaften, Austria
held@cs.sbg.ac.at

Stefan de Lorenzo
Universität Salzburg, FB Computerwissenschaften, Austria
slorenzo@cs.sbg.ac.at

Peter Palfrader
Universität Salzburg, FB Computerwissenschaften, Austria
palfrader@cs.sbg.ac.at

Abstract

Our work on minimum convex decompositions is based on two key components: (1) different strategies for computing initial decompositions, partly adapted to the characteristics of the input data, and (2) local optimizations for reducing the number of convex faces of a decomposition. We discuss our main heuristics and show how they helped to reduce the face count.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Computational Geometry, geometric optimization, algorithm engineering, convex decomposition

Digital Object Identifier 10.4230/LIPIcs.SoCG.2020.85

Category CG Challenge

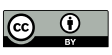
Supplementary Material The source code of our tools and heuristics is available at GitHub and can be used freely under the GPL(v3) license: `https://github.com/cgalab`.

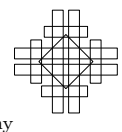
Funding Work supported by Austrian Science Fund (FWF): Grants ORD 53-VO and P31013-N31.

1 Introduction

The task of the 2020 Computational Geometry Challenge – called Challenge in the sequel for the sake of brevity – was to compute minimum convex decompositions (MCD) of point sets in the plane. We refer to the survey by Demaine et al. [2] for background information.

We employed several tools and heuristics to tackle the Challenge. All tools submitted their solutions to a central database of ours, such that tool A could query and then improve on solutions obtained by tool B, and vice versa. Most of our heuristics are based on local search: Begin with a convex decomposition and iteratively modify it locally to reduce the number of convex faces. The source code of our tools and heuristics is available at GitHub and can be used freely under the GPL(v3) license: `https://github.com/cgalab`.





2 Algorithmic methods

2.1 3-Approximation

Our tool 3Apx implements the 3-approximation algorithm by Knauer and Spillner [3]. Tests quickly showed that this approach generates decompositions that are clearly not optimal. Hence, we extended 3Apx by an approach based on onion layers [1]: We construct all onion layers and then find convex decompositions of the annuli between the layers. Contrary to [3], this approach does not modify the layer boundaries. See Figure 1 for sample decompositions obtained via 3-approximation and the onion layers. Experiments showed that computing a convex decomposition based on onion layers is superior to the 3-approximation algorithm even without merging convex faces across different onion layers, see the plot in Figure 7.

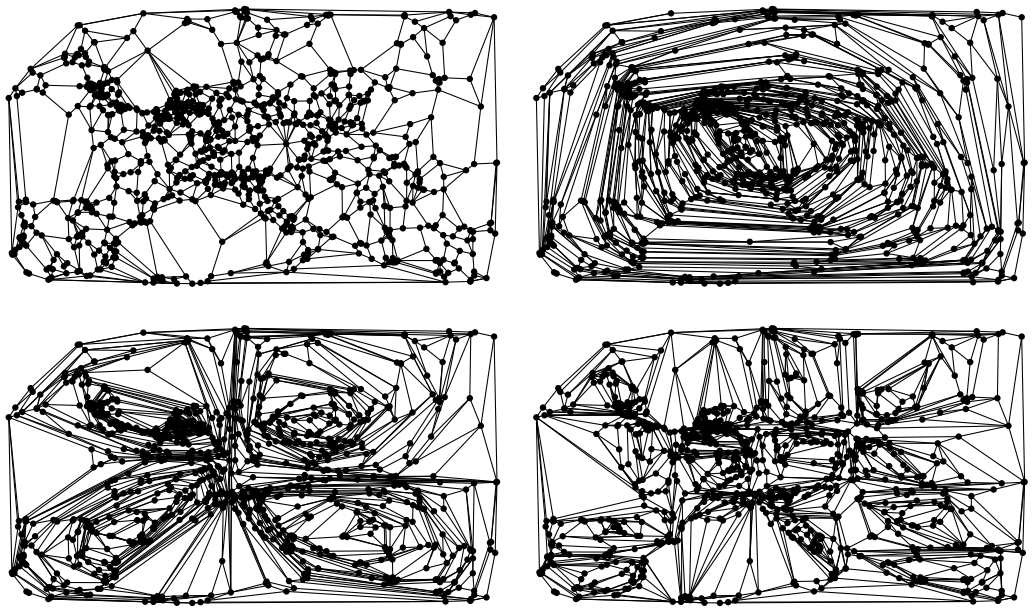


Figure 1 In reading order: When using the 3-approximation implemented in 3Apx for an instance with 1000 vertices we obtain a convex decomposition with 1350 faces; 1125 faces when using our approach based on onion layers without partitioning into cells; 1123 faces when partitioning into four cells and subsequent onion-layer based decomposition; and 1148 faces when using 16 cells.

