Computing Low-Cost Convex Partitions for Planar Point Sets Based on a Memetic Approach

Laurent Moalic
Université de Haute-Alsace, IRIMAS UR 7499, F-68100 Mulhouse, France
Université de Strasbourg, France
laurent.moalic@uha.fr

Dominique Schmitt
Université de Haute-Alsace, IRIMAS UR 7499, F-68100 Mulhouse, France
Université de Strasbourg, France
dominique.schmitt@uha.fr

Julien Lepagnot
Université de Haute-Alsace, IRIMAS UR 7499, F-68100 Mulhouse, France
Université de Strasbourg, France
julien.lepagnot@uha.fr

Julien Kritter
Université de Haute-Alsace, IRIMAS UR 7499, F-68100 Mulhouse, France
Université de Strasbourg, France
julien.kritter@uha.fr

Abstract

We present a memetic approach designed to tackle the 2020 Computational Geometry Challenge on the Minimum Convex Partition problem. It is based on a simple local search algorithm hybridized with a genetic approach. The population is brought down to its smallest possible size – only 2 individuals – for a very simple implementation. This algorithm was applied to all the instances, without any specific parameterization or adaptation.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases metaheuristics, memetic algorithms, convex partition optimization

Digital Object Identifier 10.4230/LIPIcs.SoCG.2020.84

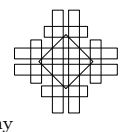
Category CG Challenge

Related Version A description of the 2020 CG:SHOP Challenge with related work and overall outcomes can be found at [2] `https://arxiv.org/abs/2004.04207`.

1 Introduction

Given a set P of points in the plane, the Minimum Convex Partition problem is that of identifying a partition of the convex hull of P into the smallest number of convex polygons whose vertices are the points of P. Finding the minimum convex partition of given instances of points with integer coordinates was the aim of the 2020 CG:SHOP Challenge [2].

It has recently be shown that the Minimum Convex Partition problem is NP-hard, when the point sets are not necessarily in general position [3]. Thus, simple local search algorithms, which are prone to be trapped in local optima, are not efficient enough and do not yield the best solutions for several instances of this problem. For these reasons, we propose to use a memetic approach, an effective class of metaheuristics commonly used to solve various combinatorial problems [5].



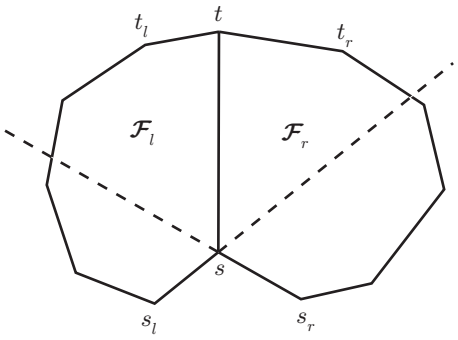


Figure 1 The edge st is rotatable at s.

2 Algorithmic Methods

2.1 A memetic approach

The main idea is to hybridize two mechanisms, a local search which intensifies the search (exploitation phase, which improves a solution by converging to a nearby local optimum), and a crossover which diversifies it (exploration phase, to escape from local optima and explore new areas of the search space). The goal is to cover all parts of the search space as well as possible and find the best local solution in each part. The idea of using only two individuals in the population was first successfully introduced in [4], for the graph coloring problem. It has the advantage of removing the selection phase, as well as the replacement strategy.

The memetic scheme starts here with two identical individuals which are the Delaunay triangulation of the point-set. In order to avoid wasting time with reparations, the individuals are kept legal, that is, convex partitions of the point-set. The fitness value of a solution is given by the number of its polygons, which we aim at minimizing.

2.2 A simple descent local search

Let P be a set of n points in the plane and let $\mathcal{P}$ be a convex partition of P, i.e., a partition of the convex hull of P into convex polygons whose vertices are the points of P.

The aim of the descent local search is to remove internal edges of $\mathcal{P}$, i.e. edges that do not belong to the boundary of the convex hull of P. Thereby, the two faces of $\mathcal{P}$ on each side of the removed edge are merged to become a single face. The descent local search never degrades the current convex partition.

Let st be an internal edge of $\mathcal{P}$ and let $\mathcal{F}_l$ and $\mathcal{F}_r$ be the faces of $\mathcal{P}$ that share the edge st, and that are respectively on the left side and on the right side of st (see Figure 1). Let s_l (resp., t_l) be the vertex of $\mathcal{F}_l$ that precedes s (resp., succeeds t) on the boundary of $\mathcal{F}_l$ in counterclockwise direction. Let s_r (resp., t_r) be the vertex of $\mathcal{F}_r$ that succeeds s (resp., precedes t) on the boundary of $\mathcal{F}_r$ in counterclockwise direction. Consider the two following conditions:

1. The point s_r is on the left side of or on the oriented straight line $(s_l s)$.
2. The point t_l is on the left side of or on the oriented straight line $(t_r t)$.

