Area Optimal Polygonization Using Simulated Annealing

NIR GOREN, EFI FOGEL, and DAN HALPERIN, The Blavatnik School of Computer Science, Tel Aviv University, Israel

We describe a practical method to find near-optimal solutions for the area-optimal simple polygonization problem: Given a set of points $\mathcal{S}$ in the plane, the objective is to find a simple polygon of minimum or maximum area defined by $\mathcal{S}$. Our approach is based on the celebrated metaheuristic Simulated Annealing. The method consists of a modular pipeline of steps, where each step can be implemented in various ways and with several parameters controlling it. We have implemented several different algorithms and created an application that computes a polygon with minimal (or maximal) area. We experimented with the various algorithmic options and with the controlling parameters of each algorithm to tune up the pipeline. Then, we executed the application on each of the benchmark instances, exploiting a grid of servers, to obtain near optimal results.

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:
Nir Goren, Efi Fogel, and Dan Halperin. 2022. Area Optimal Polygonization Using Simulated Annealing. *J. Exp. Algorithmics* 27, 2, Article 2.3 (February 2022), 17 pages.
https://doi.org/10.1145/3500911

1 INTRODUCTION

Given a set of points $\mathcal{S}$ in the plane, finding a simple polygon P of minimum area, such that every point in $\mathcal{S}$ appears exactly once in the boundary of P, is referred to as the ***Min-Area Polygonization*** **(MnAP)** problem. The similar problem, where the objective is finding a polygon of maximum area, is referred to as the ***Max-Area Polygonization*** **(MxAP)** problem.

Both MnAP and MxAP are combinatorial optimization problems; here we need to find an optimal object from a finite set of objects. Every set of n non-collinear points admits at least one polygonization. However, the number of different polygonizations of a set of n points is large. The trivial bound is $(n-1)!/2$. The quest for upper and lower bounds on this number has a long history; see, e.g., [18], and the references therein. Both MnAP and MxAP problems are NP-complete [7].

This work has been supported in part by the Israel Science Foundation (grant no. 1736/19), by NSF/US-Israel-BSF (grant no. 2019754), by the Israel Ministry of Science and Technology (grant no. 103129), by the Blavatnik Computer Science Research Fund, and by the Yandex Machine Learning Initiative for Machine Learning at Tel Aviv University.
Authors' address: N. Goren, E. Fogel, and D. Halperin, The Blavatnik School of Computer Science, Tel Aviv University, Tel Aviv, 6997801, Israel; emails: nirgoren@mail.tau.ac.il, efifogel@gmail.com, danha@post.tau.ac.il.

1084-6654/2022/02-ART2.3 $15.00
https://doi.org/10.1145/3500911

Therefore, exhaustive search is not tractable. Several approaches to solve combinatorial optimization problems with non-tractable search spaces, also referred to as metaheuristics, have been proposed [10]. We have opted for a variant of ***simulated annealing*** **(SA)** [13] (thus named because it mimics the process undergone by misplaced atoms in a metal when it is heated and then slowly cooled). Many of these approaches consist of two phases, and SA is not an exception. In the first phase a feasible initial object (a.k.a. state) is found. The next phase consists of a sequence of iterations that aims to converge to an optimal state. (While these techniques are unlikely to find an optimum solution, they can often find a very good solution.) Each iteration accepts a feasible state as input, applies a transition step, and results in a different feasible state, hopefully but not necessarily, of better quality compared to the input state. For simplicity of exposition, we assume henceforth that every iteration results in a feasible (valid) state; on rare occasions and for particularly small inputs, this may not be the case (and this can be easily detected and handled). However, for all our experiments reported below and for all the contest inputs, an iteration always ended with a valid new state.

Simulated annealing improves the exhaustive search strategy through the introduction of two key elements. The first is the so-called "Metropolis criterion" [15], in which some transitions that do not improve the state are accepted, as they enable the further exploration of the solution space. Such "bad" transitions are allowed under the condition that

$$e^{(-\Delta D/T)} > R(0,1), \tag{1}$$

where ΔD is the change of quality implied by the transition (negative for a "good" transition; positive for a "bad" transition), T is a "synthetic temperature", and $R(0,1)$ is a random number in the interval $[0,1]$. D is called a "cost function", and corresponds to the excess energy in the case of annealing a metal (in which case the temperature parameter would actually be kT, where k is Boltzmann's Constant and T is the physical temperature, in the Kelvin absolute temperature scale). If T is large, many "bad" transitions are accepted, and a large part of the solution space is accessed. The second key element is, again by analogy with annealing of a metal, the lowering of the "temperature". After making many transitions and observing that the cost function declines only slowly, one lowers the temperature, and thus limits the number of allowed "bad" transitions. After lowering the temperature several times, one may then "quench" the process by accepting only "good" transitions in order to find the local minimum of the cost function. There are various "annealing schedules" for lowering the temperature. Using different settings can increase the convergence rate.

We have created an application that computes near-optimal solutions for the MnAP and MxAP problems. The application consists of a modular pipeline formed of several steps, where each step can be implemented in various ways and is governed by several parameters. The first step in the pipeline accepts as input a set of points in the plane and produces as output a simple polygon defined by the input set. All other steps accept as input a simple polygon and produce as output a different simple polygon defined by the same set of points. The code depends on CGAL [20] **(Computational Geometry Algorithm Library)**. In particular, we have used the Linear Geometry Kernel package [1], the kD-tree search structure provided by the "dD Spatial Searching" package [19], and the convex hull computation from the "2D Convex Hulls and Extreme Points" package [12]. The code resides in a git repository publicly accessible at https://bitbucket.org/taucgl/optimal_area_polygonalization/; see the Wiki section within for instructions for building the application and executing it. We have used our application to compete in a contest that was part of the CG Challenge 2019 [5] and won third place. For the other triumphant participants see [2, 9, 14, 16].

In this article we describe the various algorithms we used and the considerations we took into account while implementing these algorithms, and executing their implementations. We also list the results that we have obtained while participating in the contest. Finally, we mention several ideas for improving our application, which we have conceived, but have not had the time to explore yet, and conclude with final remarks.

2 ALGORITHMS

When applying Simulated Annealing to solve an optimization problem, an initial state must be provided to start with. In the physical world metaphor this would be the heated metal. In our case, this is a valid polygonization of our input points. In addition, the process uses an operation that determines the cost of a given state. In the case of annealing a metal, this would be the excess energy of the metal that can be released. In our case, this is a measure that indicates how "bad" our polygonization is with respect to the objective function (being either the minimum area or the maximum area). Finally, a transition operation must be available. When the process is implemented in software, three corresponding functions listed below must be implemented.

(1) Computing an initial state.
(2) Performing a transition.
(3) Evaluating the quality of a given state.

2.1 Notations

Let $\mathcal{AR}(P)$ denote the area of a polygon P. Let $\mathcal{CH}(\mathcal{S})$ denote the convex hull of a point set $\mathcal{S}$. Let $\mathcal{CR}(P) = \frac{\mathcal{AR}(P)}{\mathcal{AR}(\mathcal{CH}(P))}$, referred to as the coverage ratio, denote the ratio between the area of the polygon P and the area of the convex hull of the polygon. Given a point set $\mathcal{S}$, let $P_{\mathcal{S}}^{\text{opt,max}}$ and $P_{\mathcal{S}}^{\text{opt,min}}$ be the maximum-area and minimum-area polygons defined by $\mathcal{S}$, respectively. Similarly, let $P_{\mathcal{S}}^{\text{best,max}}$ and $P_{\mathcal{S}}^{\text{best,min}}$ be the maximum-area and minimum-area polygons defined by $\mathcal{S}$ obtained by our application, respectively. Naturally, $0 < \mathcal{CR}(P_{\mathcal{S}}^{\text{opt,min}}) \leq \mathcal{CR}(P_{\mathcal{S}}^{\text{best,min}})$ and $\mathcal{CR}(P_{\mathcal{S}}^{\text{best,max}}) \leq \mathcal{CR}(P_{\mathcal{S}}^{\text{opt,max}}) \leq 1$.