A visual inspections of the results achieved by 3Apx quickly made it apparent that the decompositions computed contained lots of extremely long and thin triangles. Hence, we tried to partition a given input P into smaller "cells" and then run 3Apx on each cell individually. Then the individual decompositions are joined by triangulating the area between them and randomly dropping triangulation edges if this is possible without violating convexity. This produced visually nicer images such as the last two decompositions in Figure 1 but did not reduce the face counts substantially.

2.2 Merging faces

One of our earliest ideas was to do the obvious: Start with a triangulation of P and then merge adjacent faces by randomly dropping triangulation edges as long as faces remain convex. Tests with an initial straightforward implementation, MergeRefine, suggested that this is a promising approach, easily beating 3Apx (Figure 7). In order to be better prepared for refined heuristics we quickly re-implemented it in a new tool called Recursor. In particular, we resorted to a more advanced data structure for storing our decompositions.

RECURSOR keeps its state in a variant of a doubly-connected edge list (DCEL) or half-edge data structure. The base layer of our variant is a DCEL of a triangulation of P. Additionally, each half-edge pair is considered either *constrained* or not constrained, depending on whether the edge is part of our convex decomposition of P. As a layer on top of the base DCEL, each constrained half-edge, in addition to the pointers to the next triangulation edges encountered in clockwise (CW) or counter-clockwise (CCW) direction, also holds a reference to next CW and CCW constrained edges; cf. Figure 2. This enables constant-time testing whether an edge can be dropped, i.e., marked unconstrained, while keeping a fully fledged triangulation of P during the entire process. To obtain a decomposition, RECURSOR first uses Shewchuk's TRIANGLE [4] to construct a Delaunay triangulation of P, and then iterates over the edges in a random order, dropping every edge that can be dropped. This process continues until no further edge can be dropped, i.e., the decomposition is locally optimal.

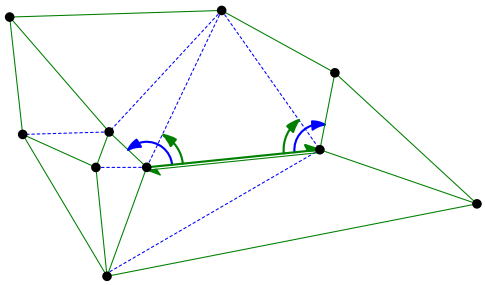


Figure 2 Our two-layer doubly-connected edge list stores two planar graphs simultaneously, with one planar graph being a subgraph of the other. A constrained half-edge h has references to its neighbors in the convex decomposition (green) and to the underlying triangulation (blue). The Challenge data set `euro-night-10` is shown.

Hole refinement. It is not surprising that a locally optimal decomposition may consist of many more faces than the true global optimum. Therefore, we worked on strategies that allow us to move away from local optima: RECURSOR picks a face f of the decomposition and a (random) number of its neighbors as a "hole" to work on. In general, it picks a face f that is incident to a high-degree vertex. We consider a vertex of the decomposition to be of high degree if its degree is larger than 3 or if its degree is equal to 3 and two incident edges span an angle of 180°. In other words, high-degree vertices are vertices whose degree could (locally) be reduced without violating convexity.

Once a hole has been selected, RECURSOR marks all its triangulation edges as constrained again. In the next step it tries to drop these edges in a (different) random order. If this results in a decomposition with no more faces than previously, we keep the new decomposition. Otherwise, we abandon the modifications and restore the old decomposition. See Figure 3 for a sample modification of a decomposition for the Challenge data set `euro-night-0000100`.

RECURSOR has several parameters to adjust, and we tried to fine-tune them "on the fly" as we applied it to the Challenge instances. Eventually we settled on hole sizes of $7 + P$ faces where P is a random number drawn from a geometric distribution with $p = 0.4$. In each hole, we try a number of decompositions that is equal to the number of triangulation edges in that hole.

