2-Opt Moves and Flips for Area-optimal Polygonizations

GÜNTHER EDER, MARTIN HELD, STEINÞÓR JASONARSON, PHILIPP MAYER, and PETER PALFRADER, FB Computerwissenschaften, Universität Salzburg, Austria

Our work on the Computational Geometry Challenge 2019 on area-optimal polygonizations is based on two key components: (1) sampling the search space to obtain initial polygonizations and (2) optimizing such a polygonizations. Among other heuristics for obtaining polygonizations for a given set P of input points, we discuss how to combine 2-opt moves with a line sweep to convert an initial random (non-simple) polygon whose vertices are given by P into a polygonization $\mathcal{P}$. The actual optimization relies on a constrained triangulation of the interior and exterior of a polygonization to speed-up local modifications of the polygonization to increase or decrease its area.

CCS Concepts: • **Theory of computation** → **Design and analysis of algorithms**; *Computational Geometry;*

Additional Key Words and Phrases: Computational geometry, geometric optimization, algorithm engineering, exact algorithms, polygonization, area optimization

ACM Reference format:
Günther Eder, Martin Held, Steinþór Jasonarson, Philipp Mayer, and Peter Palfrader. 2022. 2-Opt Moves and Flips for Area-optimal Polygonizations. *J. Exp. Algorithmics* 27, 2, Article 2.7 (March 2022), 12 pages.
https://doi.org/10.1145/3500913

1 INTRODUCTION

Consider a given set P of n distinct points in the plane. A (simple) *polygonization* $\mathcal{P}$ of P is a closed tour on P that forms a simple polygon. The task of the Computational Geometry Challenge 2019 was to find polygonizations of P with minimum ("Min-Area") and maximum ("Max-Area") enclosed area for various input sets P and different sizes n. We refer to Reference [5] for a survey with background information on this problem and to References [4, 8, 11, 13] for descriptions of the approaches taken by the other four top teams.

In general, the number of polygonizations of P is exponential in n: García et al. [7] prove a lower bound of $\Omega(4.642^n)$ for the number of polygonizations that a set of n points can admit. Hence, a brute-force search for Min-Area/Max-Area polygonizations of P is no option. This fact motivated us to base our approach on two key components:

Sampler: The sampler computes an initial polygonization of P.

Work supported by Austrian Science Fund (FWF): Grants ORD 53-VO and P31013-N31.
Authors' address: G. Eder, M. Held, S. Jasonarson, P. Mayer, and P. Palfrader, FB Computerwissenschaften, Universität Salzburg, Salzburg, Austria; emails: {geder, held, sjas, pmayer, palfrader}@cs.sbg.ac.at.

1084-6654/2022/03-ART2.7 $15.00
https://doi.org/10.1145/3500913

Optimizer: The optimizer takes a polygonization and, while maintaining its simplicity, modifies the polygonization to optimize its area.

It is obvious that the mere number of polygonizations makes it highly unlikely that a sampler will return a polygonization that already has a near-optimal area. It is less clear, though, whether different classes of initial polygonizations might allow the optimizer to yield better results. And even if a "good" class of initial polygonizations existed, it is not obvious how to compute them efficiently.

We opted for computing a large sample set of polygonizations with different characteristics for every given set of points. This initial generation of polygonizations on P is accomplished by our two random polygon generators **RandomPolygonGenerator (Rpg)** and **SweepPolygonGenerator (Spg)**, by our simple generator for monotone polygons, PAO-mono, and by Helsgaun's TSP code LKH [10]. Then, for every polygonization of our sample set, we apply local optimizations to optimize the area. This task is carried out by Pao-Flip, our polygon area optimizer. Source code of our implementations is available on `GitHub` and can be used freely under the GPL(v3) license:

Spg	`https://github.com/cgalab/genpoly-spg`
Rpg	`https://github.com/cgalab/genpoly-rpg`
Pao-Flip	`https://github.com/cgalab/pao-flip`

In the sequel, we describe the algorithms of Rpg and Spg and analyze their properties. Then we present Pao-Flip's optimization heuristics applied to convert our initial polygonizations into area-optimal polygonizations.

2 METHODS

2.1 Polygonization

Rpg was designed and implemented by Auer and Held [1] in the late 1990s. It offers various heuristics to generate pseudo-random polygonizations for a given set of vertices. We resurrected this code, updated it to make it compile on modern platforms, and extended and improved its algorithms. For instance, Rpg now supports the generation of polygons with holes [6].

Let P denote a set of n input points in the plane that are indexed from 0 to $n-1$. (In the sequel, all indices will be taken modulo n.) We used three heuristics of Rpg: RPG-star to generate pseudo-random star-shaped polygonizations, and RPG-2opt and RPG-space to generate two families of arbitrary polygonizations.

RPG-star: It computes the convex hull $CH(P)$ of P and then picks a random locus p within $CH(P)$. Then a star-shaped polygonization $\mathcal{P}$ of P is obtained in $O(n \log n)$ time by sorting the points of P radially around p.

RPG-space: It is a divide&conquer approach that partitions P recursively into two subsets whose convex hulls are disjoint. The recursion ends when a subset has three or fewer vertices and, therefore, a polygonal chain on these vertices can be obtained easily. The final polygonization is obtained by splicing the individual chains together. The worst-case complexity of RPG-space is $O(n^2)$, but its expected complexity is $O(n \log n)$.