The edge st is said to be *immovable*, *rotatable*, or *removable* respectively, when 0, 1, or 2 of the conditions 1 and 2 are satisfied. When st is rotatable, we say that st is rotatable at t if condition 1 is satisfied, and we say that st is rotatable at s if condition 2 is satisfied.

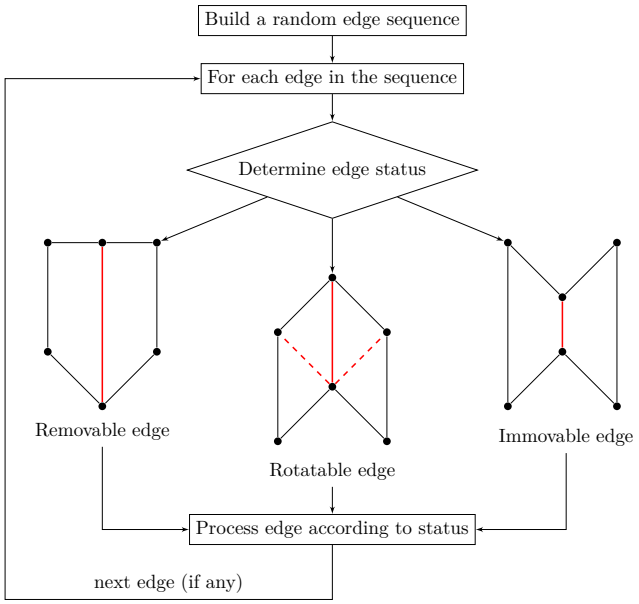


Figure 2 One step of the local search: a remove, move, unmove approach.

Let $\mathcal{F}$ be the face obtained by removing st and by merging $\mathcal{F}_l$ and $\mathcal{F}_r$. If the edge st is removable, $\mathcal{F}$ is obviously convex. If st is immovable, $\mathcal{F}$ is not convex, neither in s, nor in t. The only way to cut $\mathcal{F}$ into two convex faces is to put back the edge st.

If st is rotatable, suppose, without loss of generality, that it is rotatable at s. In this case, $\mathcal{F}$ is not convex in s but is convex in all other vertices. To cut $\mathcal{F}$ into two convex faces, we have to add an edge st' that connects s to a vertex t' of $\mathcal{F}$ other than s, s_l, and s_r. Whatever the choice of t', the two new faces are convex in all their vertices, except possibly in s. To ensure the convexity in s, t' must be chosen altogether on the left side of or on (ss_r), and on the left side of or on $(s_l s)$. The vertices t' that satisfy these conditions are called the *valid positions* for t (relatively to st in $\mathcal{P}$). The valid positions for t form a connected polyline on the boundary of $\mathcal{F}$. This polyline contains necessarily t and may be reduced to t.

The descent local search uses the above properties to improve the current convex partition $\mathcal{P}$. It proceeds by steps. In each step, a random sequence of internal edges of $\mathcal{P}$ is generated (see Figure 2). For every edge st in the sequence:

- if st is removable, it is removed from $\mathcal{P}$,
- if st is rotatable at s, the set of valid positions for t is generated, a point r in the set is randomly chosen, and st is replaced by sr (the processing is symetric if st is rotatable at t).

Clearly, every removable edge produces an improvement of the current convex partition. Rotatable edges give rise neither to improvements nor to degradations, but by moving a rotatable edge, another edge may become removable and may therefore be removed later on in the process.

This process is repeated until no more improvement seems to be possible, i.e. until the current convex partition is trapped in a local optimum. For all instances of the competition, the number of steps was fixed to 10,000.

Let us now consider the complexity of one step. The generation of a random sequence of edges, as well as the processing of all removable and immovable edges are linear in the number of edges. Thus, the complexity will be determined by the processing of the rotatable

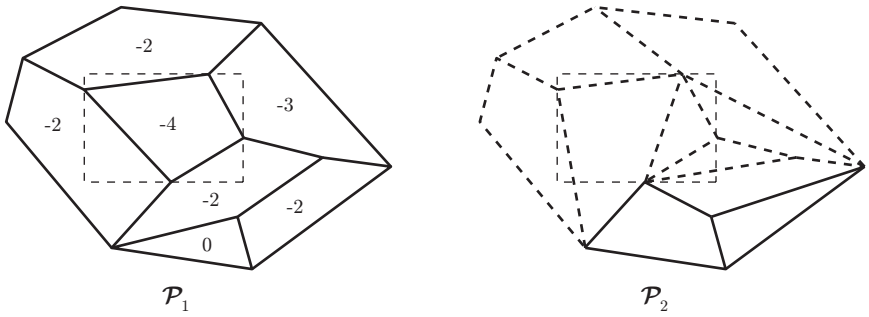


Figure 3 The numbers in the faces of $\mathcal{P}_1$ are the scores of theses faces with respect to $\mathcal{P}_2$. If the best face of $\mathcal{P}_1$ is transmitted to a child, then the faces of $\mathcal{P}_2$ with full edges are the only ones that can still be transmitted to that child.

