COMP 651: Geometric Optimization Challenge 2019

Max Hudnell

April 2019

1 Problem

Given a set S of n points in the plane find a simple polygon whose vertex set is S that minimizes (or maximizes) the enclosed area. (An NP-hard problem)

2 Method of Solution

2.1 Constructing triangulations

Our approach aims to construct a minimum area polygon by piecing together triangles whose vertices are in S. Every simple polygon can be composed as a set of triangles, a triangulation, so we aim to build this triangulation from the ground up in order to form a polygon whose vertex set is S with minimum area.

Greedy approach. We explore multiple methods for building this triangulation. The first method is a simple greedy approach. It starts by finding the minimum area triangle constructed from vertices in S. This can be done in $\mathrm{O}(n^2)$ time via arrangements. Then, for each edge of the triangle (with points p_1 and p_2), find the point $p_3 \in S$ for which the triangle $\triangle p_1p_2p_3$ has minimum area (this implies $\triangle p_1p_2p_3$ contains no other vertices of S) and is valid. This triangle being valid means that it does not intersect any edge of our current polygon. Find the smallest possible triangle for each edge in this manner and store them as candidate triangles. Then add the smallest of these candidate triangles to the triangulation. Repeat this process (computing the smallest valid triangles for all edges of your current polygon). The algorithm terminates after $(n-2)$ triangles have been added to the triangulation.

Substituting 'smallest' for 'largest' in the above paragraph will yield a possible greedy approach for constructing a reasonable triangulation for the maximum-area polygon, with the only necessary addition being that one must check that the candidate triangles are empty. However we did not test this approach and thus cannot speak to its results.

Solution space search. The second method we propose aims to improve upon our greedy approach explained above, and provide a minimum-area polygon closer to the optimal solution. To do this, we perform a search of the solution instance space.

In constructing our triangulation, we can imagine a tree in which the root is our initial smallest triangle, let's call x, and its child is the triangulation consisting of x and the candidate triangle which got added on to it. Therefor height in this tree corresponds to the number of triangles in our triangulation. A polygon whose vertices consist of every point in S is reached once the height of the tree is $n-2$.

If we no longer restrict ourselves to adding the smallest triangles to our triangulation, we can imagine a tree in which every possible triangulation of a polygon with vertices in S corresponds to a path from root to leaf in the tree. Thus is we perform an exhaustive search of the tree, we will find an optimal minimal-area polygon. However this being an NP-hard problem, this tree has $O(n^n)$ leaves. We aim to utilize a branch and bound method to prune paths of this tree during our search.

Restrictions	euro_night_0000015 area	euro_night_0000015 run-time	us_night_0000020 area	us_night_0000020 run-time
A	7,107,334.0	2:55.48	?	?
A + B	7,107,334.0	0:38.96	7,040,960.0	64:54.47
A + C	7,661,692.0	0:10.09	6,799,308.0	4:02.73
A + B + C	8,752,112.0	0:01.56	8,544,290.0	1:20.45

Table 1: Run-time format is (minutes : seconds). Experiments which took too long and didn't finish are denoted by '?'. (A) Restrict intermediate triangles. (B) Restrict initial triangles. (C) Bound estimation.

3 What was attempted

3.1 Exhaustive search of solutions

We choose to traverse the solution space tree by using a priority queue, which allows us to visit the space in a depth-first manner. The priority queue prioritizes first on the height of a node in the tree. Remember, height corresponds to the number of triangles in our triangulation, so the algorithm intends to build a maximal triangulation as soon as possible. In the case of a tie in regards to heights of nodes, the priority queue then prioritizes on the minimum area the nodes. Together, this means the algorithm begins by performing the greedy approach described above, in which minimum area triangles are added on to our triangulation.

3.2 Branch and bound

To begin, we applied minimum criteria for pruning instances which have no chance of removing the optimal solution from the tree. This criteria is as follows:

- **Redundant polys.** If a triangulation instance is visited which corresponds to the same polynomial as a previously visited triangulation, it can safely be pruned. This was implemented using a hash table which stores intermediate polygons.
- **Exceeds greedy bound.** If an instance corresponds to a polygon which has area greater than our initial greedy solution, then it can safely be proved since area cannot be decreased by descending the tree. The bound which our greedy solution provides is updated if a better solution is found.