RPG-2opt: Initially, a polygon is created by choosing a random permutation of the input points. This initial polygon is not simple with high probability. Self-intersections are removed by repeatedly applying so-called 2-opt moves [3, 16]: Two polygon edges that intersect are regarded as the diagonals of a quadrilateral and replaced by a suitable pair of edges of that quadrilateral. If the polygon contains the vertex sequences $(\dots, v_{i-1}, v_i, v_{i+1}, v_{i+2}, \dots)$ and $(\dots, v_{j-1}, v_j, v_{j+1}, v_{j+2}, \dots)$ such that the edges $\overline{v_i, v_{i+1}}$ and $\overline{v_j, v_{j+1}}$ intersect, then we

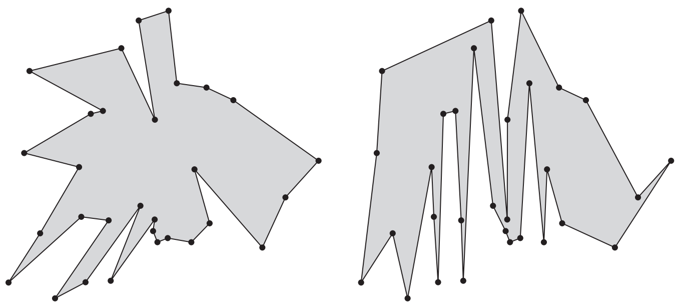


Fig. 1. Left-to-right: A sample RPG-star polygonization, an RPG-space polygonization and an RPG-2opt polygonization is shown for the same set of 30 input points.

replace these sequences by $(\ldots, v_{i-1}, v_i, v_j, v_{j-1}, \ldots)$ and $(\ldots, v_{i+2}, v_{i+1}, v_{j+1}, v_{j+2}, \ldots)$; see Figure 3(a). (Note that a 2-opt move results in a re-orientation of a chain of the polygon). A key property of a 2-opt move is that it decreases the perimeter of the polygon (if the two edges are not collinear). This guarantees that we will eventually arrive at a polygonization if we apply 2-opt moves repeatedly to resolve intersections. Very similar to Held's triangulation code Fist [9], a uniform grid is applied as a subdivision of the plane in order to speed up the detection of pairs of polygon edges that intersect. Every grid cell that is intersected by two (or more) edges of the polygon is a candidate cell that might witness (at least) one edge intersection. These candidate cells are checked in random order, and a 2-opt move is applied to resolve each edge intersection found. (Of course, the set of candidate cells will change dynamically as 2-opt moves are applied.) A result by van Leeuwen and Schoone [15] tells us that we may need $O(n^3)$ 2-opt moves to convert an initial polygon into a simple polygonization.

We refer to Auer and Held [1] for a more detailed description of these three heuristics of Rpg. Sample polygonizations computed by these heuristics are shown in Figure 1. While RPG-2opt generates every possible polygonization on P with positive probability, RPG-space cannot generate all polygonizations and tends to generate outlines that contain lots of zigzags.

At the start of this work we attempted to formulate the area optimization problem as a TSP, with "distances" chosen to reflect Gauss's area formula. While we did not succeed with this approach, we did end up using an approximate TSP tour (with standard Euclidean distance) as one additional initial polygonization per input. These TSP tours were generated by the Lin-Kernighan-Helsgaun TSP solver **LKH**[1] [10]. Helsgaun's LKH uses a local search heuristic based on the variable depth local search of Lin and Kernighan [12]. The Lin-Kernighan local search can be seen as a generalization of 2-opt and 3-opt moves. Since an approximate TSP tour need not be a polygonization, we checked the simplicity of all **LKH** outputs by means of one of our 2-opt codes. (As a matter of fact, all **LKH** outputs turned out to be polygonizations.)

Finally, while we were working on the other codes for sampling the polygonizations, we implemented a simple algorithm to generate x-monotone polygonizations (**PAO-mono**): A polygon's lower chain is given by the lower chain of the convex hull of the input points, and its upper chain is given by the left-to-right sequence of all remaining points. To get more than one sample polygonization, we applied a (random) rotation to every point set prior to applying this simple algorithm, thus obtaining a total of 64 monotone polygonizations per input set.

[1]We obtained version 2.0.9 from http://akira.ruc.dk/~keld/research/LKH/.

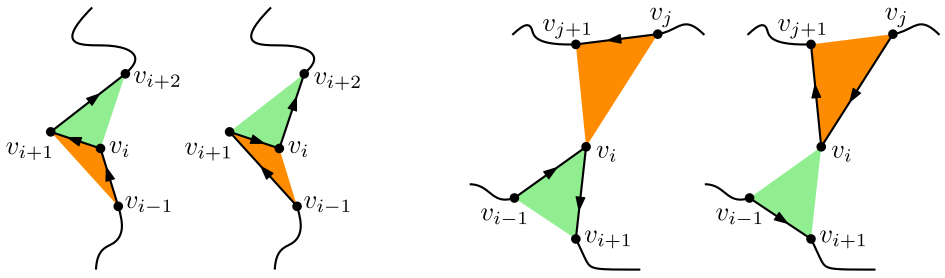


Fig. 2. Local optimizations by "inverting" (left pair of figures) and "flipping" (right).

2.2 Area Optimization

The first method that we used is *inverting*: Consider four consecutive vertices $v_{i-1}, v_i, v_{i+1}, v_{i+2}$ of a polygonization $\mathcal{P}$ such that v_i is convex[2] and v_{i+1} is reflex or vice versa; see Figure 2. If the two triangles $\Delta(v_{i-1}, v_i, v_{i+1})$ and $\Delta(v_i, v_{i+1}, v_{i+2})$ are empty,[3] then we can locally modify $\mathcal{P}$ by replacing the polygonal chain $(v_{i-1}, v_i, v_{i+1}, v_{i+2})$ by the chain $(v_{i-1}, v_{i+1}, v_i, v_{i+2})$, i.e., by inverting the order of v_i and v_{i+1} within $\mathcal{P}$. Of course, this will preserve the simplicity of $\mathcal{P}$.

The second method is *flipping*: Consider a vertex v_i such that the triangle $\Delta(v_{i-1}, v_i, v_{i+1})$ is empty. If there exist consecutive vertices v_j, v_{j+1} such that the triangle $\Delta(v_j, v_{j+1}, v_i)$ is also empty and if one of the triangles belongs to the interior of $\mathcal{P}$ while the other triangle belongs to the exterior of $\mathcal{P}$, then we can modify $\mathcal{P}$ by connecting v_{i-1} and v_{i+1} directly and inserting v_i between v_j and v_{j+1}. Again the simplicity of $\mathcal{P}$ is preserved; see Figure 2. We note that inverting can be seen as a special case of flipping, with $j = i + 1$.

