University of Athens
Dept. of Informatics & Telecommunications
K17c: Software Development for Algorithmic Problems

Kyriakopoulos Dimitrios[*] – `sdi1900093`
Poulis Angelos[*] – `sdi1900230`
Rontogiannis Dimitrios[*] – `sdi1900165`

Fall 2022 - Project 2 (Optimal Area Polygonization)

Contents

1 Introduction

We implemented three algorithms (Local Search/Simulated Annealing/Ant Colony) for optimal area point set polygonization. We provide the aforementioned implementations along with scripts for testing the algorithms and a Python script to visualize the results. Our code was written in C++, using the CGAL library. At the end of this report, we present a comparison of the algorithms on a sample test case.

[*]All authors have contributed equally.

2 What is included

```
Optimal-Area-Polygonization
├─ build - files created by CMake
├─ CMakeLists.txt - CMake for build
└─ include - Header files
   ├─ optimal_polygonization - header files for optimal polygonization
   │  ├─ ant_colony_algo.hpp - ACO algorithm
   │  ├─ global_transition.hpp - Global Transition for SA algorithm
   │  ├─ local_transition.hpp - Local Transition for SA algorithm
   │  ├─ subdivision.hpp - Subdivision to subproblems for SA algorithm
   │  ├─ local_search_algo.hpp - Local Search optimization algorithm
   │  └─ simulated_annealing_algo.hpp - SA optimization algorithm
   ├─ simple_polygonization - header files for simple polygonization
   │  ├─ ch2polyline_algo.hpp - Definition of based on CH algorithm
   │  ├─ incremental_algo.hpp - Definition of Incremental algorithm
   │  └─ onion_algo.hpp - Definition of Onion algorithm
   ├─ common.hpp - Definition of some common used functions
   ├─ graham_scan.hpp - Definition of Graham Scan
   ├─ pick_algo.hpp - Definition of Pick's theorem
   └─ utils.hpp - Definition of util functions
```

- src - Source files
  - optimal_polygonization - source files for optimal polygonization
    - local_search_algo.cpp - Local Search implementation
    - simulated_annealing_algo.cpp - Simulated Annealing implementation
    - local_transition.cpp - Local transition for Local Search
    - global_transition.cpp - Global transition for Local Search
    - subdivision.cpp - Simulated Annealing with subdivision
    - ant_colony_algo.cpp - ACO implementation
  - simple_polygonization - source files for simple polygonization
    - ch2polyline_algo.cpp - Based on CH algorithm implementation
    - incremental_algo.cpp - Incremental algorithm implementation
    - onion_algo.cpp - Onion algorithm implementation
  - graham_scan.cpp - Custom Graham Scan algorithm
  - main.cpp - Main function implementation
  - pick_algo.cpp - Implementation of Pick's theorem
  - utils.cpp - Implementation of util functions
- scripts - Helper scripts
  - plot_polygon.py - Python script for plotting a polygon line
  - plot_graphs.py - Python script for plotting benchmarks
  - run_test_cases.sh - Bash script for running test-cases

3 Usage

3.1 Build and run

CMake file it is provided for compilation of the project. Build and run the project as follows. Go to the /build directory and run:

```
$ cmake .. && make
$ ./bin/optimal_polygon -i <input_file> -o <output_file>
-algorithm <local_search/simulated_annealing/ant_colony>
-init_algo <onion/ch2poly | except ant colony>
-L <L parameter according to algorithm>
-max/min
-threshold <double | only in local search>
-annealing <local/global/subdivision | in simulated annealing>
-alpha <double | only in ant colony>
-beta <double | only in ant colony>
-rho <double | only in ant colony>
-elitism <0/1 | only in ant colony>
```

Parameters description:

- <input_file> : File with the point-set to polygonize.
- <output_file> : File to print the results.
- <algorithm> : Optimal area polygonization algorithm to run:
  - "local_search" for Local Search
  - "simulated_annealing" for Simulated Annealing
  - "ant_colony" for Ant Colony
- <init_algo> : Algorithm to produce the initial simple polygon line for LS and SA:
  - "onion" for Onion algorithm
  - "ch2poly" for Based on Convex Hull algorithm
- <L> : L parameter according to algorithm:
  - number of iterations for Local Search
  - number of iterations for Simulated Annealing
  - Number of cycles to perform for ACO $\in [10, 50]$
- <min> : Minimal area polygonization.
- <max> : Maximal area polygonization.

