Optimal Area Polygonalization

Martín Cepeda
martin.cepeda@polytechnique.edu

Cédric Javault
cedric.javault@polytechnique.edu

***Abstract*—Given a point cloud $\mathcal{S}$ of n 2D vertices, we implement two greedy triangle insertion heuristics in order to approximate the MINimum Area simple Polygonalization and MAXimum Area simple Polygonalization of $\mathcal{S}$ (MINAP and MAXAP respectively), an NP-hard problem. We analyse the geometric sub-problems and present the results of our approach: for example 0.2622 for MINAP and 0.8875 for MAXAP for the euro-night-0000100 instance.**

I. Introduction

The objective of this project is, given a set $\mathcal{S}$ of n points in the plane (with non negative integer coordinates), to compute simple polygonalizations of $\mathcal{S}$. We will refer to this problem as **MINAP** or **MAXAP** depending on whether the polygonalization has the minimal (resp. maximal) area amongst all possible polygonalizations of $\mathcal{S}$. These problems can be viewed as variations of **Euclidean TSP** (simple polygon) with constraints on the area rather than the perimeter of the solution.

MINAP and **MAXAP** are shown to be NP-hard [1], which suggests approximation or heuristic algorithms to solve them rather than brute force optimization, especially for large numbers of points. In the following sections we will present algorithms to check whether a candidate solution is a valid polygonalization, our implementation to solve these problems, and our results.

II. Correct polygonalizations

In order to implement any strategy to solve **MINAP** or **MAXAP**, we need an algorithm to compute a polygon's area and a way to check whether a possible solution is a valid polygonalization with :

- No interior points
- No auto-intersection

A. Area

Given a simple polygon $\mathcal{P}$ of n vertices, we can compute its area by the "shoelace formula" (or surveyor's formula) [2]:

$$A = \frac{1}{2} \cdot \left| \sum_{i=1}^{n} \det \begin{pmatrix} x_i & x_{(i+1 \mod n)} \\ y_i & y_{(i+1 \mod n)} \end{pmatrix} \right| \tag{1}$$

This can be directly implemented in $\mathcal{O}(n)$ time. Small remark : twice the area has to be an integer but the area itself has not (look at the triangle (0,0) (1,0) (0,1)). For task 3, we used the type long to avoid numerical issues when multiplying two big integers... and changed it to double to divide by two.

B. Interior point

Given a simple polygon $\mathcal{P}$ of n vertices and a query point p, a rather clever method to know if p is interior is to ray-trace from p and count the number of intersections with $\mathcal{P}$ (known as the PNPOLY algorithm) [3], as illustrated in the following figure. This check can be computed in $\mathcal{O}(n)$ time.

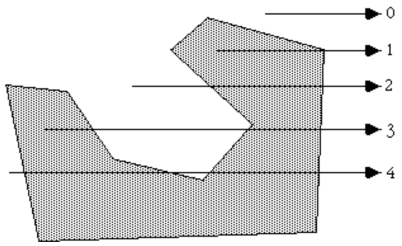

Figure 1: Horizontal ray-tracing for points 0 to 4, odd number of crossings indicate interiorness (source)

C. Intersection of two segments

Let us consider 2 segments s_1 and s_2, each of them defined by 2 points. They intersect if :

- The two extremity points of s_1 lie on different sides of the line that is defined by s_2 or vice-versa
- They are co-linear with a non empty intersection.

This can be computed with the line equation and sign discrimination [4]. An important remark is that in polygons, consecutive segments *always* intersect in the common vertex, so we must allow that. This procedure can be done in $\mathcal{O}(1)$ time for a pair of segments.

D. Polygon auto-intersections

With the above procedure, we can naively check whether a polygon $\mathcal{P}$ of n vertices (thus defined by n segments) auto-intersects in $\mathcal{O}(n^2)$.

However, by considering the lexicographical order of segments we could build a priority queue of events by line sweeping the plane, which allows to detect intersections in a set of n segments in time $\mathcal{O}(n \log n)$. This (optimal) algorithm was introduced by Shamos & Hoey [5].