2.2 Computing the Initial State

We have implemented two methods for computing an initial state—one that computes an x-monotone polygon and another that computes a star-shaped polygon; see Figure 1 for an illustration. Computing an x-monotone polygon is done by obtaining the lower hull of the input points, then sorting the remaining points in decreasing lexicographical order, and finally, connecting the two sequences. Special care must be taken in the degenerate cases where the first or last segment in one or both sequences is vertical. Computing a star-shaped polygon is done by computing the convex hull of the points, then choosing a point inside the convex hull (e.g., the centroid of the convex hull), then sorting the input points by their polar angle at the chosen point and connecting them in that order.

2.3 Simulated Annealing Transition Steps

A transition function accepts a valid state as input and results in a new valid state, hopefully but not necessarily, of improved (lower) cost compared to the input state. We devised two kinds of transition steps for the simulated annealing process, referred to as the *local* and *global* steps.

In a local step we swap the positions of a randomly chosen point q and its successor r, in the polygonal chain; see Figure 2. Let p be the point that precedes q, and let s be the point that succeeds r, in the polygonal chain. The resulting polygon is valid iff (a) the newly introduced segments $\overline{pr}$

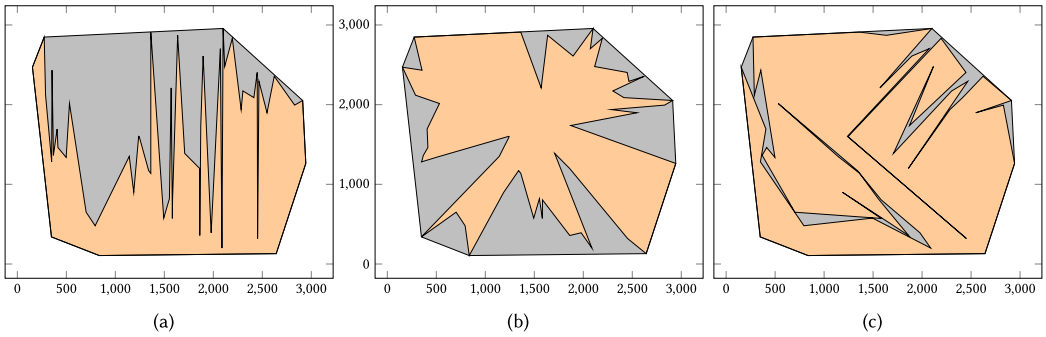


Fig. 1. Different polygonizations of the point set provided in the file `uniform-0000050-1.instance`. The obtained polygons are rendered in orange and the respective convex hulls excluding the polygons are rendered in gray. (a) An x-monotone polygon (54.93%). (b) A star-shaped polygon (61.83%). (c) The largest-area polygon obtained by our application (91.34%). The numbers in parentheses indicate the coverage ratio.

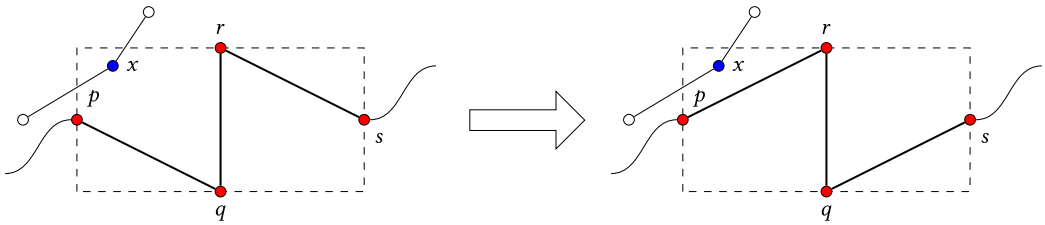


Fig. 2. A local transition step.

and $\overline{qs}$ do not intersect each other, and (b) these new segments do not intersect any other segment in the polygonal chain.

Observation 1. *If a segment $\overline{xy}$ intersects one of the two newly introduced segments ($\overline{pr}$ or $\overline{qs}$), then either the point x or the point y are located inside the open axis-parallel bounding rectangle of the points p, q, r, and s.*

Proof. Assume, that there exists a segment $\overline{xy}$ that intersects either $\overline{pr}$ or $\overline{qs}$. Since the input state is a valid polygonization, $\overline{xy}$ cannot intersect any segments in the input state, and in particular, none of $\overline{pq}$, $\overline{qr}$, or $\overline{rs}$; thus, it does not cross $\mathcal{CH}(p, q, r, s)$. Therefore, to intersect either $\overline{pr}$ or $\overline{qs}$, either x or y must be properly inside $\mathcal{CH}(p, q, r, s)$. As $\mathcal{CH}(p, q, r, s)$ is contained inside the bounding box B, point x or point y must be properly inside B. □

We efficiently check the validity of the resulting polygonization using a kD-tree search structure [4, Chapter 5]. We construct the kD-tree once as part of the initialization of the entire process and insert all input points into the tree. When a local transition step is carried out, we first randomly choose a point q in the current polygonal chain, and obtain a new polygon as described above. Then, we check the validity of the new polygon. In particular, we check whether the two newly introduced segments, namely $\overline{pr}$ and $\overline{qs}$, intersect each other, and we check whether either $\overline{pr}$ or $\overline{qs}$ intersect any other segment in the polygonal chain as follows: we submit a range-search query, where the query range is the rectangular axis-parallel bounding box of the points p, q, r, and s. For every point located in the range, we test whether one of the two segments incident to the point intersects one of $\overline{pr}$ or $\overline{qs}$. If an intersection is detected, the new polygon is discarded and the process is repeated. In practice we used the kD-tree provided by Cgal [19]. While the worst

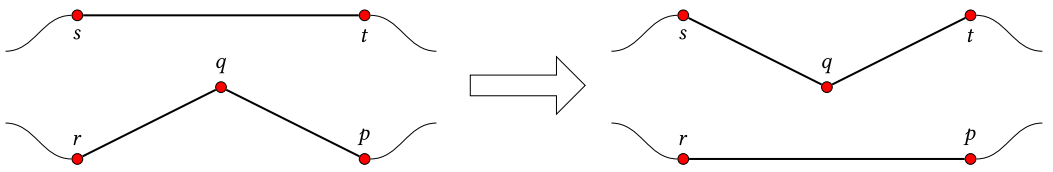


Fig. 3. A successful global transition.

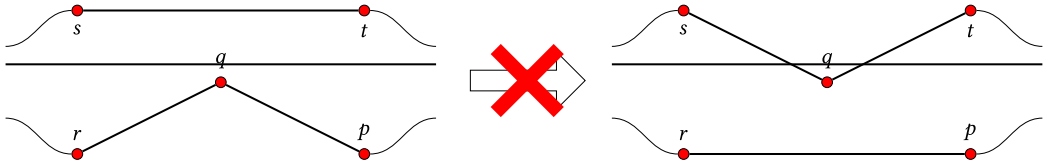


Fig. 4. An unsuccessful global transition.

case asymptotic running time of kD-tree queries is high [4, Chapter 5], in practice, as we observe here as well, queries are answered very quickly. The local transition step described above can be generalized; see Section 5.5. However, we have not implemented the generalization.