edges. For each edge st rotatable at, say, s, all vertices of the two faces on both sides of st may be valid positions for t, except s and its two neighbors s_l and s_r. This leads to a complexity of $O(n)$ per edge, and thus to an overall complexity of $O(n^2)$ for one step of the algorithm.

In practice, this complexity is much lower. Consider, for example, the instances of the competition with 100,000 points. The largest set of valid positions encountered over 10 runs for each of these instances contained only 16 points. The average size of the set was about 2.64 points. The sets are larger for instances with large numbers of collinear points. For the instance "rop" with 64054 points, a set of 305 valid positions was found. The average size was 7.56 points.

2.3 A crossover which gets the best of the parents

Diversification starts with two given convex partitions $\mathcal{P}_1$ and $\mathcal{P}_2$ of P - the parents - generated by the descent local search. The crossover aims at getting part of each parent's "gene pool" to produce two new convex partitions of P - the children. To generate a child, the idea is to get some non-overlapping faces from each parent. The child is then, at first, a partial solution made of convex polygons and isolated points.

A "good" child is typically one which gathers "good" faces from its parents. Clearly, the optimal convex partition of P minimizes the sum of its vertices' degrees. We therefore compute a score for every face of each partition, which measures the attractiveness of the face relatively to the other partition. The score of a face of, say, $\mathcal{P}_1$ is obtained by summing up the degrees of its vertices and by subtracting the degrees of these same vertices in $\mathcal{P}_2$ (see Figure 3). Roughly speaking, if the score is negative, the environment of the face is better in $\mathcal{P}_1$ than it is in $\mathcal{P}_2$. The best face is the one with lowest score.

The crossover algorithm first sorts the faces of $\mathcal{P}_1$ and $\mathcal{P}_2$ independently by increasing scores. Then, it transmits alternatively one face from $\mathcal{P}_1$ and one face from $\mathcal{P}_2$ to one child, in order. Ties are broken randomly. This stochasticity helps to generate two different children. It is enhanced by the fact that the first face transmitted to the first child comes from $\mathcal{P}_1$, while the first face transmitted to the second child comes from $\mathcal{P}_2$.

So that the faces transmitted to a same child do not intersect, intersection tests between the faces in $\mathcal{P}_1$ and $\mathcal{P}_2$ have to be performed. To accelerate the intersection test when a face in, say, $\mathcal{P}_1$ is transmitted to a child, the axis-parallel bounding box of the face is computed. All faces in $\mathcal{P}_2$ which intersect the box are disabled, so that they cannot be transmitted to

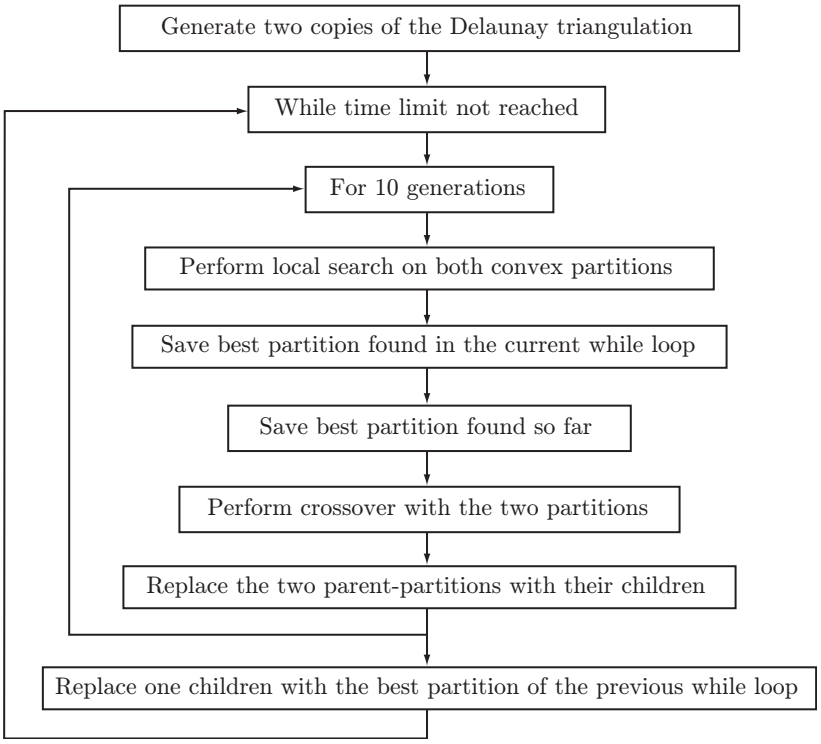


Figure 4 General scheme of the memetic approach.

that child. When no more faces in $\mathcal{P}_1$ and no more faces in $\mathcal{P}_2$ can be transmitted to a child, a constrained triangulation of P and of the set of transmitted faces is computed. The two child-partitions constructed that way replace their parents and are used in the next iteration of the algorithm (see Figure 4).