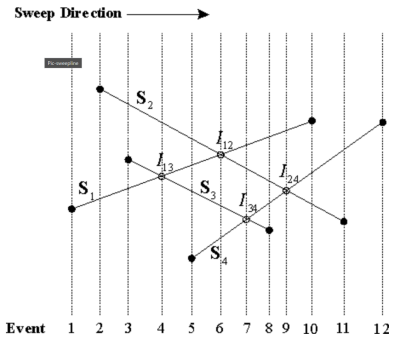

Figure 2: Line sweeping and event ordering (source)

III. Our implementation

A. Many heuristics possible

A possible way of solving **MINAP** and **MAXAP** (as we treat with integer coordinates) is to optimize the area inspired by Pick's Theorem by searching "grid-full" or "grid-empty" polygons [1], nevertheless, random algorithms and greedy approaches have also shown to perform reasonably well in this setup.

Among all strategies documented on the web we can cite random vertex swapping, random rotations, piece-wise concatenation of visible chains, greedy triangle removal starting from Delaunay, neural networks, etc. [6], with more elaborate strategies including optimization techniques such as simulated annealing [7] and divide-and-conquer.

As a baseline procedure, we implemented the following heuristics to approximate optimal areas. These approaches were greatly inspired by [8], [9], [6] and a bit of common sense. In both configurations, we start from a point cloud $\mathcal{S}$ of n points.

B. *MINAP_greedy*

Starting from a randomly chosen point p_0, the polygon $\mathcal{P}_3$ is the triangle formed by p_0 and its 2 nearest neighbors. Then, we iterate. At the i-th iteration, we choose a random point p_i from $\mathcal{S}/\mathcal{P}_{i-1}$ and $\mathcal{P}_i$ is formed when we insert the Point p_i between two **consecutive points** of $\mathcal{P}_{i-1}$. We choose those two points so that:

- the area of the triangle they form with p_i is minimal
- the new polygon $\mathcal{P}_i$ is still valid. Note : In some cases, $\mathcal{P}_{i-1}$ is such that it is impossible to add any new point and get $\mathcal{P}_i$. In that case, our implementation fails and stops (but we can iterate the whole random procedure).

We repeat until we obtain $\mathcal{P}_n$. In lesser words, it's a randomly and greedy vertex insertion algorithm.

C. *MAXAP_greedy*

We first compute the convex hull $\mathcal{P}_0$ of $\mathcal{S}$. At each iteration i, we insert the point p_i of $\mathcal{S}/\mathcal{P}_{i-1}$ that forms the triangle with minimal area with two consecutive vertices of $\mathcal{P}_{i-1}$ to get $\mathcal{P}_i$.

We repeat this $n - n_h$ times, where n_h is the size of the convex hull of $\mathcal{S}$. In lesser words, this time it's a greedy removal of triangles starting from the convex hull (the one with maximum possible area). It's worth to mention that this algorithm is deterministic, at each iteration we find the smallest removable triangle.

D. Complexity and numerical issues

For both procedures, the runtime is in $\mathcal{O}(n^3)$ with naive auto-intersection checks (our implementation), but it can be reduced to $\mathcal{O}(n^2 \log n)$ by implementing Shamos-Hoey intersection detection.

For instances with a small number of points, we iterated many times **MINAP_greedy**, and kept the best solution (there is no use to do that for **MAXAP_greedy** which is a deterministic algorithm.

For large number of points and more precisely for instance where the coordinates are large integers (like world-0010000), we had an overflow issue in the segment intersection routine using the JAVA long type. Long uses 8 bytes and is therefore limited to $2^{63} = 9,22.10^{18}$ and we were manipulating numbers bigger than 10^{20} to check where the points were with respect to the line defined by the other segment. We had to change to the type double to solve the problem. This could result in a lack of numerical stability in some cases but did not experience it...

IV. Results & discussion

The code, extended from the Java-JCG source provided can be found here. We modified all classes, so it's *strongly* suggested to replace all initial source files.