In a global step we randomly pick a point q, then randomly pick a point s. Let p and r be the points that precedes and succeeds q, respectively, and let t be the point that succeeds s, in the polygonal chain. We first connect p and r (removing q from the polygonal chain) and then insert p between s and t; see Figure 3. The resulting polygon is valid iff (a) the newly introduced segment $\overline{pr}$ does not intersect one of the other newly introduced segments $\overline{sq}$ and $\overline{qt}$, and (b) each of these new segments does not intersect any other segment in the polygonal chain.

In a global step there is no guarantee that an endpoint of a segment that intersects any one of the newly introduced segments is located inside the axis-parallel bounding box of the involved points, as illustrated in Figure 4. Thus, to check whether the resulting polygon is valid, we need to check whether any one of the newly introduced segments intersects any other remaining segment in the polygonal chain. Observe that a local step is a special case of a global step. If t and r denote the same point, then the linear time validity check can be replaced with the much more efficient validity check of a local step.

2.4 Spatial Subdivision

We have applied a spatial subdivision strategy described below to solve the problem at hand, subdividing the original problem into several subproblems once, and combining the solutions of the subproblems to yield the final solution. The input set of n points is subdivided into $k = \lceil (n-1)/(m-1) \rceil$ subsets of approximately m points each for a given integer number m, based on the lexicographic ordering of the points; see Figure 5. Every two subsets of points located in two adjacent vertical slabs, respectively, share a common point. We use a greedy approach to make sure that the rightmost segment of the lower hull of every subset, except for the last subset, is strictly monotone increasing, and similarly, that the leftmost segment of the lower hull of every subset, except for the first subset, is strictly monotone decreasing. More precisely, let $\mathcal{S}_i$ and $\mathcal{S}_{i+1}$ be two subsets of points located in two adjacent vertical slabs, $0 \leq i < k-1$, and let q_i be their common point. We make sure that there exist points $p_i \in \mathcal{S}_i$ and $r_i \in \mathcal{S}_{i+1}$, such that $y(p_i) < y(q_i)$ and $y(q_i) > y(r_i)$, where $y(p)$ denotes the y-coordinate of p. When we construct the subset $\mathcal{S}_i$, if $i = 0$, we initialize it with the first m points in the lexicographic ordering; otherwise, we initialize it with the rightmost point of $\mathcal{S}_{i-1}$, and append the first remaining $m-1$ points in the lexicographic ordering. We

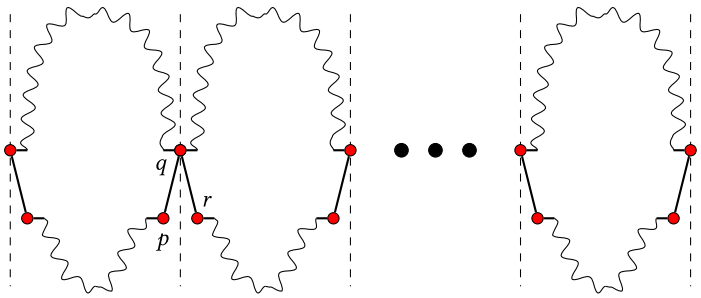


Fig. 5. Spatial subdivision.

incrementally extend $\mathcal{S}_i, 0 \leq i$ as long as the conditions above are not met. If the number of remaining points at the end is below some threshold, we append the remaining points to the last subset as well. Once the construction of the subsets is complete, we mark the rightmost segment of the lower hull of every subset except for the last subset, and the leftmost segment of the lower hull of every subset except for the first subset. Then, we solve the polygonization problem for each subset using simulated annealing, under the constraint that the marked segments appear in the partial results. We force the marked segments to be in the initial state and we never apply a transition that compromises this constraint. Finally, we easily merge the solutions of the subproblems. Let $\overline{pq}$ and $\overline{qr}$ be the marked segments in S_i and S_{i+1}, respectively. We remove the segments $\overline{pq}$ and $\overline{qr}$ and instead introduce the segment $\overline{pr}$. We have not implemented other subdivision and merging strategies. Naturally, applying more sophisticated strategies may improve the final results; see Sections 5.1 and 5.2.

3 ALGORITHM ENGINEERING

As mentioned above, the application consists of a modular pipeline formed of several steps, where each step can be implemented in various ways and its execution is governed by several parameters. In the following we describe the structure of the pipeline in detail.

3.1 Program Structure and Options

The application supports several command line options that can be used to set the parameters that govern the pipeline execution. In particular, one option enables the application of the spatial subdivision, and another option sets the size of the subsets in case spatial subdivision is indeed enabled. Another option selects the algorithm used to compute the initial state; see Section 3.2. Other options control the usage of local and global steps, e.g., the type of steps and the number of steps. Several additional options control the simulated annealing process, e.g., the rate of cooling. Using command line options, the user can select the geometric kernel and the number type used; see Section 3.7. Finally, there exists an option that indicates whether to bypass the computation of the initial state. Bypassing the computation of the initial state is allowed under the precondition that the input consists of a valid simple polygon. This option enables progressive refinement of the result using successive executions.

We have experimented with the various parameters and attempted to predict the best settings for the various input instances. We found out that the following strategy achieves good results for all instances. First we apply spatial subdivision solving the polygonization problem on each subset and merging the individual intermediate results. We bound the subset size to enable the relatively quick application of simulated annealing mainly using global steps. Recall, that the validity check of a global step takes time linear in the size of the input, which is expensive; thus, applying a

large sequence of global steps on a large input set may be prohibitively time consuming. Then, we continue applying simulated annealing, this time on the merged result, mainly using local steps.

3.2 Computing the Initial State

The first phase in the pipeline accepts as input a set $\mathcal{S}$ of n points and computes an initial state, that is, a polygonization of $\mathcal{S}$ that yields a simple polygon. For inputs with points distributed uniformly in a square, the area of the polygon obtained by the first method was observed to be roughly 50% of the area of the convex hull of the input points, compared to 44% with the second method (when the pivot is chosen to be the lexicographically middle point between the leftmost and rightmost points in the input). However, no significant difference has been observed between the results obtained after applying the simulated annealing process on the polygons obtained by either methods. The first method allowed for a simple implementation of the spatial subdivision strategy, and in particular of the merging sub-strategy described in Section 2.4, and thus was the one used for most input instances.

3.3 Evaluating the Quality of a State

The energy of a state P for MnAP is set to be proportional to $\mathcal{CR}(P)$. Empirically, we have noticed that the difference between the coverage ratios of consecutive states P_i and P_{i+1} (where P_{i+1} is obtained as a result of either a local or a global transition step) decreases with the increase in the input-set size. Thus, the exact energy is set to be the coverage ratio multiplied by the input-set size. Applying this amplification was justified by our experiments, as it increased the convergence rate. More precisely, for MnAP the energy of a state P, which is a polygonization of a point set $\mathcal{S}$ of size n, is given by the expression $n \times \mathcal{CR}(P)$. For MxAP the energy of a state P is given by $n \times (1 - \mathcal{CR}(P))$. Recall, that the algorithm only requires computing the energy difference, which is $n \times \Delta\mathcal{CR}(P)$ and $n \times -\Delta\mathcal{CR}(P)$ for MnAP and MxAP, respectively.