Edge flips. Our initial decompositions were based on Delaunay triangulations of the input points. But there is no argument to justify why Delaunay edges were to be preferred over other triangulation edges. Hence, the next improvement of RECURSOR does a number of

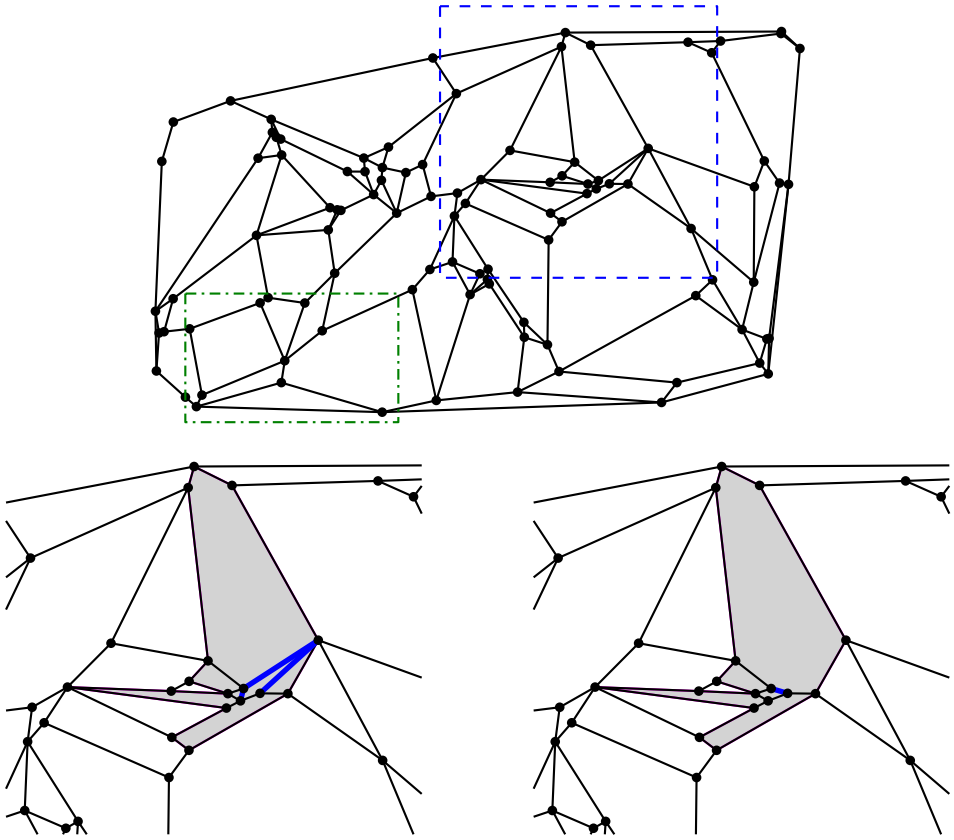


Figure 3 Top: An initial decomposition of `euro-night-0000100` by RECURSOR. Bottom: A detail of the initial decomposition (of the dashed blue frame in the full image), with those seven faces shaded in gray that were selected by the hole-refinement algorithm. The decomposition after one round of local optimization is shown on the right. The edges affected are shown in blue and bold. The improved variant has two faces less.

random edge flips on the triangulation of a hole before attempting to drop edges. The number of edge flips used by our code changed over time. After a series of quick experiments we ended up with using roughly $\sqrt[5]{t}$ edge flips, where t is the number of triangles in the hole.

Continuous refinement. So far, each run of RECURSOR always started from a triangulation of an input. We modified RECURSOR such that it could load a previous decomposition and work on it. This allowed us to run it on different instances whenever we had computational resources to spare, with no need to run it for long continuous stretches of time.

Parallel recursor. RECURSESPLIT is a wrapper around RECURSOR that partitions a given decomposition into a few dozen or a hundred non-overlapping sets of contiguous faces such that each set of faces forms a simply-connected region. Each such region is handed to a dedicated instance of RECURSOR which attempts to reduce the face count within that region. Note that RECURSOR does not require such a region to be convex. Since every individual run of RECURSOR operates strictly within its own region, the resulting decompositions can be merged trivially upon the completion of all runs of RECURSOR.

2.3 Flipper

Flipper was implemented relatively late during the time of the Challenge, not even a month prior to its end. It picks a point set and loads our currently best decomposition for that point set. Then it performs the following steps repeatedly: First, Flipper picks a high-degree vertex v and finds, if one exists, an incident edge (u, v) that can be rotated away from v without violating convexity at either u or v. That is, if u_0, u, u_1 is a CCW ordering of the vertices that share a decomposition edge with v then Flipper attempts to replace the edge (u, v) by either (u, u_0) or (u, u_1) if permissible. See the green edge in Figure 4, left. As shown in Figure 4, right, such a rotation may cause one of the edges incident at v or u to become unnecessary. In that case, we drop it. If, however, no edge can be removed, then the degree of v has decreased and the degree of either u_0 or u_1 has increased by one. Flipper then applies this process to u_0 or u_1.

Variations, added even later, try to pick a specific input point p at regular intervals. Then, with some probability, a rotation may only be carried out if the vertex whose degree is increased by one gets closer to p. The motivation for this decision was that finding edges that can be dropped gets easier if several vertices with higher degree are in close proximity.