Both inverting and flipping replace one triangle by another triangle. In general, the areas of these two triangles will not be identical. Hence, repeated appropriate applications of inverting or flipping increase or decrease the area of a polygonization. Of course, we may run out of applicable inverting/flipping operations and this process may get stuck without achieving a polygonization with optimum area.

3 ALGORITHM ENGINEERING

3.1 Efficient 2-opt Moves

The main building block for carrying out 2-opt moves among the edges of a polygon is given by the detection of pairs of edges that intersect. Nevertheless, we need to avoid an all-pairs check. For a static set of straight-line segments, the Bentley-Ottmann line sweep [2] is a classical (and relatively easy to implement) tool for detecting all pairs of edges that intersect. In our application the situation is complicated by the fact that we need to do more than intersection detection on a static set of line segments: As we apply 2-opt moves to resolve intersections among edges, new intersections can be introduced, so our algorithm must be able to perform a limited version of dynamic segment intersection detection.

As it is not immediately obvious how to combine a line sweep with 2-opt moves in an efficient way, we were surprised that we could not find any experimental analysis of such a line sweep. Hence, we implemented Spg and tested three variants of the sweep. The main sweep direction of all our sweeps is from left to right. Our sweeps differ mainly in how they detect and resolve an intersection.

[2]As usual, we call a vertex v of $\mathcal{P}$ convex (or reflex) if the interior angle at v is less than 180° (or greater than 180°, respectively).

[3]We call a triangle Δ empty if the intersection of Δ with (the boundary of) $\mathcal{P}$ is formed only by vertices and edges of Δ.

Assume that the sweep line is at x coordinate x_s and has encountered a pair of edges that intersect to the right of x_s. This intersection is resolved by applying a 2-opt move.

SPG-loop: The sweep continues rightwards and scans edges to the right of x_s. When the sweep reaches the right-most input point, we restart the sweep from scratch at the left-most input point. We loop through this left-right sweeping until no further intersection is found.

SPG-resolve: We check for other intersections at the current sweep-line position x_s and resolve all those intersections. Then the sweep continues rightwards and scans edges to the right of x_s. Again, after reaching the right-most input point we restart the sweep at the left-most input point.

SPG-reverse: We reverse the sweep direction and scan edges to the left of x_s. This leftwards sweep allows us to deal with possibly new edge intersections introduced by the 2-opt move to the left of x_s, i.e., to detect and resolve such intersections by additional 2-opt moves. We resume our rightwards sweep once the leftwards sweep has reached the left-most end-point of an edge involved in one of those 2-opt moves. Hence, we get the invariant that no intersection exists to the left of the current sweep position whenever we sweep rightwards. Thus, the sweep is finished once the right-most input point has been reached.

All three variants have in common that they keep sweeping leftwards or leftwards/rightwards until we get one full pass over all edges that does not reveal an intersection. This way it is guaranteed that the final polygon is simple.

In addition to these three variants of a Bentley-Ottmann line sweep combined with 2-opt moves, we implemented a brute-force algorithm (**BF-2opt**) for comparison purposes: Initially, we sort all input points lexicographically[4] with respect to their x and y coordinates. Furthermore, we associate with each point p the two edges of the polygon that are currently incident at p. Then intersections are found and resolved by two nested loops, where the outer loop runs over all edges e of the polygon and the inner loop runs over all edges e' whose projection onto the x-axis overlaps with e.

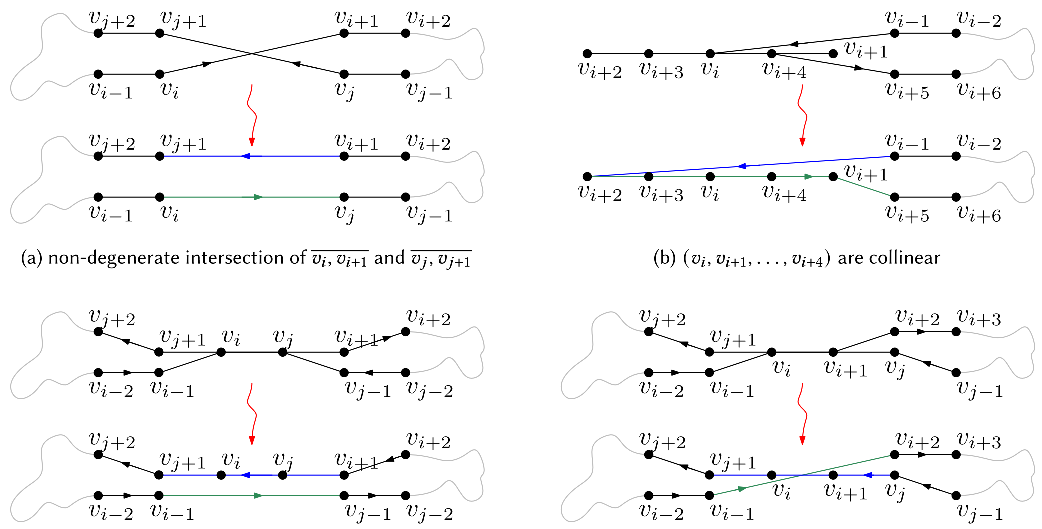
Handling Collinearities. We note that collinear edges need special care, because a 2-opt move applied to collinear edges need not always result in a shortening of the perimeter of the polygon. Worse, we might get into a two-opt cycle that keeps swapping collinear edges forth and back. To guarantee that the perimeter of the polygon decreases strictly monotonically, we

(1) ensure that consecutive edges of the polygon that are collinear do not overlap each other, and
(2) refine the 2-opt move for non-consecutive edges that overlap.