Calculating the difference in the areas of two consecutive states is done in constant time, simply by adding and subtracting the area of the relevant triangles. For example, if the new state is obtained as the result of the local transition step illustrated in Figure 2, then the area of the triangle (s, r, q) is added and the area of the triangle (p, q, r) is subtracted (assuming that the polygon interior is to the left of the directed polygonal chain). If the new state is obtained as the result of the global transition step illustrated in Figure 3, then the area of the triangle (p, q, r) is added and the area of the triangle (s, q, t) is subtracted.

3.4 Tuning the Simulated Annealing Parameters

The starting temperature is chosen to be 1. The simulated annealing process is terminated after a specified number of iterations ℓ (transition steps). (This was the only termination criterion we used.) As the number of simulated annealing iterations is predetermined and set by the user, the temperature is decreased by $1/\ell$ at the end of each iteration. We apply a transition if the difference between the energy of the source and target states, ΔD, is negative, or ΔD is non-negative and Condition 1 (Metropolis) holds.

3.5 Tuning the Subset Sizes for the Spatial Subdivision

Using global steps achieves much better results compared to using local steps when applying the same number of steps. However, due to the low probability of producing a (valid) step for large point sets, and the high (linear) time consumption of the validity check for large point sets (which must be performed for each potential transition), it proved to be an effective strategy for input sets containing not more than few hundreds of points. We tried to strike a balance between the

size of the subsets for each instance, and the number of (global) steps applied to each subset. With smaller subset sizes, the best area this method is able to obtain is further from the true optimum, since the search space is more limited and the simple merging process we used does not aim to optimize the area of the merged polygon. However, the larger the subset is, the more time it takes to validate a potential global transition.

We quickly executed our program on small size input sets (smaller than 1,000) without applying spatial subdivision to solve the MxAP problem, and extracted preliminary values of the computed coverage ratio $\mathcal{CR}(P_{\mathcal{S}}^{\text{best,min}})$, averaged over the size of the fed input point set (see Figure 6 for the final values). We noticed that even for these small size input sets, we hardly obtained more than 90% coverage. Then, we tuned the subset size as follows: We observed that the time it takes to achieve a certain coverage using global steps is roughly cubic in the number of points. Recall, that the time required to validate a step grows linearly with the input size. Both the total number of (valid) steps required and the average number of potential transitions needed to find a new valid state were observed to grow linearly with the input size as well. We estimated the number of steps required and the time it would take to achieve 90% coverage for a subset of a certain size. For each instance, we aimed to split it into as large as possible subsets so that for each, 90% coverage would be obtained in the allotted time. We did not apply spatial subdivision for instances of sizes smaller than 1,000. We used the same subset sizes that we used to solve the MxAP, to solve the MnAP problem.

3.6 Data Structures

When spatial subdivision is applied, we subdivide the input set into subsets. Then, we compute an initial polygon for each subset and store the points of the polygonal chains in corresponding lists. In addition, for each subset of size m we maintain an array that stores the m iterators to the points in the list. When we perform a transition as part of the application of simulated annealing on a subset, we quickly update the list that stores the polygonal chain (that is, the current state), and the relevant entries in the array. We use the array to efficiently access random points as part of the transition steps. When we change the position of a point in the list, we use the index stored in the record that extends the point type to access the corresponding entry in the array and update it; see Section 3.7.

3.7 Geometric Kernel and Number Type

Our application includes conditional steps that are based on the evaluations of predicates, carried out through the computation of the sign of an arithmetic expression. For example, the computation of the convex hull of the input points, requires the computation of the orientation of three points, which requires the calculation of the sign of the determinant of a matrix that consists of the point coordinates. Such predicates are sensitive to limited precision arithmetic, and constitute a genuine threat to program correctness and normal termination when limited precision arithmetic is actually used.

The coordinates of the points in the input sets are even integral values. While the area is ensured to be an integral value [17], overflow may still occur while evaluating an arithmetic expression during the execution of the application. To overcome this potential pitfall, we based our implementation on CGAL, which follows the **Exact Geometric Computation (EGC)** paradigm, and thus guarantees robust execution (and exact results).

The developed software adheres to a generic programming paradigm; it enables the convenient selection of a geometric kernel and a specific number type; see, e.g., [8, Chapter 1], for details. In particular, we used an extended filtered Cartesian kernel, provided by CGAL [1], instantiated

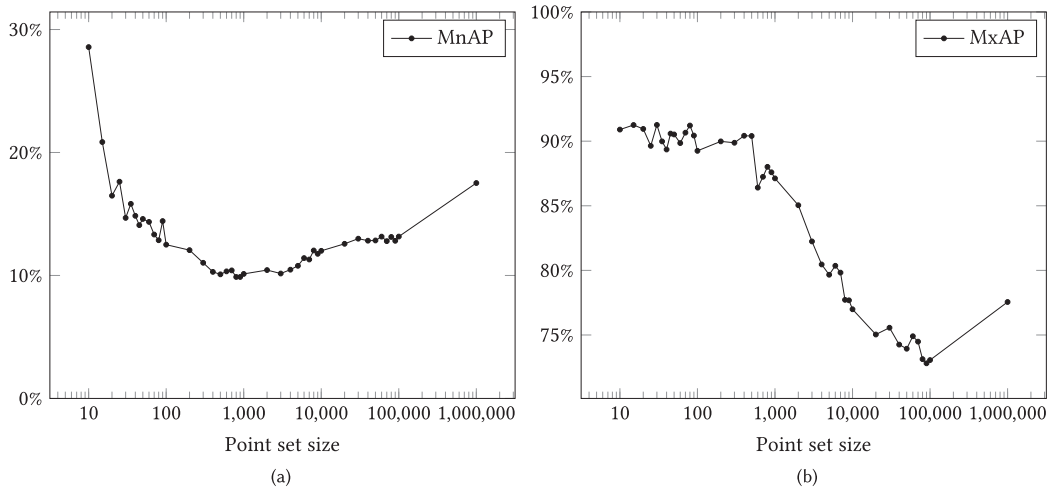


Fig. 6. The percentage computed coverage ratio, averaged over the different instances of the same input size, as a function of the point set size for MnAP (a), and for MxAP (b). Note that the horizontal axis is on a logarithmic scale.

with either the `long` standard number type or the `Gmpz` number type from the The **GNU Multiple Precision Arithmetic (GMP)** library [11]. The former target type typically has a width of 64 bits. The latter target type is of unlimited precision (up to memory limitation). For all the given instances, the limited precision type turned out to be sufficient. However, future instances may cause overflow, and thus may require the selection of the unlimited precision based kernel.

The point type of the kernel is extended with a record that contains several fields as follows: (a) The index to the point in the input sequence of points. The output format (defined by the contest rules) was a permutation of the input points given as a sequence of indices. The index above was used to construct this permutation from the resulting sequence of points (defining the final polygonal chain) in linear time. Upon initialization, we store the input points in a list, if spatial subdivision is suppressed, or in several lists otherwise, and set their indices. (b) A Boolean flag that indicates whether the point lies on the lower hull. This flag is used when constructing an x-monotone polygon, which serves as an initial state. (c) An enumeration that indicates whether the point is leftmost or rightmost in the subset, and thus shared with the left or right neighboring subset, respectively, or whether the point is the other endpoint of the leftmost or rightmost segment of the lower hull. Recall, that this segment must remain in the polygonal chain; see Section 2.4. (d) An integral value used in the computation of a constrained triangulation; see Section 5.3.