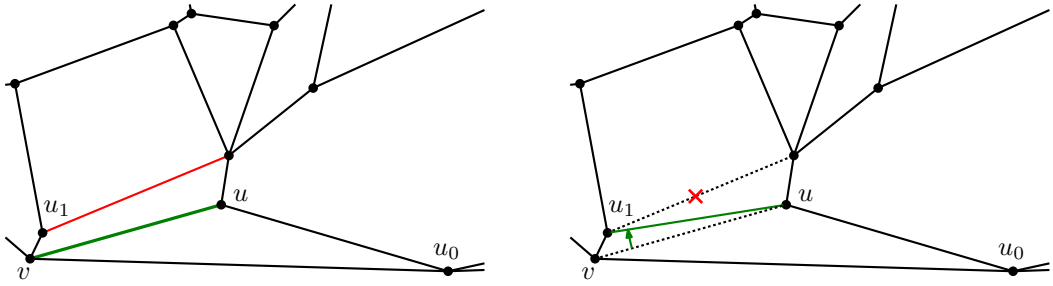


Figure 4 A detail of the initial decomposition (within the dash-dotted green frame of Figure 3): Rotating the green edge allows to drop the red edge while maintaining the convexity of all faces.

2.4 Orthogonal optimizer

Towards the end of the Challenge, a second batch of input instances was made available. While the organizers had warned a priori that the inputs may contain collinear points, the first batch of inputs contained relatively few subsets of collinear points per instance. In contrast, in the second batch of data, each input instance contained points sampled from a dense integer grid, resulting in every input instance containing many subsets of collinear points aligned along horizontal and vertical lines

A visual inspection quickly revealed that the approaches implemented so far did not generate decent decompositions for several inputs of the second batch. Therefore, we were forced to devise and implement a new heuristic. OrthoOpt generates initial convex decompositions geared towards this new type of input instances. It proceeds as follows: First, it sorts the input points of P lexicographically. Then it connects input points that share the same x-coordinate in order of increasing y-coordinates. Finally, it constructs a bottom bounding chain B and a top bounding chain T by linking the bottom-most (top-most, resp.) input points, and it triangulates all pockets between the convex hull of P and the current decomposition, as bounded by B and T. Of course, OrthoOpt can also proceed relative to y-coordinates rather than x-coordinates; see Figure 5. These initial decompositions were

passed to FLIPPER and RECURSOR for further optimization. In particular, these tools helped to get rid of unnecessary triangulation edges inside of the pockets formed by the convex hull of P and the two chains B and T.

3 Practical computation

3.1 Computational environment

Our tools were run on a diverse set of computers operated by our lab as well as by other groups at the University of Salzburg. We used a varying number of standard PCs plus some (rather small) compute servers, whenever a machine was available. (We did not have access to a genuine high-performance computer.) In particular, we used our own desktop machines whenever they were (partially) idle. One of them, an Intel Core i7-6700 CPU clocked at 3.40 GHz, was used to obtain the performance plot of Figure 6, which shows CPU-time consumptions of several of our tools for Challenge instances with different numbers of points.

Our low-profile way of accessing computers resulted in a highly non-uniform consumption of computational resources, which in turn had highly non-uniform performance levels, ranging from 15-year-old compute servers to machines acquired just a year ago. The availability of a particular machine or of some of its cores was discussed with the operator of that machine on a day-by-day or week-by-week basis. We set up a database and engineered some scripts that allowed all machines to fetch problem instances from and send results back to a home base.