If the polygon under consideration contains a vertex sequence $(\ldots, v_i, v_{i+1}, \ldots, v_{j-1}, v_j, \ldots)$ of three or more consecutive vertices that are collinear, then we arrange these vertices in lexicographical order. In the example depicted in Figure 3(b), where $j = i + 4$, re-arranging these vertices either (as depicted) in descending lexicographical order $(v_{i+1}, v_{i+4}, v_i, v_{i+3}, v_{i+2})$ or in ascending lexicographical order $(v_{i+2}, v_{i+3}, v_i, v_{i+4}, v_{i+1})$ ensures we reduce the perimeter of the polygon. Thus, in this example, we could choose the order randomly. If only one order reduces the perimeter of the polygon, then this order is chosen. (A simple case analysis shows that at least one of the two orders always reduces the perimeter.)

Now suppose that the polygon under consideration contains vertex sequences $(\ldots, v_{i-2}, v_{i-1}, v_i, v_{i+1}, v_{i+2}, \ldots)$ and $(\ldots, v_{j-2}, v_{j-1}, v_j, v_{j+1}, v_{j+2}, \ldots)$ such that the edges $\overline{v_i, v_{i+1}}$ and $\overline{v_j, v_{j+1}}$ are collinear and overlap. We get two basic geometric scenarios of how these edges can

[4]If $p.x$ denotes the x coordinate and $p.y$ denotes the y coordinate of the point p, then p is lexicographically smaller than the point q precisely if either $p.x < q.x$ or $p.y < q.y$ in case of $p.x = q.x$.


v_{j+2} v_{j+1} v_{i+1} v_{i+2}
v_{i-1} v_i v_j v_{j-1}
v_{i-1} v_{i-2} v_{i+1}
v_{i+2} v_{i+3} v_i v_{i+4} v_{i+5} v_{i+6}
v_{i-2} v_{i-1} v_{j-1} v_{j-2}
v_{i+3} v_{i+1}

(a) non-degenerate intersection of $\overline{v_i, v_{i+1}}$ and $\overline{v_j, v_{j+1}}$

(b) $(v_i, v_{i+1}, \ldots, v_{i+4})$ are collinear

(c) v_j lies on $\overline{v_i, v_{i+1}}$ and v_i lies on $\overline{v_j, v_{j+1}}$

(d) v_i and v_{i+1} lie on $\overline{v_j, v_{j+1}}$

Fig. 3. Details of the edge replacements if edges intersect or overlap.

interact; see Figure 3(c) and (d). If v_j lies on $\overline{v_i, v_{i+1}}$ and v_i lies on $\overline{v_j, v_{j+1}}$, as shown in Figure 3(c), then we resolve the overlap by considering the new vertex sequences $(\ldots, v_{i-2}, v_{i-1}, v_{j-1}, v_{j-2}, \ldots)$ and $(\ldots, v_{i+2}, v_{i+1}, v_j, v_i, v_{j+1}, v_{j+2}, \ldots)$. If both v_i and v_{i+1} lie on $\overline{v_j, v_{j+1}}$, as shown in Figure 3(d), then we resolve the overlap by considering the new vertex sequences $(\ldots, v_{i-2}, v_{i-1}, v_{i+2}, v_{i+3}, \ldots)$ and $(\ldots, v_{j-1}, v_j, v_{i+1}, v_i, v_{j+1}, v_{j+2}, \ldots)$. Note that the new vertex sequence, once again, may contain new intersections. However, it is easy to see that such a (generalized) 2-opt move reduces the perimeter of the polygon in both scenarios. Hence, this modification of the standard 2-opt moves guarantees that we will end up with a simple polygon even in the presence of collinear input points.

3.2 PAO-Flip

The flipping and inverting methods were implemented in C++ in our optimizer Pao-Flip. Its input is a polygonization $\mathcal{P}$. Without loss of generality, assume that we seek a Min-Area polygonization. Since inverting can be seen as a special case of flipping, we focus on describing our approach to flipping.

We use Shewchuk's Triangle [14] to construct a constrained Delaunay triangulation of $CH(P)$, with the edges of $\mathcal{P}$ forming the constraints. By taking advantage of the triangulation, we can easily identify candidates for a flip. A candidate pair consists of two triangles such that we can flip the adjacent boundary, as described in Section 2.2 and shown in Figure 2.

Pao-Flip scans $\mathcal{P}$ until it arrives at a reflex vertex v_i where the exterior triangle $\Delta(v_{i-1}, v_i, v_{i+1})$ is empty. Due to an oversight in the implementation of the algorithm that was discovered only recently, well after the end of our work, the symmetric case of *convex* vertices v_i, where the triangle $\Delta(v_{i-1}, v_i, v_{i+1})$ is an interior triangle, was not considered in our implementation. Hence, there may have been invert/flip operations that were not considered and that might have further improved our results.

Emptiness of such a triangle can easily be checked by considering the end-points of all triangulation edges incident to v_i that are outside of $\mathcal{P}$. Then Pao-Flip iterates through the interior triangles incident at v_i and identifies the triangle $\Delta(v_i, v_j, v_{j+1})$ with maximal area and that is incident to a boundary edge. Let A_e denote the area of that exterior triangle, and let A_i denote the area of the interior triangle. If $A_i > A_e$, then we apply flipping and decrease the area bounded by $\mathcal{P}$. We continue the scan along $\mathcal{P}$ and repeat this process until all reflex vertices have been visited.

This basic approach runs in $O(n)$ time, because we look at every reflex vertex at most once and we have a linear number of triangulation edges that are examined at most twice. Our tests quickly demonstrated that the polygonizations obtained were not necessarily area-optimal or even very close to area-optimal.

Hence, we introduce randomness by weakening the greediness of our approach: We allow flipping whenever v_i has at least one incident interior triangle with area A_i such that $A_i > A_e$. The actual triangle to be flipped is chosen uniformly at random from all interior triangles whose area is greater than A_e. Furthermore, we visit the vertices in random order. This modification does not change the overall runtime, since we iterate over all triangles, store the relevant triangle indices, and then choose an index at random.