- <threshold> : Threshold in local search.
- <annealing> : Annealing parameter for simulated annealing
  - "local" for local transition
  - "global" for global transition
  - "subdivision" for subdivision to subproblems
- <alpha> : $\alpha \simeq 1$ parameter for ACO
- <beta> : $\beta \in [1, 5]$ parameter for ACO
- <rho> : $\rho \simeq 1$ parameter for ACO
- <elitism> : elitism $\in \{0, 1\}$ parameter for ACO

3.2 Testing scripts

We also provide some scripts for testing. Run them as follows from the scripts/ directory:

```
$ ./run_test_cases -i <test cases folder>
    -algorithm <local_search/simulated_annealing/ant_colony>
    -init_algo <onion/ch2poly | except ant colony>
    -L <L parameter according to algorithm>
    -max/min
    -threshold <double | only in local search>
    -annealing <local/global/subdivision | in simulated annealing>
    -alpha <double | only in ant colony>
    -beta <double | only in ant colony>
    -rho <double | only in ant colony>
    -elitism <0/1 | only in ant colony>
```

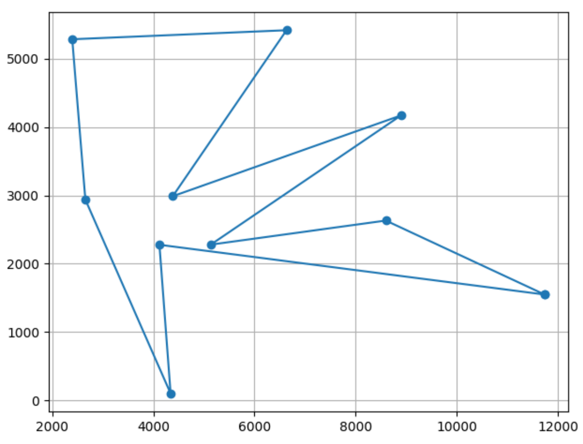
- run_test_cases.sh - *Runs a given algorithm with its options for all test cases in the specified folder.*

In addition we provide a Python script that plots the produced polygon line. Run it as follows (from scripts/):

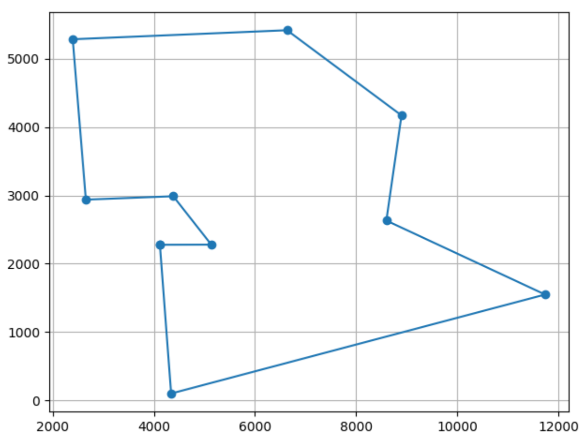
```
$ python3 plot_polygon.py -i polygon_file
```

4 Experiments

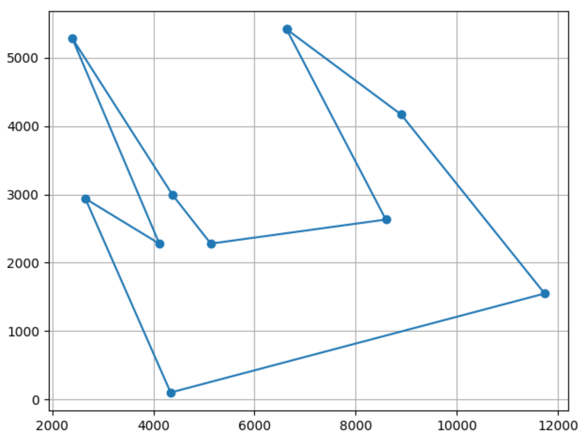
We present the optimal area polygonizations of "instances/data/images/euro-night-0000010.instance" using all three algorithms for minimal and maximal area optimization. We ran Local Search with $L = 10$ and $threshold = 10^{-5}$, Simulated Annealing with annealing = global and $L = 10^4$, and Ant Colony with $L = 50$, $\alpha = 1.0$, $\beta = 3.0$, $\rho = 1.5$ & $elitism = 1$.