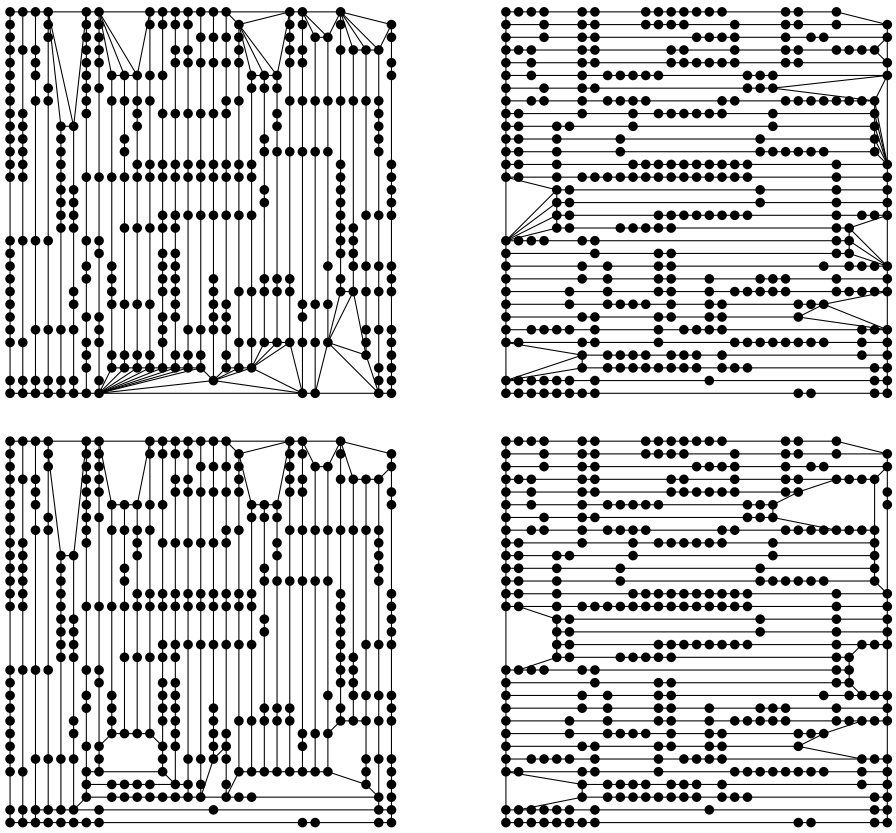


Figure 5 The two top figures show initial decompositions generated by ORTHOOPT for the 355-vertex instance `rop0000355`. The bottom left figure shows the best decomposition (with 44 faces) derived from an initial triangulation of `rop0000355`. The bottom right figure shows our overall best decomposition (with 36 faces) derived from an initial decomposition generated by ORTHOOPT.

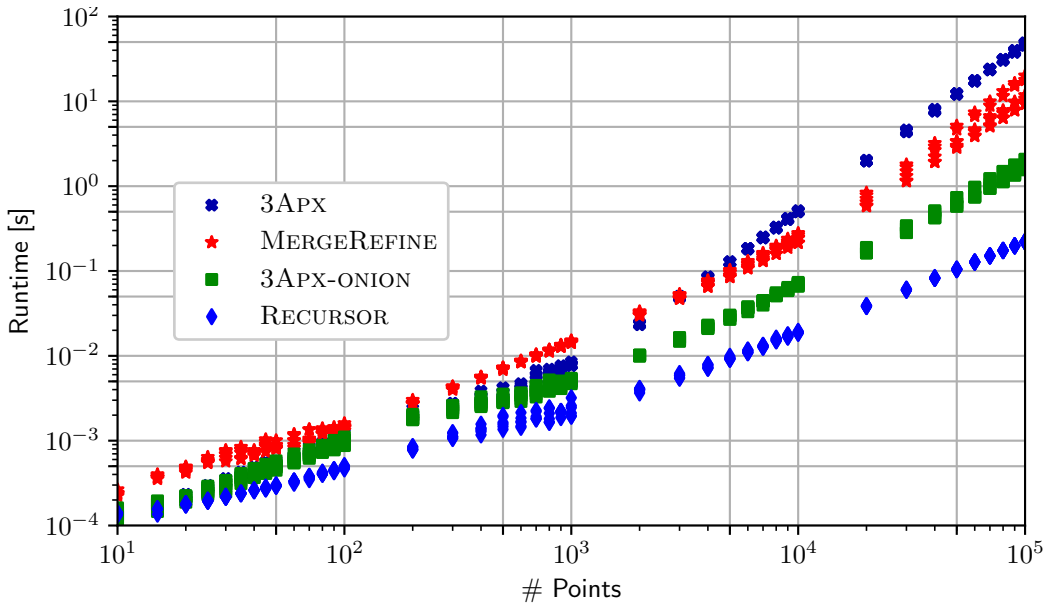


Figure 6 Time needed to obtain one initial decomposition for the competition inputs.

The heterogeneity of (our use of) the computational resources makes it very difficult to come up with a reliable ball-park figure of the total CPU time consumed. We estimate, though, that our tools would have kept a standard desktop machine busy for a few years.

3.2 Experimental results

The estimated quality of a specific convex decomposition is based upon its *score*, where

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

Figure 7 plots the score for the Challenge instance `euro-night-0100000` over time. It reflects the improvements achieved by refining our tools. While we did not generate such a plot for each and every instance, we did compare sample plots for a few instances: No significant differences were observed. That is, the plot shown in Figure 7 can be regarded as representative for the progress that we made on the Challenge instances of the first batch. The plot shows nicely how Recursor and Flipper interacted. Near the end of the competition, Recursor and Flipper by themselves rarely found better decompositions. However, even when a tool did not reduce the total number of faces, it still restructured the decomposition and uploaded it to our central server, which in turn may have enabled another tool to find some small improvement. The plot also indicates that each new tool yielded a substantial improvement at the beginning, with the gains tapering off as time progressed. So, likely, investing drastically more computational resources than what we had at our disposal would have hardly led to truly substantial improvements. In our case, the availability of human resources for devising and implementing new tools was the decisive limiting factor.