Furthermore, we increase the number of candidates available for flipping at a reflex vertex v_i by first modifying the triangulation. Rather than sticking to the original constrained Delaunay triangulation during the entire optimization, we apply standard edge flipping: Whenever two adjacent interior triangles form a convex quadrilateral, we could flip their common diagonal. A linear number of random edge flips is applied between rounds of triangle flipping.

4 RESULTS

In addition to attempting to compute Min-Area/Max-Area-polygonizations, we subjected our codes to various tests. All runtime experiments were run on an Intel Core i7-6700 CPU clocked at 3.40 GHz, with 256 KiB L1 cache, 1 MiB L2 cache and 8 MiB L3 cache. Figure 4 shows a plot of the runtimes for the input sets of the Challenge, except for one input with one million points. As predicted by theory, both RPG-space and RPG-star show a slightly super-linear complexity and seem to run in roughly $10^{-7}\, n \ln n$ seconds. Among the different variants of the 2-opt algorithm, the brute-force intersection checking of BF-2opt is slowest and the rightwards/leftwards sweep of SPG-reverse is fastest for polygonizations of at least 10^3 input points. For such inputs, SPG-reverse is roughly 25 times faster than BF-2opt, and still about 6 to 8 times faster than the other variants based on line sweeping. It is not surprising that BF-2opt works better the smaller the input is.

To get a better understanding of our Spg variants, we investigated their main characteristics by running each of them 100 times per instance. The left plot of Figure 5 shows how often SPG-resolve and SPG-loop restart from scratch to pass over all edges and how often SPG-reverse reverses the sweep direction from rightwards to leftwards. The plot indicates a barely super-linear growth rate of the number of reversals and a clearly sub-linear growth rate of the number of passes. However, keep in mind that each new pass of SPG-resolve and SPG-loop involves scanning all edges once again, while SPG-reverse avoids the scanning of edges that are known to contain no intersections.

The right plot of Figure 5 tells us that SPG-reverse performs fewer 2-opt moves to obtain a polygonization than SPG-resolve and SPG-loop. Furthermore, SPG-loop again fares better than SPG-resolve. The numbers of 2-opt moves of all three variants seem to have a slightly super-linear growth rate. That is, the number of 2-opt moves observed in our experiments always stayed far below the cubic worst-case bound established by van Leeuwen and Schoone [15]. (But it is not particularly surprising that our tests failed to make this bound evident, since we tested far too few samples relative to the huge sample spaces of all polygons defined by n vertices.)

These plots support our understanding that SPG-reverse requires fewer computational steps than SPG-loop, which in turn is better than SPG-resolve. This conclusion is reflected by the runtimes shown in Figure 4.

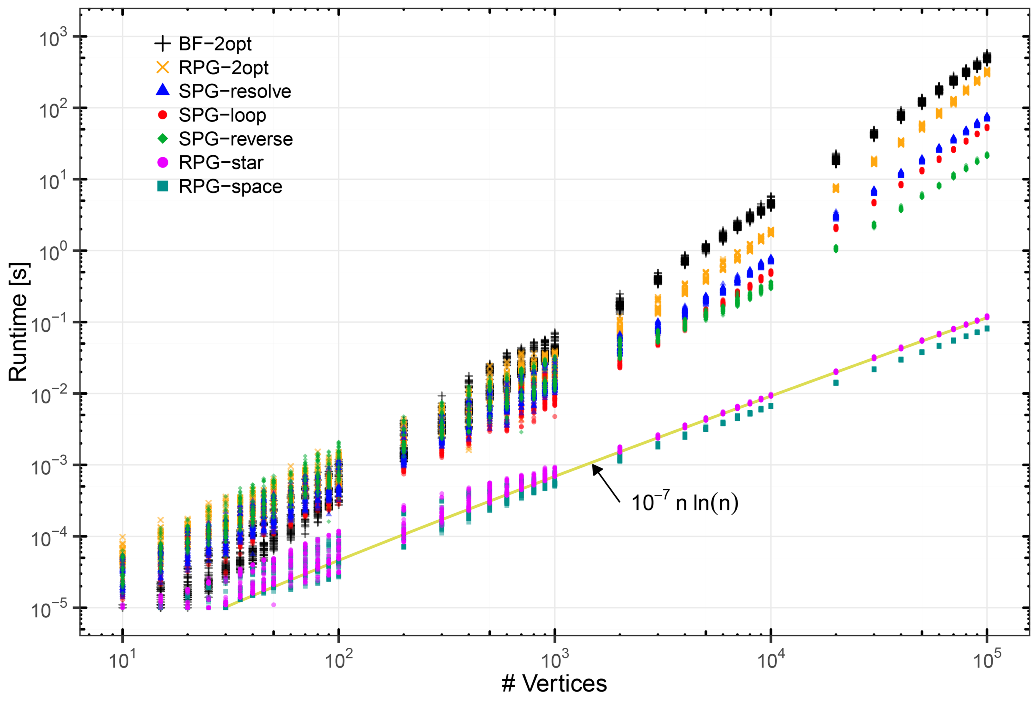
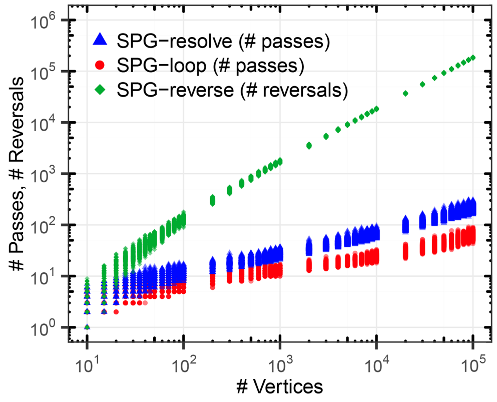


Fig. 4. Runtimes of different polygon samplers.