4 RESULTS

We have divided the entire set of instances into subsets of identical point-set sizes. Notice that the subset of instances of the largest size, that is 1,000,000 points, consists of a single member. All the experiments described in this section apply to these subsets. Given a subset $\mathcal{I}$ of a specific size, we either compute the average of the percentage coverage ratio for the members of $\mathcal{I}$ or provide the worst coverage ratio or the best coverage ratio. For MnAP (resp. MxAP) the worst and best are the largest (resp. smallest) and smallest (resp. largest), respectively. We used an Intel Core i5 clocked at 3.5GHz with 16GB of RAM to obtain the benchmarks; see Figures 8 and 9.

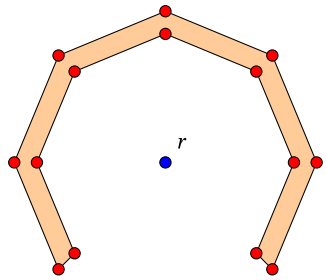


Fig. 7. The polygonization of 14 points with the minimum coverage ratio. Adding the new point p increases the minimum coverage ratio.

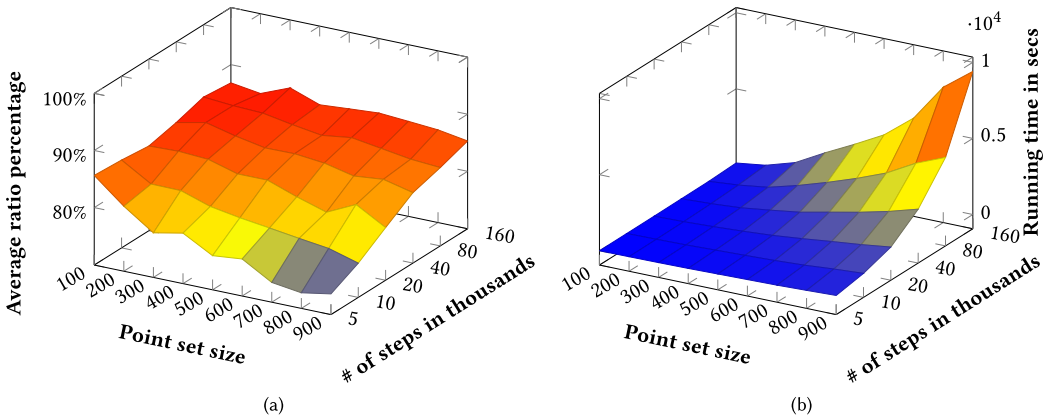
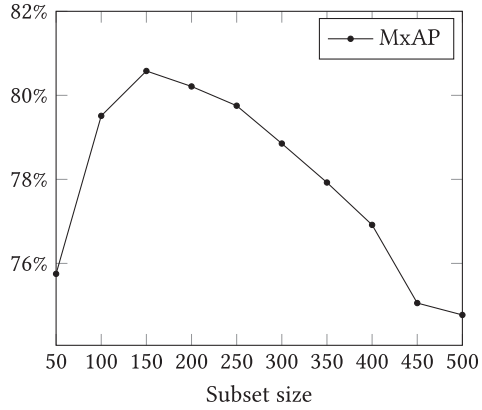


Fig. 8. The percentage computed coverage ratio (a) and the running times (b) as a function of the point set size and the number of successive global transition steps for MxAP. Note that the y-axis (which measures steps in thousands) is on a logarithmic scale.

Figure 6(a) shows a graph of the computed coverage ratio $\mathcal{CR}(P_{\mathcal{S}}^{\text{best,min}})$, averaged over the different instances of the same input size, as a function of the input set size for MnAP. The graph is roughly monotone decreasing in the range where the point set size is smaller than 1,000. We believe that the graph of the true optimum is monotone decreasing throughout (at least for points chosen uniformly at random). Notice though that there are cases where adding a point to a given set of points $\mathcal{S}$ inside the convex hull of $\mathcal{S}$ increases the area of the minimum-area polygon of $\mathcal{S}$; see Figure 7. In our results, the graph increases past 1,000 points probably due to limitations of the methodology. Similarly, Figure 6(b) shows a graph of the computed coverage ratio $\mathcal{CR}(P_{\mathcal{S}}^{\text{best,max}})$, averaged over the different instances of the same input size, as a function of the input set size for MxAP. Observe that spatial subdivision was not applied for instances of sizes smaller than 1,000. The sudden increase of the coverage for 1,000,000 points can be accounted for the fact that, as noted above, the set of instances includes only a single input set with 1,000,000 points, which were not uniformly sampled at random, as well as for the relatively large amount of sources we invested in this case. The computed coverage ratio $\mathcal{CR}(P_{\mathcal{S}}^{\text{best,min}})$, averaged over the different instances of the same input size, as a function of the input set size, behaves differently for the MnAP and MxAP problems. It hints, that the same strategy is not equally suitable for the two different problems, and a better approach would have been to treat each of them separately.



(a)

Subset Size	# of Global Steps	Time (secs)	Coverage %
50	50000	444.152	75.7512
100	25000	377.554	79.5115
150	16667	413.646	80.5798
200	12500	417.640	80.2118
250	10000	394.022	79.7532
300	8333	422.329	78.8530
350	7143	450.608	77.9252
400	6250	398.978	76.9165
450	5556	430.564	75.0562
500	5000	407.572	74.7731

(b)

Fig. 9. (a) The percentage coverage ratio as a function of the subset size of different subdivisions for MxAP (computed only for `uniform-0002000-1.instance`). (b) The same information including the number of global steps and the running times.

We have created benchmarks using the uniform instances with point-set sizes of $100, 200, \ldots, 900$ for the global transition steps we implemented for MxAP. The benchmarks show the effect of the input size and the number of successive steps on (i) the quality of the result (Figure 8(a)), and (ii) the running times (Figure 8(b)). Naturally, increasing the number of steps improves (increases for MxAP) the coverage ratio. An examination of the numbers reveals that the coverage ratio is approximately $n^{0.05}$ where n is the number of steps for the point set size of 900. This figure monotonically decreases with the decrease of the point set size, and reaches approximately $n^{0.01}$ for the size of 100. Evidently, the time consumption is approximately linear in the number of steps and quadratic in the point set size, as expected.

We have also created benchmarks using the `uniform-0002000-1.instance` input file (which has 2,000 points) to measure the effect of the size of the subproblems in the spatial subdivision (see Section 2.4) on the quality of the results for MxAP. Recall that solving each subproblem is done by applying global steps, as these steps are more effective. Observe that as long as the sizes of the subproblems are limited, the running times remain under control. In an attempt to obtain comparable benchmarks with approximately the same running times, we have linearly decreased the number of steps with the increase in the subproblem size; see Table 9(b). As can be seen in Figure 9(a), dividing the input into subproblems of 150 points each yields the best coverage ratio for this specific instance and for a specific amount of allotted time. The optimal subproblem size is likely to grow with the increase in the number of global steps (and the consequent growth in running times). Also, the overhead and potential loss caused by the merging process lessens with the growth of the subproblem size. Indeed, for the competition we used different subproblem sizes for different instances based on their total size; see Section 3.4.