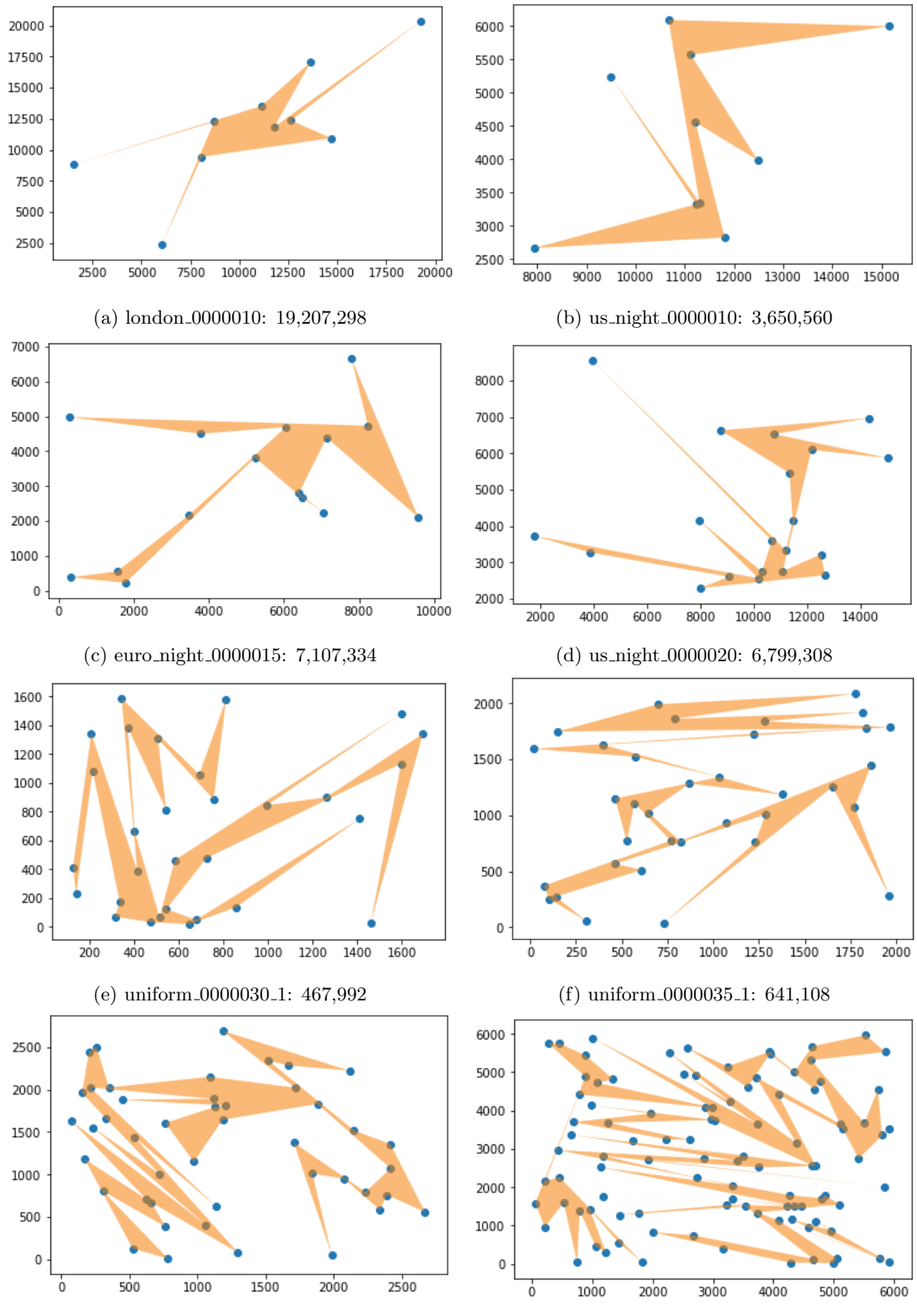
The following changes were tested and found to improve run-time while still leading to a decent minimal-area polygon:

- **Restrict intermediate triangles.** We restrict the intermediate triangles (referred to as $\triangle p_1 p_2 p_3$ in Section 2) to the smallest valid triangle, rather than all valid triangles.
- **Restrict initial triangles.** We restrict the initial triangles to the smallest valid triangle for each point in S, rather than all valid triangles.
- **Bound estimation.** We prune instances based on polygon area like before, but now we predict what the minimum area of our final polygon will be via, based on the polygon (p) for the current instance and the most recently added triangle (t) to the instance:

$$estimated\ area = area(p) + area(t) * (len(S) - len(p)) \tag{1}$$

  Instances are pruned if their estimated area is greater than our bound.

The above changes follow the simple heuristic that a typical minimum-area polygon will consist of small triangles, so those are the triangles we aim to consider when constructing triangulations. Table 1 shows the effects the discussed changes have on run-time and the generated minimum-area polygon. Examples of generated polygons are shown in Figure 1.



(a) london_0000010: 19,207,298

(b) us_night_0000010: 3,650,560

(c) euro_night_0000015: 7,107,334

(d) us_night_0000020: 6,799,308

(e) uniform_0000030_1: 467,992

(f) uniform_0000035_1: 641,108

(g) uniform_0000045_1: 1,509,976

(h) uniform_0000100_1: 8,466,926

Figure 1: Examples of minimum-area polygons and corresponding areas. For $n > 35$ early stopping was used, the full solution tree was not explored / pruned.