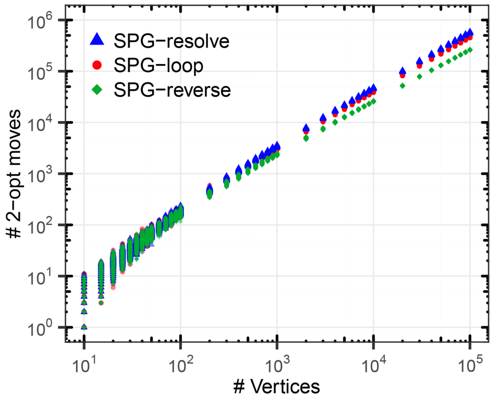


Fig. 5. Number of passes/reversals of our Spg variants, and number of 2-opt moves carried out.

Our basic assumption during this work was that a polygonization with small (respectively, large) area is more likely to be obtained by our optimizer Pao-Flip if the initial polygonization has already a comparatively small (respectively, large) area. A second initial assumption was that it would not matter which particular heuristic generated the initial polygon. To probe this claim we analyzed the percentage of the convex hull of a point set that was covered by the interior of an actual polygonization. Figure 6 shows that RPG-space tends to generate initial polygonizations with larger area than RPG-star. We use the same scoring scheme as in the Challenge: The score for an instance is the ratio given by the area achieved divided by the area of the convex hull. Thus, a lower value is better for Min-Area, whereas a larger value is better for Max-Area.

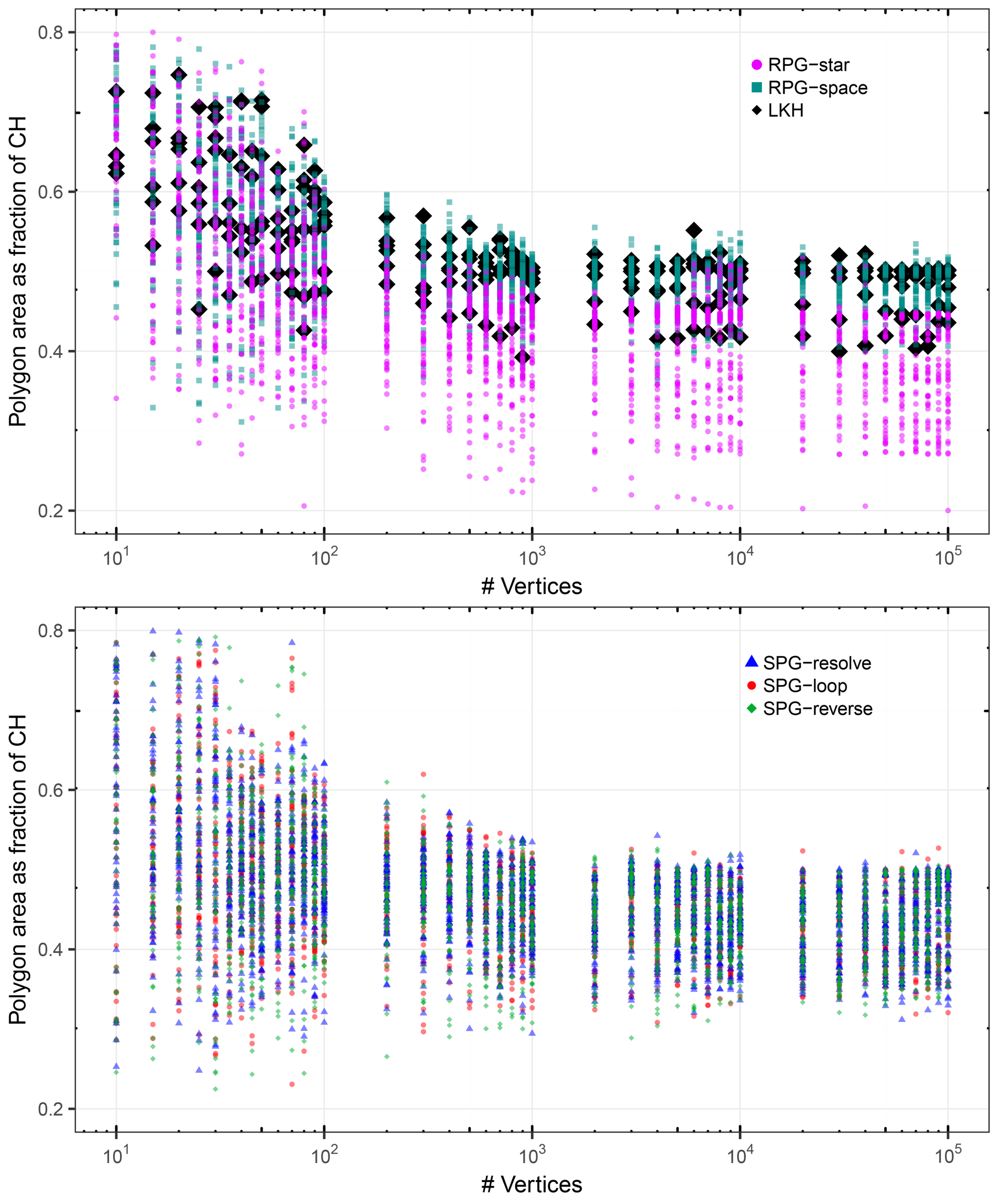
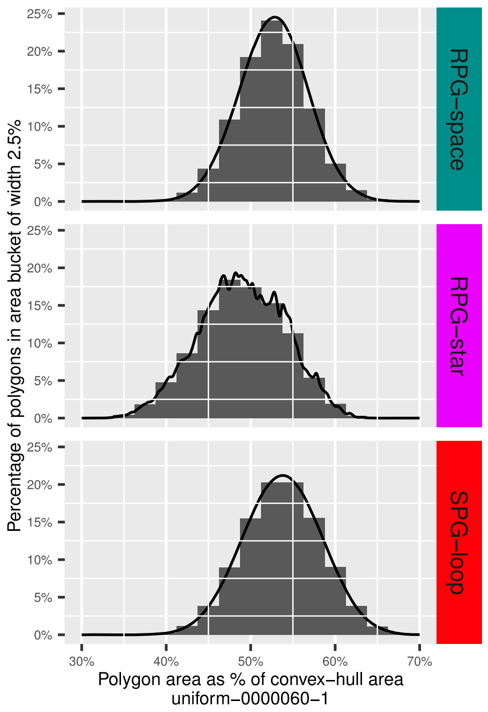


Fig. 6. Area distribution of randomly sampled polygonizations with Rpg and Spg and the area of approximate TSP tours by LKH.