We executed our application in a **High Throughput Computing (HTC)** environment controlled by a software tool called HTCondor[1] and supported by the school of Computer Science at Tel Aviv University. The tool, among other things, harnesses wasted CPU power from otherwise idle workstations, and enables grid computing. The contest included 247 instances. We submitted two jobs for each instance, which executed our application to solve the MnAP and MxAP problems,

[1] https://research.cs.wisc.edu/htcondor/.

Table 1. Our Results for the Polygonization of Several Instances Provided as Part of the CG:SHOP 2019 Contest

Instance Name	Subset size	MnAP		MxAP	
		Coverage %	# of global steps	Coverage %	# of global steps
euro-night-0000100	100	12.49	100,000	93.19	100,000
paris-0001000	500	9.30	1,000,000	87.81	1,000,000
uniform-0010000-1	200	16.55	1,000,000	80.53	1,000,000
world-0010000	250	9.85	2,000,000	87.15	2,000,000
sun-0050000	200	12.13	5,000,000	72.56	5,000,000
uniform-0100000-2	150	16.62	13,333,000	79.86	13,333,000
mona-lisa-1000000	100	17.52	100,000,000	77.55	100,000,000

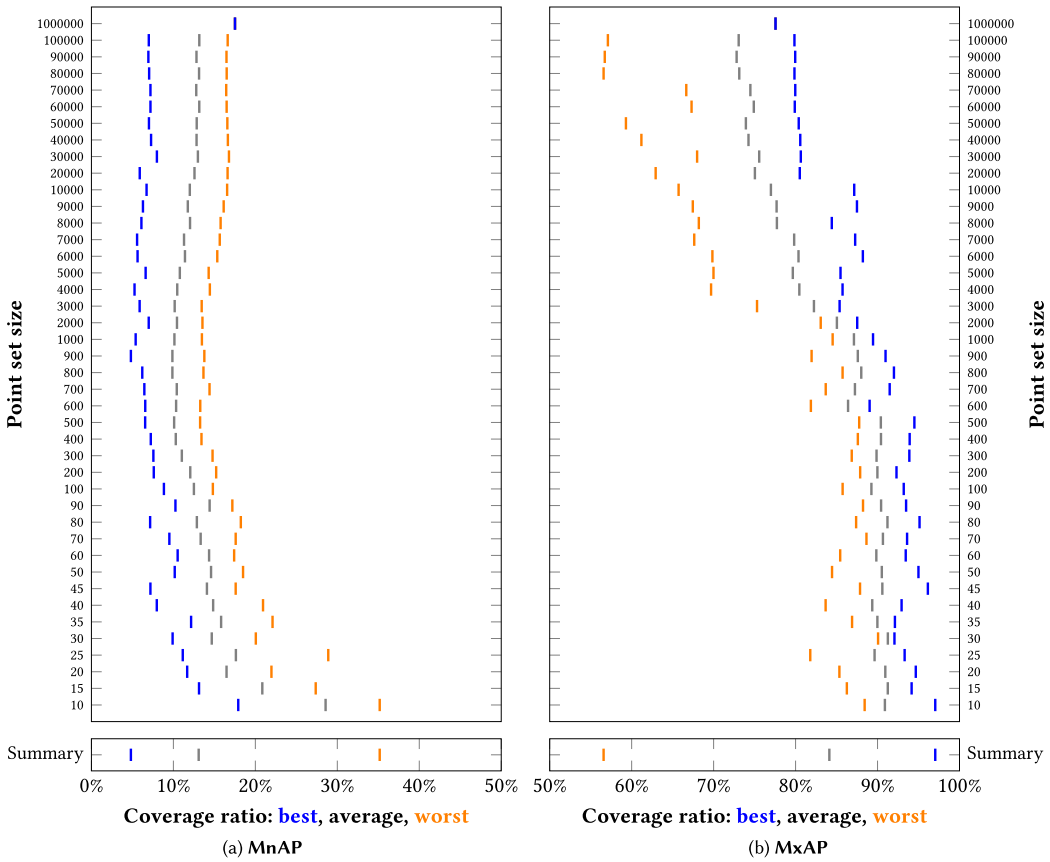


Fig. 10. Final results. In the summary row (at the bottom of the charts), the worst, best, and average values correspond respectively to the worst, best, and average coverage ratios over the coverage ratios for all instances. As there is a single instance with 1,000,000 points, the best, worst, and average values are equal.

respectively. We saved the results and then executed additional jobs this time without applying spatial subdivision and bypassing the computation of the initial state. We repeated the above for instances that exhibit progress above a certain threshold. Table 1 lists selected results. Figure 10 shows the final results as bar charts.

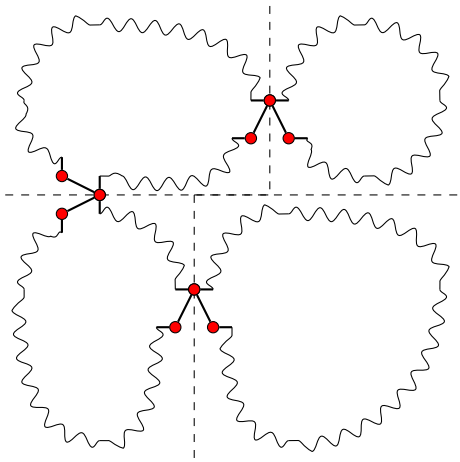


Fig. 11. Spatial subdivision with kD-tree like subdivision.

5 CONCLUSIONS AND OUTLOOK

In this final section we outline suggestions for further improving our solution pipeline and conclude with final remarks.

5.1 Improving the Subdivision Strategy

The spatial subdivision strategy is a combination of a specific spatial strategy and the sizes of the subsets. We have implemented only one strategy, where the input points are subdivided into subsets of approximately equal sizes located in distinct vertical slabs. For input instances with points distributed non-uniformly, subdividing the input set into subsets of different sizes could be advantageous. We believe that different spatial subdivision strategies can result with significant improvements. For example, instead of subdividing the points into subsets located in distinct vertical slabs, the points can be subdivided into subsets, such that the convex hulls of the subsets are pairwise disjoint, except, perhaps, at a single point (depending on the merging strategy); see Figure 11. When the points are subdivided using axis-aligned halfplanes (as depicted in the figure) the subdivision operation resembles the construction of a kD-tree. Finally, with sufficient resources, several strategies can be executed in parallel. For example, we can execute the currently implemented strategy and in parallel, a similar strategy, where the points are subdivided into subsets located in distinct horizontal slabs. Then we can compare the results, and retain the polygonization with the best quality among these executions, as the input to the next step in the pipeline, see Section 5.4.

5.2 Improving the Merging Strategy

The naive merging strategy currently implemented, which forces specific segments to be part of the resulting polygonization, is straightforward and free of degenerate cases, and thus, easily implemented. However, it may come at a cost, which could be significant. A more sophisticated strategy that does not impose any constraints or at least imposes relaxed constraints may prove advantageous.

First we can relax the constraints as follows: We divide the input point set into k strictly disjoint subsets $\mathcal{S}_0, \mathcal{S}_1, \ldots, \mathcal{S}_{k-1}$ of distinct points, such that the points of $\mathcal{S}_i$ and $\mathcal{S}_{i+1}$, $0 \leq i < k-1$, are located in adjacent vertical slabs. For each subset $S_i, 0 \leq i < k-1$, we find the rightmost point q_i.