The second batch of Challenge instances made it apparent that our heuristics had been (implicitly) geared towards the inputs that they had to handle. The `rop*` input class proved to be particularly challenging for our initial strategy. Therefore, we introduced OrthoOpt

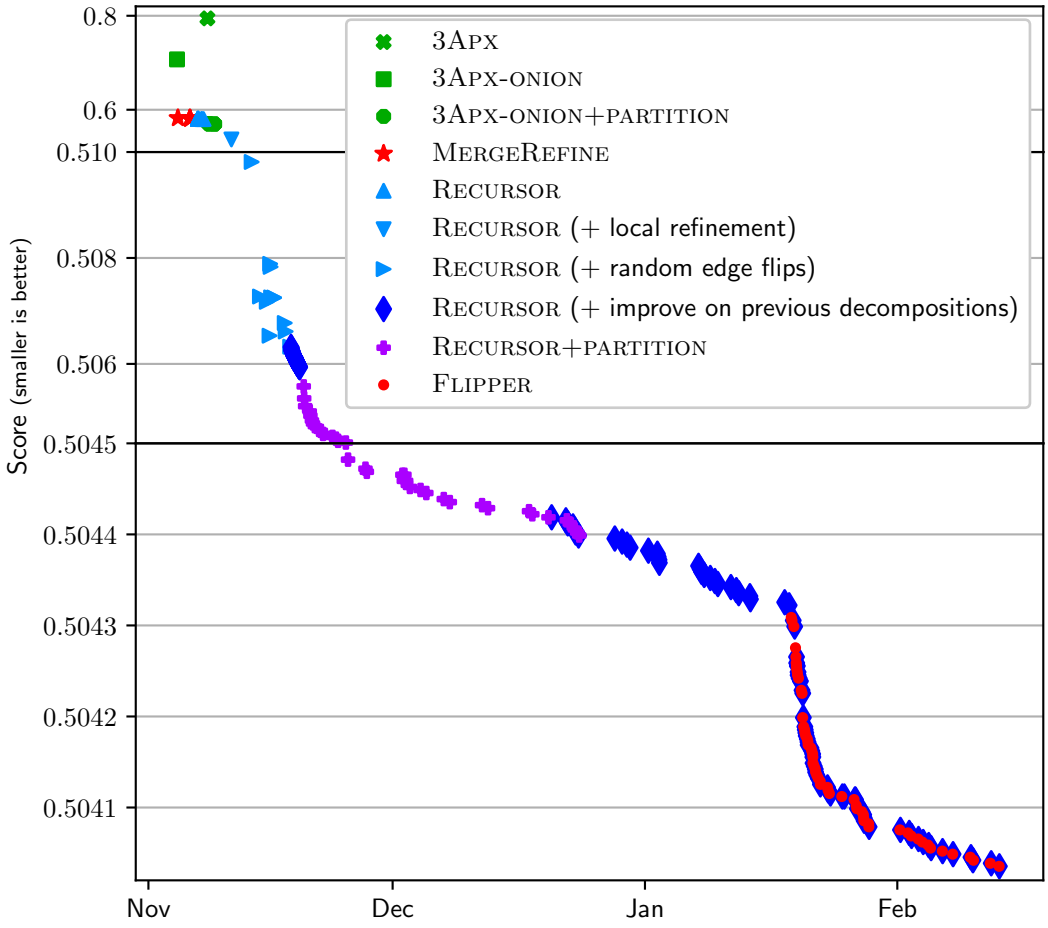


Figure 7 Score over time for `euro-night-0100000`. Note that the y-axis changes scale twice.

to generate initial decompositions that are tailored towards inputs with lots of dense, grid-aligned and, thus, collinear points. Figure 8 illustrates the score over time for `rop0064054` and `ortho_rect_union_47381`, which act as representatives for their corresponding input classes. Apparently, the introduction of OrthoOpt improved our solutions for the `rop*` instances, whereas it provided no improvement for the `ortho_rect_union*` input class.

In Figure 9, we show the scores of the overall best decompositions for various Challenge instances. Additionally, Figure 10 illustrates the development of the average score over time. Note that the significant improvement of the average score in mid January is due to the introduction of Flipper.