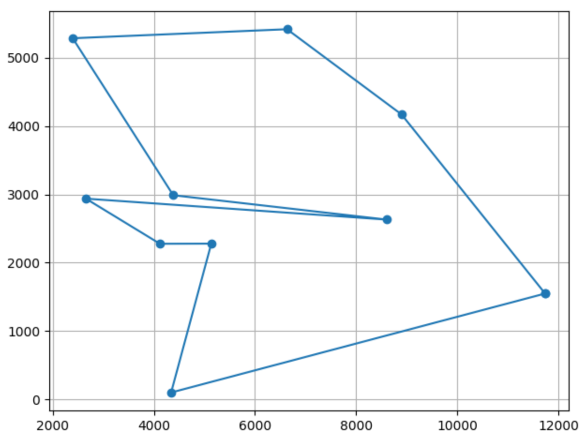
(a) Local Search - min (Ratio: 0.45)



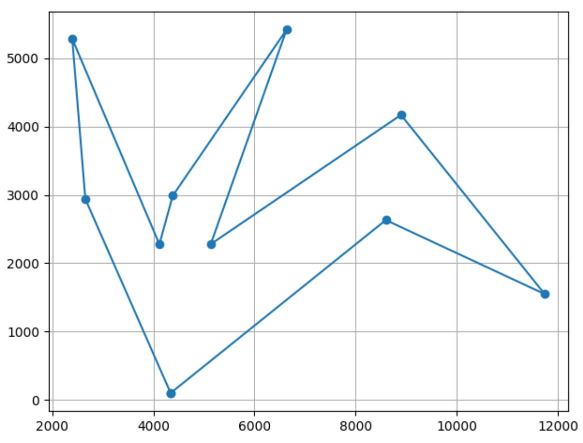
(b) Local Search - max (Ratio: 0.84)



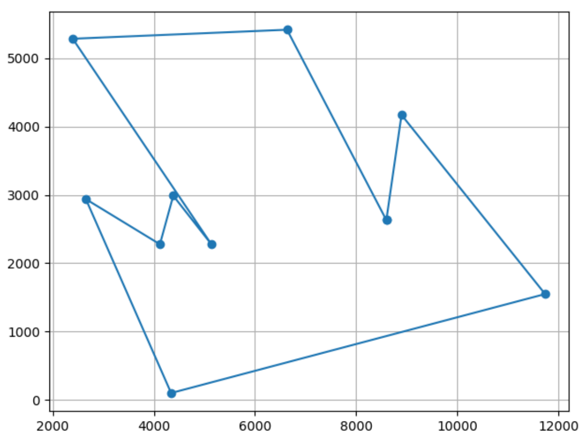
(c) Simulated Annealing - min (Ratio: 0.57)



(d) Simulated Annealing - max (Ratio: 0.84)



(e) Ant Colony - min (Ratio: 0.47)



(f) Ant Colony - max (Ratio: 0.85)

Figure 1: Min/Max area polygonizations of "instances/data/images/euro-night-0000010.instance"

5 Remarks & Conclusion

We compared the different algorithms on the instances that were given with number of points less or equal than 20, in order to include ACO in our comparisons and not exceed the memory limit. We performed further testing of these algorithms using a custom dataset generation that generates 10-point point-sets in $\mathcal{N}(0, 10) \times \mathcal{N}(0, 10)$.

In general we observed the following:

- The Ant Colony Optimization algorithm requires too much memory but produces the best results. Though, we got it to run for $\simeq$ 20 points before we ran out of memory.
- Simulated Annealing algorithm is the fastest among the three but without much difference from the initial's polygon line area.
- Local Search produces better results than Simulated Annealing and is close to ACO's results, but runs too slow for point-sets with more than 100 points.

As a general conclusion we can say that, in order to approach the global optimum, the run-time increases rapidly.