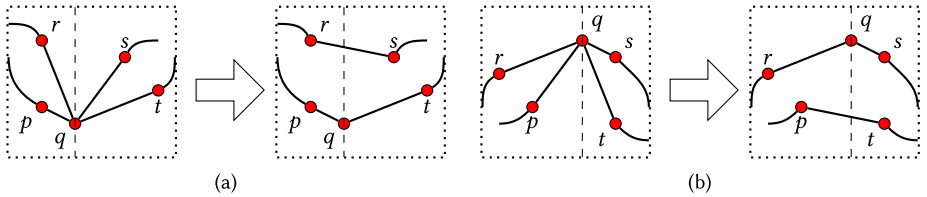


Fig. 12. Spatial subdivision, merging subsets.

Then, we find the rightmost segment $\overline{p_i q_i}$ of the lower hull of S_i and the rightmost segment $\overline{q_i r_i}$ of the upper hull of S_i. Similarly, we find the leftmost $\overline{q_i t_i}$ segment of the lower hull of $S_{i+1} \cup \{q_i\}$ and the leftmost segment $\overline{s_i q_i}$ of the upper hull of $S_{i+1} \cup \{q_i\}$. If the segment $\overline{p_i t_i}$ does not intersect $\overline{q_i r_i}$, we mark the segments $\overline{p_i q_i}$ and $\overline{q_i t_i}$; otherwise, we mark the segments $\overline{q_i r_i}$ and $\overline{s_i q_i}$, as they cannot intersect $\overline{s_i r_i}$. Then, we solve the polygonization problem for $\mathcal{S}_0$ and for each $\mathcal{S}_{i+1} \cup \{q_i\}, 0 \leq i < k-1$ as originally done. Finally, we easily merge the solutions of the subproblems; see Figure 12. For each pair of neighboring subsets $\mathcal{S}_i$ and $\mathcal{S}_{i+1}$, we remove the two marked edges, with common point q_i, from $\mathcal{S}_i$ and $\mathcal{S}_{i+1}$, respectively, and instead connect the other endpoints of the two segments. The above requires a new method for computing an initial state, as the two marked segments in a given subset may belong to distinct semi-hulls (lower and upper).

Secondly, we can try to lift the constraints altogether as follows: We solve the polygonization problem for each subset as originally done without the marking of any segment. Then, we attempt to merge the polygons of neighboring subsets. If the merging attempt fails, we resort to the aforementioned constraint-based strategy. The merging of two polygons P and Q can be done by searching for two consecutive points p_1, p_2 in P and two consecutive points q_1, q_2 in Q, such that the segments $\overline{p_1, q_2}$ and $\overline{p_2, q_1}$ do not intersect each other nor any other segment of P and Q.

5.3 Expediting the Execution of Global Steps

Global steps proved to be more effective than local steps, and thus, we based the size of the subsets in the spatial subdivision on the estimated time consumption of global steps; see Section 3.5. Expediting the execution of global steps, that is, increasing the probability of producing a valid transition and improving the performance of the validity check, would have allowed us to increase the size, and in turn achieve better results.

Checking the validity of a potential global transition takes linear time: We check that the three new segments do not intersect each other nor any of the other segments. We made an attempt to expedite the global transition steps. It is described below. We maintain a constrained triangulation of the convex hull of the input points, such that the edges of the current polygonal chain are assured edges of the triangulation. A triangle in the triangulation is either internal or external to the current polygon and is colored accordingly. When a transition step is carried out we update the coloring of the affected triangles; see Figure 13. There are two symmetric cases. We start with the first case; see Figure 13(a). When we are about to apply a global step we search for a vertex v_q, referred to as the pivot vertex, that is incident to an internal triangle t_i and an external triangle t_e, such that (i) t_i is an ear of the polygon,[2] (ii) $\overline{v_s v_t}$ is an edge in the polygonal chain, and (iii) the area of t_e is larger than the area of t_i. We can use a metaheuristic approach here as well, replacing condition (iii) with the Metropolis criterion to accept "bad" transitions, and lowering the synthetic temperature to decrease the rate of acceptance of "bad" transitions. Observe that when t_i and t_e

[2]Three consecutive vertices v_{i-1}, v_i, v_{i+1} along the boundary of a polygon form an ear if the line segment $\overline{v_{i-1} v_{i+1}}$ is fully contained inside the polygon.

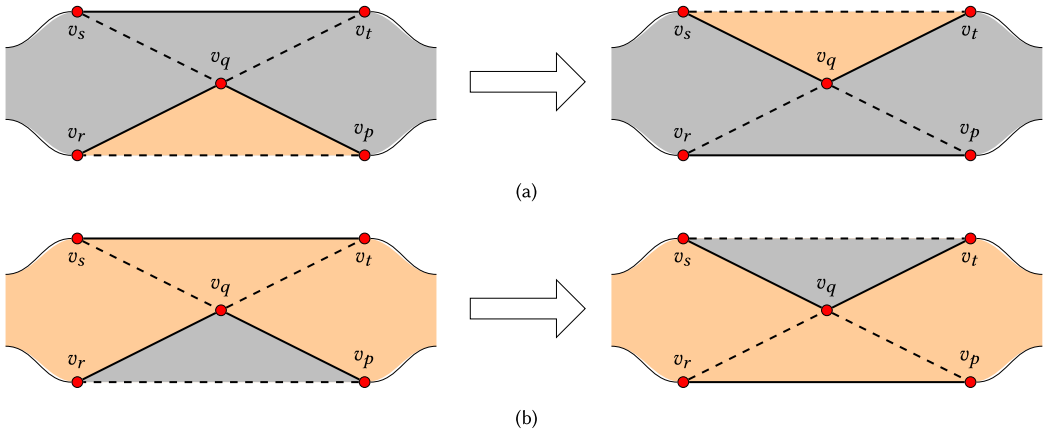


Fig. 13. Triangulations of the Convex Hulls. The orange triangles are internal and the gray triangles are external. (a) The internal triangle (v_p, v_q, v_r) and the external triangle (v_s, v_q, v_t) switch roles. (b) The external triangle (v_p, v_q, v_r) and the internal triangle (v_s, v_q, v_t) switch roles.

share an edge, the global transition reduces to a local transition. We continue with the second case; see Figure 13(a). We search for an external triangle t_e and an internal triangle t_i, such that (i) t_e is an ear of the complement of the polygon, (ii) $\overline{v_s v_t}$ is an edge in the polygonal chain, and (iii) the area of t_i is larger than the area of t_e. Clearly, we can use a metaheuristic approach here as well.

The main problem with the above approach is that there might be vertices in the current polygon that would ideally serve as the pivot vertex, but the triangulation currently in place does not allow the selection of any one of them. There are several options to remedy this situation or at least relax the search condition at the account of performance to increase the chances of revealing an appropriate pivot vertex as follows. (a) Re-triangulate the entire point set every once in a while with the edges of the current polygon as the new constraints. (b) Maintain a triangulation of the polygon only, and only search for a vertex v_q that is incident to a triangle $t = (r, q, p)$ that is an ear. This, at least, guarantees that the segment $\overline{pr}$ does not intersect any other segment. In addition, re-triangulate the polygon every once in a while using the so called *ear clipping* (a.k.a. ear trimming) algorithm [6]. This triangulation, by definition, favors ears, and as such increases the chances of revealing potential pivot vertices. (c) Maintain a triangulation of the complement of the polygon only, and only search for a vertex v_q that is incident to a triangle that is an ear of the polygon complement. In addition, re-triangulate the complement of the polygon every once in a while using the *ear clipping* algorithm. (d) Alternate between (a) and (b). (e) Maintain several constrained triangulations.