4 Conclusion

Our work makes it apparent that well-crafted heuristics run on moderate computing equipment are good enough to achieve decent minimum convex decompositions. But the second batch of Challenge instances made it also apparent that heuristics need not be universally applicable. Rather, they may require an adaption relative to the characteristics of the input data.

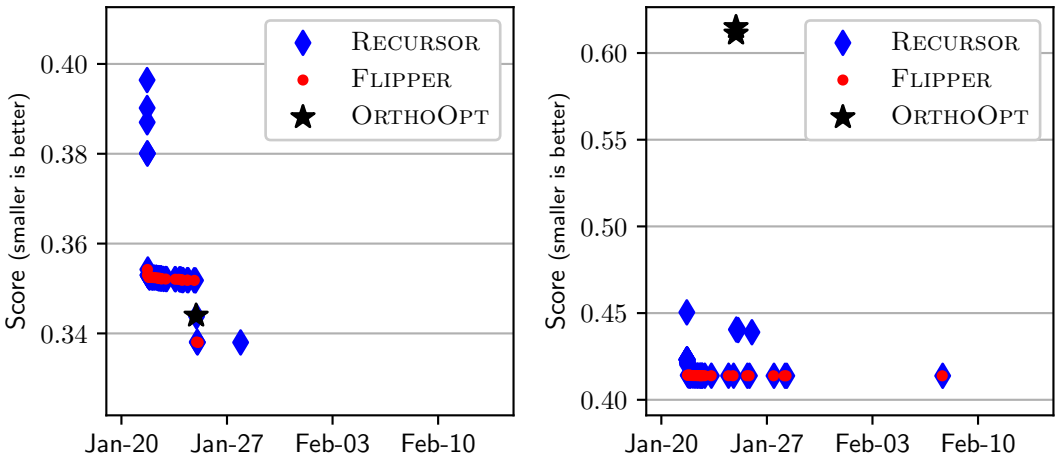


Figure 8 Score over time for `rop0064054` and `ortho_rect_union_47381`.

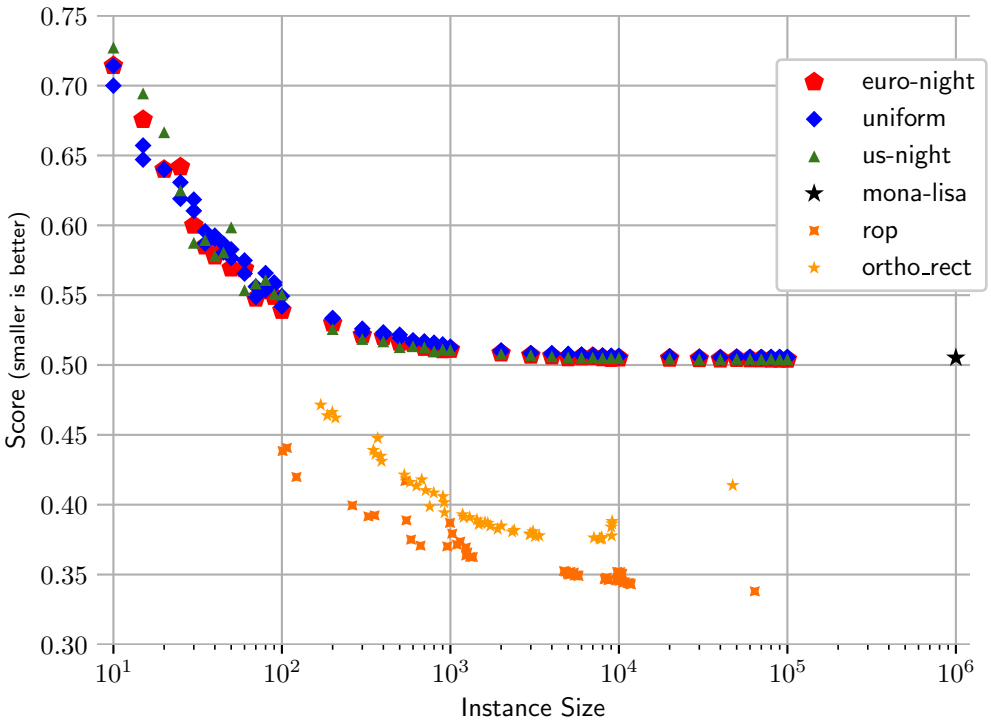


Figure 9 Score per instance.