In terms of performance, we obtained the following scores selecting the best from 100 runs of **MINAP_greedy** (due to its random initialization) for instances with 1000 points or less, and directly computing **MAXAP_greedy**.

Instance	MIN score	MAX score
uniform-0000010	0.3517	0.8448
uniform-0000100	0.3244	0.8382
uniform-0001000	0.2628	0.7798
uniform-0010000	–	–
uniform-0100000	–	–
euro-night-0000100	0.2622	0.8875
paris-0001000	0.3665	0.8320
world-0010000	–	–

Table I: Score results, for uniform-0100000 the runtime was excessive for our naive-greedy approach. The instances with 10 000 points are feasible but need more than 24 hours and we did not compute them.

In the following table, the run time for MIN is the one that gave the computation of the best area (and not the sum of all the trials). It is in seconds on a pretty slow laptop. For the Uniform serie, we can check that multiplying by 10 the number of points multiplies the time computation by something in the order of 10^3 (660 for MAX between 100 and 1000 points), which is as expected.

Instance	MIN runtime	MAX runtime
uniform-0000010	2.12e-4	3.93e-4
uniform-0000100	5.01e-2	3.05e-2
uniform-0001000	24.13	20.13
euro-night-0000100	4.86e-2	3.00e-2
paris-0001000	10.13	8.377

Table II: Runtimes for each instance (in sec)

Finally, here are a few drawings of the poligonizations we computed :