5.4 Parallelizing the Application

The code can be parallelized in several ways. First, when applying spatial subdivision, the processing of each subset is independent of the processing of all other subsets; thus, the processing of all subsets can be carried out simultaneously exploiting available processors. Secondly, we can use a parallel version of the simulated annealing algorithm exploiting various techniques [3]. In particular, we can launch several threads, which, e.g., execute different transition steps or different subdivision strategies, and let each operate until a termination condition is met (e.g., a specified temperature is reached, a specified amount of time has elapsed, or a number of transitions has been successfully completed). Then, pick the state with the best quality among all threads, and repeat the process.

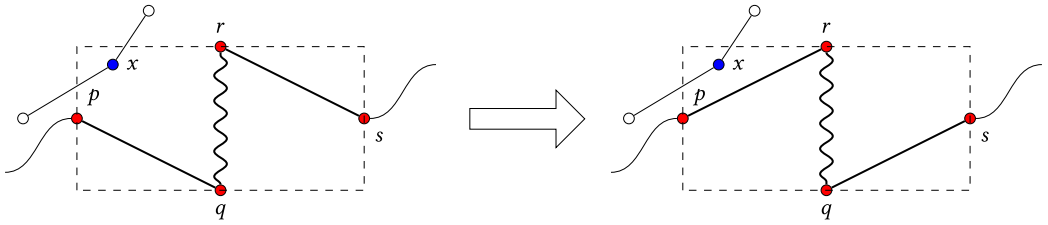


Fig. 14. A generalized local transition step.

5.5 Local Step Generalization

The local transition step can be generalized as follows: Instead of randomly choosing a single point (and swap it and its successor), we randomly choose two arbitrary points and reverse the order of the polygonal chain between them; see Figure 14. Checking the validity of a potential generalized local transition can be done efficiently in a similar way to the validation performed for the simplified (applied) local transition step. Observation 1 and its proof are trivially generalized. Computing the area difference between two consecutive states, where the latter is obtained as a result of a generalized local transition step, requires the addition and subtraction of the area of the polygons bounded by $(s, r, \ldots, q)$ and $(p, q, \ldots, r)$, respectively.

5.6 Final Remarks

The MnAP and MxAP problems are examples of geometric optimization problems that are difficult to solve in theory and in practice. Both problems are NP-complete, and while the solution space is discrete and finite, it is enormous. However, solving the problem, which seemed elusive at first, turned out to be feasible after all even for very large instances. Naturally, solving such problems requires harnessing several concepts from various disciplines, and this is exactly what we did: we use simulated annealing, convex hull computation, constrained triangulation, spatial subdivisions, kD-tree search structures, geometric kernels, and when all this was insufficient, we exploited a grid of computers leveraging their combined computation power.

REFERENCES

[1] Hervé Brönnimann, Andreas Fabri, Geert-Jan Giezeman, Susan Hert, Michael Hoffmann, Lutz Kettner, Sylvain Pion, and Stefan Schirra. 2020. 2D and 3D linear geometry kernel. In *CGAL User and Reference Manual* (5.0.2 ed.). CGAL Editorial Board. https://doc.cgal.org/latest/Manual/packages.html#PkgKernel23.

[2] Loïc Crombez, Guilherme D. da Fonseca, and Yan Gerard. 2021. Greedy and local search solutions to the minimum and maximum area polygons. *The ACM J. of Experimental Alg.* (2021). to appear.

[3] Van-Dat Cung, Simone L. Martins, Celso C. Ribeiro, and Catherine Roucairol. 2002. Strategies for the parallel implementation of metaheuristics. In *Essays and Surveys in Metaheuristics*, Vol. 15. Springer, 263–308. https://doi.org/10.1007

[4] Mark de Berg, Mark van Kreveld, Mark H. Overmars, and Otfried Cheong. 2008. *Computational Geometry: Algorithms and Applications* (3rd ed.). Springer.

[5] Erik D. Demain, Sándor P. Fekete, Phillip Keldenich, Dominik Krupte, and Josepth S. Mitchell. 2021. Area-optimal simple polygonalizations: The CG challenge 2019. *The ACM J. of Experimental Alg.* (2021). to appear.

[6] Hossam ElGindy, Hazel Everett, and Godfried Toussaint. 1993. Slicing an ear using prune-and-search. *Pattern Recognition Letters* 14, 9 (1993), 719–722. https://doi.org/doi:10.1016/0167-8655(93)90141-y

[7] Sándor P. Fekete. 2000. On simple polygonalizations with optimal area. *Disc. Comput. Geom.* 23 (2000), 73–110. https://doi.org/doi.org/10.1007/PL00009492

[8] Efi Fogel, Dan Halperin, and Ron Wein. 2012. *CGAL Arrangements and Their Applications, A Step by Step Guide.* Springer, Berlin, Germany.

[9] Natanael Ramos, Raí Caetano de Jesus, Pedro J. de Rezende, Cid C. de Souza, and Fábio Luiz Usberti. 2021. Heuristics for area optimal polygonizations. *The ACM J. of Experimental Alg.* (2021). to appear.

[10] Michel Gendreau and Jean-Yves Potvin. 2003. *Handbook of Metaheuristics*. Springer, International Series in Operations Research & Management Science.
[11] Torbjörn Granlund and the GMP development team. 2020. *GNU MP: The GNU Multiple Precision Arithmetic Library* (6.2.0 ed.). http://gmplib.org/.
[12] Susan Hert and Stefan Schirra. 2020. 2D convex hulls and extreme points. In *CGAL User and Reference Manual* (5.0.2 ed.). CGAL Editorial Board. https://doc.cgal.org/5.0.2/Manual/packages.html#PkgConvexHull2.
[13] Scott Kirkpatrick, Charles D. Gelatt, and Mario P. Vecchi. 1983. Optimization by simulated annealing. *Science* 220, 4598 (1983), 671–680. https://doi.org/10.1126/science.220.4598.671
[14] Julien Lepagnot, Laurent Moalic, and Dominique Schmitt. 2021. Optimal area polygonization by triangulation and ray-tracing. *The ACM J. of Experimental Alg.* (2021). to appear.
[15] Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. 1953. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics* 21 (1953), 1087–1092. https://doi.org/dx.doi.org/10.1063/1.1699114
[16] Günther Eder, Martin Held, Steinpór Jasonarson, Philipp Mayer, and Peter Palfrader. 2021. 2-Opt moves and flips for area-optimal polygonalizations. *The ACM J. of Experimental Alg.* (2021). to appear.
[17] Joseph O'Rourke. 1998. *Computational Geometry in C* (2nd ed.). Cambridge University Press. https://doi.org/10.1017/CBO9780511804120
[18] Micha Sharir, Adam Sheffer, and Emo Welzl. 2013. Counting plane graphs: Perfect matchings, spanning cycles, and Kasteleyn's technique. *J. Comb. TheorySer. A* 120, 4 (2013), 777–794. https://doi.org/10.1016/j.jcta.2013.01.002
[19] Hans Tangelder and Andreas Fabri. 2019. dD spatial searching. In *CGAL User and Reference Manual* (5.0.2 ed.). CGAL Editorial Board. https://doc.cgal.org/latest/Manual/packages.html#PkgSpatialSearchingD.
[20] The CGAL Project. 2020. *CGAL User and Reference Manual* (5.0.2 ed.). CGAL Editorial Board. https://doc.cgal.org/latest/Manual/index.html.

Received December 2020; revised May 2021; accepted November 2021