No obvious winner is discernible among the 2-opt variants: For each input size of the Challenge, the distribution of the area of a polygonization seems to be roughly uniform within about the same range. Figure 7, where we study the area distribution of a number of polygon samples on two specific point sets, also makes it apparent that the area of samples generated by RPG-star is, on average, comparatively smaller than those of other samples. Looking at the extreme ends of the samples in Table 1(a), we note that within this test-run, the smallest area for the small



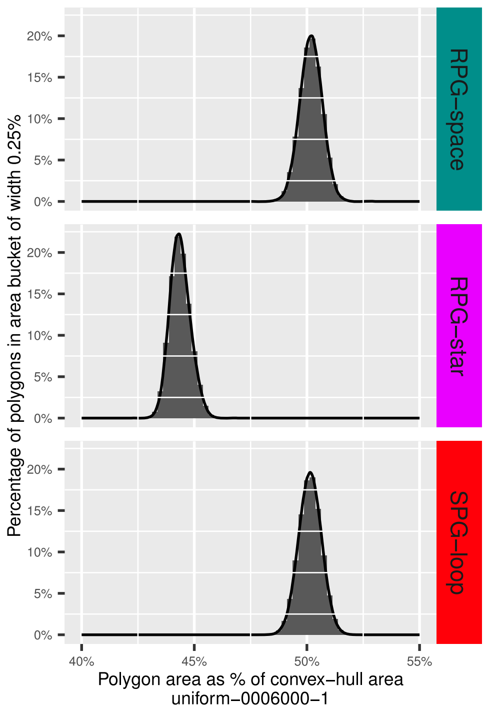


Fig. 7. Histograms of the area distribution of random polygons of the two input point sets `uniform-0000060-1` (see also Figure 8) and `uniform-0006000-1` created using RPG-space, RPG-star, and SPG-loop. Sample size was 10^8 and 10^6, respectively. See Table 1(a) for the smallest and largest area found.

Table 1. (a) Largest and Smallest Area Found (as Percentage of the Convex Hull) in the Test-runs from Figure 7, with the Cell of the Best Result Highlighted. Additionally, We Show Our Final Smallest and Biggest Area Found During the Competition After Running Our Optimizers on the Sampled Polygons. (b) Percentage Values for the Initial Polygonizations that Resulted in Min-Area/Max-Area Polygonizations

	uniform-0000060-1		uniform-0006000-1	
Heuristic	Min-Area	Max-Area	Min-Area	Max-Area
RPG-space	31.07%	73.33%	47.82%	52.54%
RPG-star	31.15%	64.60%	42.51%	46.43%
SPG-loop	30.37%	76.11%	47.89%	52.40%
optimized	15.37%	87.31%	19.48%	69.43%

(a)

Heuristic	Min-Area	Max-Area
2opt	22%	87%
PAO-mono	4%	2%
RPG-space	20%	11%
RPG-star	7%	—
LKH	47%	—

(b)

60-vertex input was actually found by SPG-loop. For the bigger point set, however, RPG-star wins the smallest sample by a huge margin.

Table 1(b) lists the percentage values for the ancestry of our final Min-Area/Max-Area polygonizations. (In that table all 2-opt variants of RPG-2opt, BF-2opt and Spg are summarized as 2opt.) We were surprised to learn that about 47 % of our final Min-Area polygonizations came from initial polygonizations obtained by running LKH, i.e., from (approximate) TSP tours, while none of our final Max-Area polygonizations stems from a TSP tour. Apparently, it was easier for Pao-Flip to trade large-area triangles for small-area triangles if the initial polygonization did not contain chains that zigzag wildly. At any rate, it seems that the initially smaller sizes for polygons produced by RPG-star did not help at the optimizing step in most cases. For some (very) small inputs

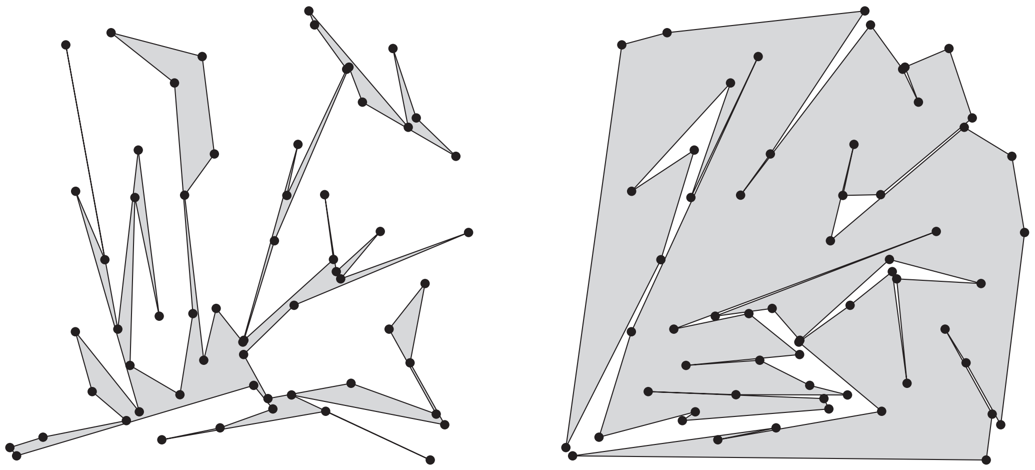


Fig. 8. Sample Min-Area/Max-Area polygonizations for `uniform-0000060-1` obtained by our flipping/inverting approach.

the initial polygonization produced by 2opt was already optimal or, at least, could not be improved by Pao-Flip.