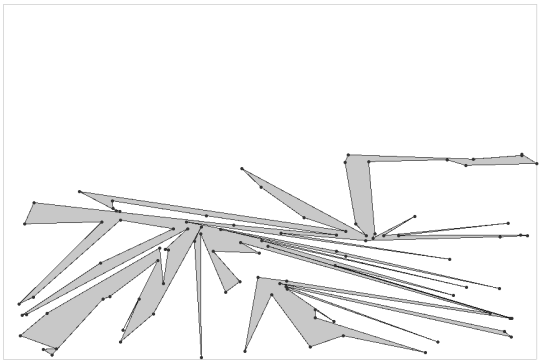

Figure 3: MIN Result pour Euro-100

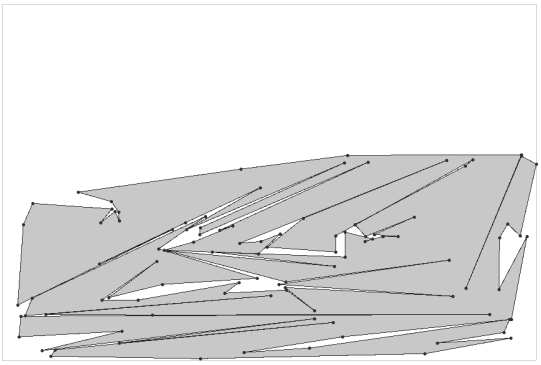

Figure 4: MAX Result pour Euro-100

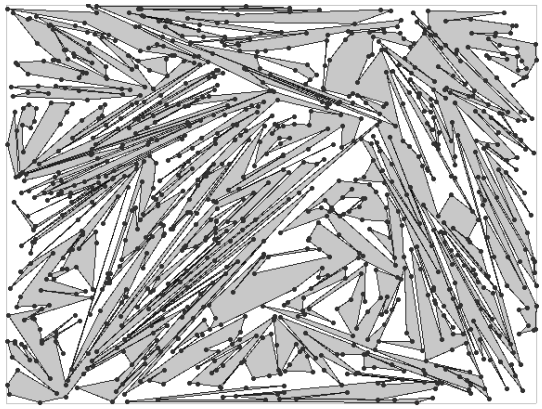

Figure 5: MIN Result pour Uniform-1000

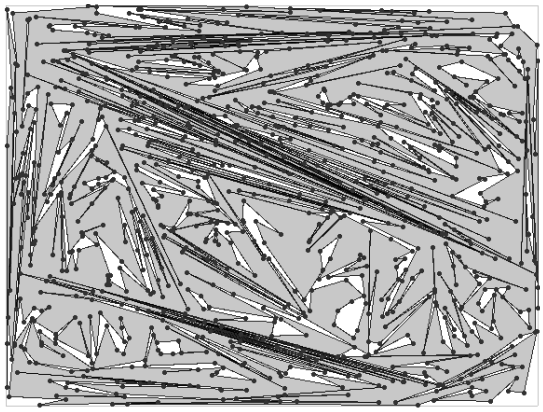

Figure 6: MAX Result pour Uniform-1000

V. Conclusion & further work

We managed to implement two algorithms for **MINAP** and for **MAXAP**. They are working and they seem to give nice results but the **MINAP** works randomly and can fail (in some cases, even with many iterations, we did not even get a solution) and both algorithms are in $\mathcal{O}(n^3)$ which make them really slow with n large.

In order to complete our work, we could :

- Change **MINAP_greedy** when it fails to add Point p_i: instead of restarting the entire procedure from scratch, we could go backward and choose another point for p_{i-1}. We discovered the issue at the very end when computing the results on large instances and did not have enough time to implement that change.
- Do a pre-ordering of the points in $\mathcal{S}$ by, for example, their distance with respect to the point cloud centroid (in $\mathcal{O}(n \log n)$). This could improve notably the search time for the minimum area triangle in **MAXAP_greedy** : as we expect the find the best point much quicker than by following the x-coordinate lexicographic order in which the points are indexed initially, we could decide to only test a fraction of the points).
- For **MINAP_greedy**, as mentioned in III-D, the bottle-neck lies in the auto-intersection check, which can be improved via a priority queue *à la* Shamos-Hoey.
- Change the greedy heuristic. As we see in 6, we have *long edges* that difficult further polygon constructions. This is tackled in [6] by including a penalization on the triangle perimeter in order to avoid extremely flat artifacts.
- Implement local search to build small polygonalizations and then merge them when $\mathcal{P}_i$ grows.

References

[1] S. P. Fekete, "On simple polygonalizations with optimal area," *Discrete & Computational Geometry*, vol. 23, no. 1, pp. 73–110, 2000. [Online]. Available: https://doi.org/10.1007/pl00009492

[2] B. Braden, "The surveyor's area formula," *The College Mathematics Journal*, vol. 17, no. 4, pp. 326–337, 1986. [Online]. Available: https://www.tandfonline.com/doi/abs/10.1080/07468342.1986.11972974

[3] W. R. Franklin, "PNPOLY - Point Inclusion in Polygon Test," last visited on 04/04/2020. [Online]. Available: https://wrf.ecse.rpi.edu/Research/Short_Notes/pnpoly.html

[4] J. Kogler, R. Gorkovenko, and M. Mykhailova, "Search for a pair of intersecting segments - Competitive Programming Algorithms." [Online]. Available: https://cp-algorithms.com/geometry/intersecting_segments.html

[5] M. I. Shamos and D. Hoey, "Geometric intersection problems," in *17th Annual Symposium on Foundations of Computer Science (sfcs 1976)*. IEEE, 1976. [Online]. Available: https://doi.org/10.1109/sfcs.1976.16

[6] E. D. Demaine, S. P. Fekete, J. S. Mitchell, and D. Krupke, "CG Week 2019 workshop presentations," last visited on 04/04/2020. [Online]. Available: https://sites.google.com/stonybrook.edu/cgweek2019-workshop/

[7] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983. [Online]. Available: https://science.sciencemag.org/content/220/4598/671

[8] M. T. Taranilla, E. O. Gagliardi, and G. H. Peñalver, "Approaching minimum area polygonization," in *XVII Congreso Argentino de Ciencias de la Computación*. Argentina: Universidad Nacional de La Plata, 2011, pp. 161–170. [Online]. Available: http://oa.upm.es/19287/

[9] J. Peethambaran, A. D. Parakkat, and R. Muthuganapathy, "An empirical study on randomized optimal area polygonization of planar point sets," *J. Exp. Algorithmics*, vol. 21, 2016. [Online]. Available: https://doi.org/10.1145/2896849