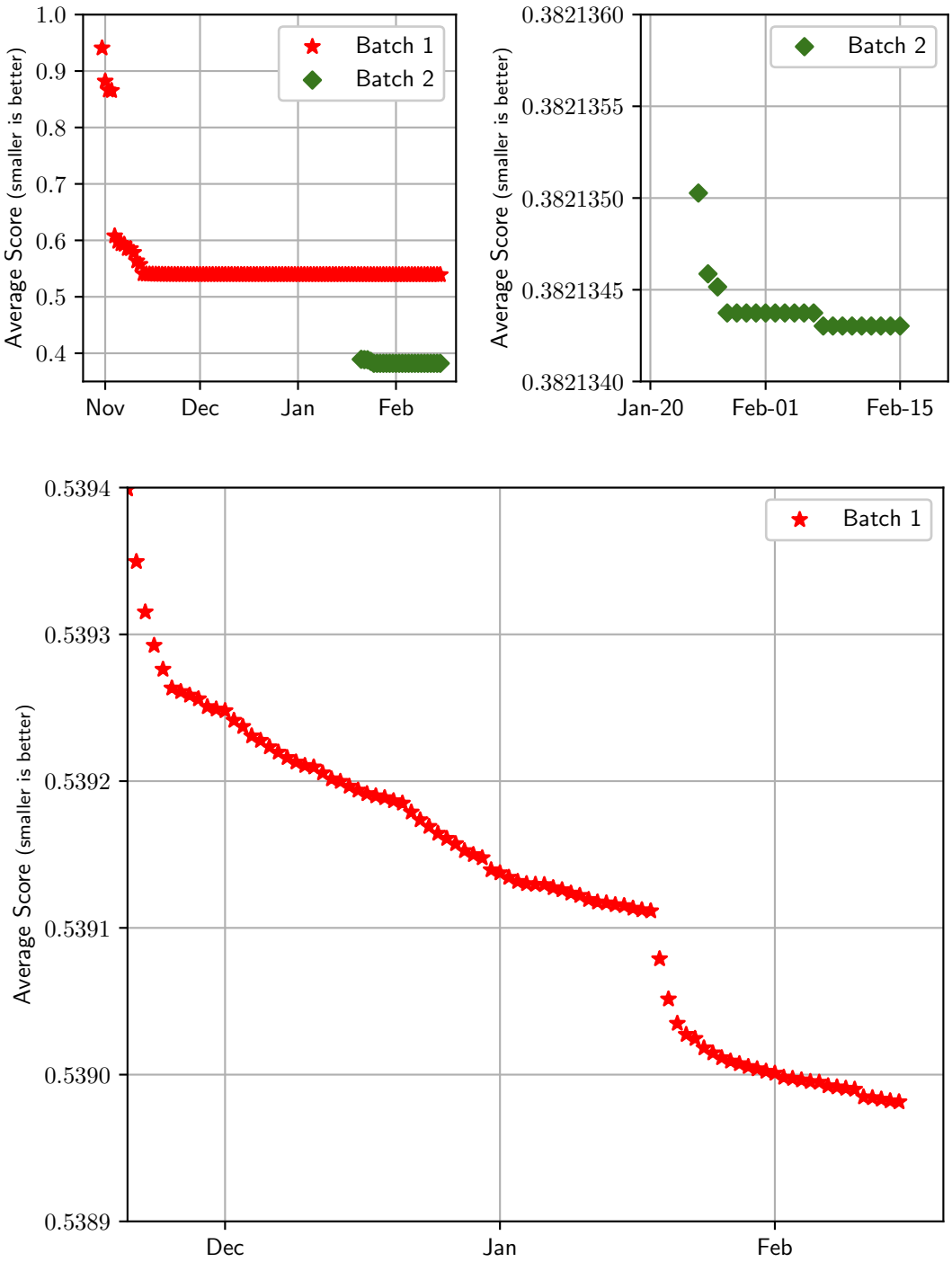


Figure 10 Average score over time.

References

1 Bernard Chazelle. On the Convex Layers of a Planar Set. *IEEE Transactions on Information Theory*, 31(4):509–517, July 1985. doi:10.1109/TIT.1985.1057060.

2 Erik D. Demaine, Sándor P. Fekete, Phillip Keldenich, Dominik Krupke, and Joseph S. B. Mitchell. Computing Convex Partitions for Point Sets in the Plane: The CG:SHOP Challenge 2020, 2020. arXiv:2004.04207.

3 Christian Knauer and Andreas Spillner. Approximation Algorithms for the Minimum Convex Partition Problem. In *Algorithm Theory – SWAT 2006*, pages 232–241, 2006.

4 Jonathan Richard Shewchuk. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In *Applied Computational Geometry: Towards Geometric Engineering*, volume 1148 of *Lecture Notes in Computer Science*, pages 203–222. Springer-Verlag, May 1996. ISBN 3-540-61785-X.