Note that the Challenge did not specify a time limit per instance. As the only time limit was given by the final deadline for submitting our results, we kept applying our heuristics and optimization procedures for as long as the competition remained open and whenever we had some capacity to spare on our systems. Instances subject to further sampling or optimization runs were selected at random, with each individual job limited to an hour of CPU time.

5 DISCUSSION

Sample results for Min-Area/Max-Area polygonizations are shown in Figure 8. The surprisingly larger percentage of final Min-Area polygonizations derived from initial TSP tours made us reconsider our basic assumption that a Max-Area (respectively, Min-Area) polygonization would be obtained best by applying Pao-Flip to an initial polygonization with a reasonably large (respectively, small) area. Perhaps we should have been less greedy and should have applied Pao-Flip randomly to initial polygonizations, without consideration of whether or not the initial polygonization has a reasonably small or large area.

During the tail end of the competition time we noticed that the optimization using Pao-Flip can get stalled at local maxima or minima, with nothing to gain by further flipping. Future work could explore the use of simulated annealing (or similar) to allow Pao-Flip to get away from local minima/maxima. For instance, while looking for a Min-Area polygonization, one could allow some flipping that increases the area of the polygonization. We leave the exploration of these variants to future research.

We note that every polygonization has a positive probability to appear as one of our starting polygonization. (Recall that all 2-opt variants start by generating random sequences of the input points, which form the seed polygons for the subsequent heuristics.) From a theoretical point of view it is interesting to ask whether the space of all polygonizations is connected under the invert/flip operations discussed in Section 2.2. So, given two polygonizations $\mathcal{P}_1, \mathcal{P}_2$ on the same set of input points, is it always possible to derive $\mathcal{P}_2$ from $\mathcal{P}_1$ via a series of invert/flip operations? We suppose this to be true but can only prove it for the class of x-monotone polygonizations. And if this conjecture were true: What is a tight bound on the number of invert/flip operations needed in the worst case to derive $\mathcal{P}_2$ from $\mathcal{P}_1$?

REFERENCES

[1] Thomas Auer and Martin Held. 1996. Heuristics for the generation of random polygons. In *Proceedings of the 8th Canadian Conference on Computational Geometry (CCCG'96)*. 38–44.

[2] Jon L. Bentley and Thomas A. Ottmann. 1979. Algorithms for reporting and counting geometric intersections. *IEEE Trans. Comput.* 28, 9 (September 1979), 643–647. https://doi.org/10.1109/TC.1979.1675432

[3] G. A. Croes. 1958. A method for solving traveling-salesman problems. *Operat. Res.* 6, 6 (1958), 791–812. https://doi.org/10.1287/opre.6.6.791

[4] Loïc Crombez, Guilherme D. da Fonseca, and Yan Gerard. 2021. Greedy and local search solutions to the minimum and maximum area. *ACM J. Experimental Algorithmics* (2021).

[5] Erik D. Demaine, Sándor P. Fekete, Phillip Keldenich, Dominik Krupke, and Joseph S.B. Mitchell. 2021. Area-optimal simple polygonalizations: The CG challenge. *ACM J. Experimental Algorithmics* (2021).

[6] Günther Eder, Martin Held, Steinþór Jasonarson, Philipp Mayer, and Peter Palfrader. 2020. Salzburg database of polygonal data: Polygons and their generators. 31 (August 2020), 105984. https://doi.org/10.1016/j.dib.2020.105984

[7] Alfredo García, Marc Noy, and Javier Tejel. 2000. Lower bounds on the number of crossing-free subgraphs of K_N. *Comput. Geom.: Theory Appl.* 16, 4 (August 2000), 211–221. https://doi.org/10.1016/S0925-7721(00)00010-9

[8] Nir Goren, Efi Fogel, and Dan Halperin. 2021. Area-optimal polygonization using simulated annealing (unpublished).

[9] Martin Held. 2001. FIST: Fast industrial-strength triangulation of polygons. *Algorithmica* 30, 4 (2001), 563–596. https://doi.org/10.1007/s00453-001-0028-4

[10] Keld Helsgaun. 2000. An effective implementation of the lin-kernighan traveling salesman heuristic. *Eur. J. Operat. Res.* 126, 1 (Oct. 2000), 106–130. https://doi.org/10.1016/S0377-2217(99)00284-2

[11] Julien Lepagnot, Laurent Moalic, and Dominique Schmitt. 2021. Optimal area polygonization by triangulation and ray-tracing. *ACM J. Experimental Algorithmics*.

[12] Shen Lin and Brian W. Kernighan. 1973. An effective heuristic algorithm for the traveling-salesman problem. *Operat. Res.* 21, 2 (1973), 498–516. https://doi.org/10.1287/opre.21.2.498

[13] Natanael Ramos, Raí C. de Jesus, Pedro de Rezende, Cid de Souza, and Fábio L. Usberti. 2021. Heuristics for area optimal polygonizations. *ACM J. Experimental Algorithmics* (2021).

[14] Jonathan R. Shewchuk. 1996. Triangle: Engineering a 2D quality mesh generator and delaunay triangulator. In *Applied Computational Geometry: Towards Geometric Engineering*. Lecture Notes in Computer Science, Vol. 1148. Springer-Verlag, 203–222.

[15] Jan van Leeuwen and Anneke A. Schoone. 1982. Untangling a travelling salesman tour in the plane. In *Proceedings of the 7th Conference Graph-theoretic Concepts in Computer Science (WG'81)*, J. R. Mühlbacher (Ed.). 87–98.

[16] Chong Zhu, Gopalakrishnan Sundaram, Jack Snoeyink, and Joseph S.B.Mitchell. 1996. Generating random polygons with given vertices. *Comput. Geom.: Theory Appl.* 6, 5 (1996), 277–290. https://doi.org/10.1016/0925-7721(95)00031-3

Received January 2021; revised June 2021; accepted November 2021