The algorithm alternates between the local search and the crossover phases until a time limit is reached. As the local search is a simple descent, the best encountered solution is recorded just before each crossover phase.

3 Practical Computation

3.1 Computational Environment

The program was written in C++ using data structures and functions from the CGAL library [1]. Several experiments were carried out on the Strasbourg high-performance computing cluster (HPC) using identical machines equipped with 2.6GHz Intel Xeon Gold 6126 CPUs. Some statistics on the results obtained by 10 executions of our algorithm are presented and discussed. To obtain these statistics, each execution was stopped after one hour. Note that the results submitted for the challenge were obtained without any information on running time. However, it turns out that, for 80% of the instances, the algorithm achieved the value submitted to the challenge in less than one hour.

3.2 Algorithm Engineering

It seems interesting to note that one of the strengths of the proposed approach is that there are no instance-specific settings. The same program is applied to all instances, regardless of their size or structure.

The only parameters are the local search duration between two crossovers, and the number of generations in one cycle. We have set the local search duration to 10,000 for all instances. That is to say that for each generation, each edge can move or be removed 10,000 times. The size of a cycle is set to 10 for all instances. That is, after 10 generations the best solution of the previous cycle is reintroduced. These values have been determined experimentally, and can be improved for a better behavior of the algorithm.

3.3 Experimental Results

For each instance of the problem, let p_{impr} be the improvement between an intermediary solution and a final solution (in percent). It is computed using the best fitness among the ones found by the first descent local search over the 10 runs (f_{first}), and the best fitness at the end of the one-hour execution over the 10 runs (f_{end}). It is given by $\frac{100\ (f_{first}-f_{end})}{f_{first}}$. For each class of instances, the evolution of this value is presented in Figure 5, over the number of points in the instance on which the algorithm is applied.

A similar percentage of improvement, denoted by p_{comp}, is computed between f_{end} and the fitness of the solution submitted for the competition.

One can see in Figure 5 that three different behaviors are adopted by the algorithm, depending on the class of instances on which it is applied:

1. On the class of "rop" instances, the values of p_{impr} are globally significantly greater than for any other class of instances. Hence, the exploration phase using a genetic crossover appears to be very useful for the "rop" instances. Among these instances, the one having the highest value of p_{comp} is "rop0010050", for which $p_{comp} = 25.74$.
2. On the "ortho_rect_union" instances, the opposite behavior is displayed by the algorithm, i.e. the values of p_{impr} are globally significantly lower than for the other classes. Let std_{first} be the standard deviation of the solutions found by the first descent for the 10 executions of our approach on a given instance. Among the "ortho_rect_union" instances, the one having the highest value of std_{first} is "ortho_rect_union_47381", for which $std_{first} = 4.93$. It is the lowest value of std_{first} compared to 50,000 and even 40,000 point instances of all other classes. Furthermore, among these instances, the one having the highest value of p_{comp} is "ortho_rect_union_7663", for which $p_{comp} = 0.10$. It could mean that a simple local search is sufficient to find good solutions for these instances.
3. On the other classes of instances except "mona-lisa", our approach appears to behave similarly. The values of p_{impr} are globally significantly greater than the ones of "ortho_rect_union" instances, but significantly lower than the ones of "rop" instances. Among the instances of all classes except the "ortho_rect_union", the "rop" and the "mona-lisa" ones, the one having the highest value of p_{comp} is "uniform-0090000-1", for which $p_{comp} = 0.13$.

For "mona-lisa", containing only one instance of 1,000,000 points, it is not possible to observe how p_{impr} would evolve over the number of points. However, p_{comp} equals 0.65, which is low compared to most "rop" instances. One can also notice significant fitness differences between instances belonging to the same class, which could be due either to a stability issue of the approach or to the nature of the instances.

The analyses presented in Figures 6, 7, and 8 are based on one of the 10 runs that leads to the median performance. It shows the evolution of the number of faces of the best solution found so far over the generations. These 3 figures correspond to a one hour run.

One can see that our approach is able to converge to a good solution in few generations.

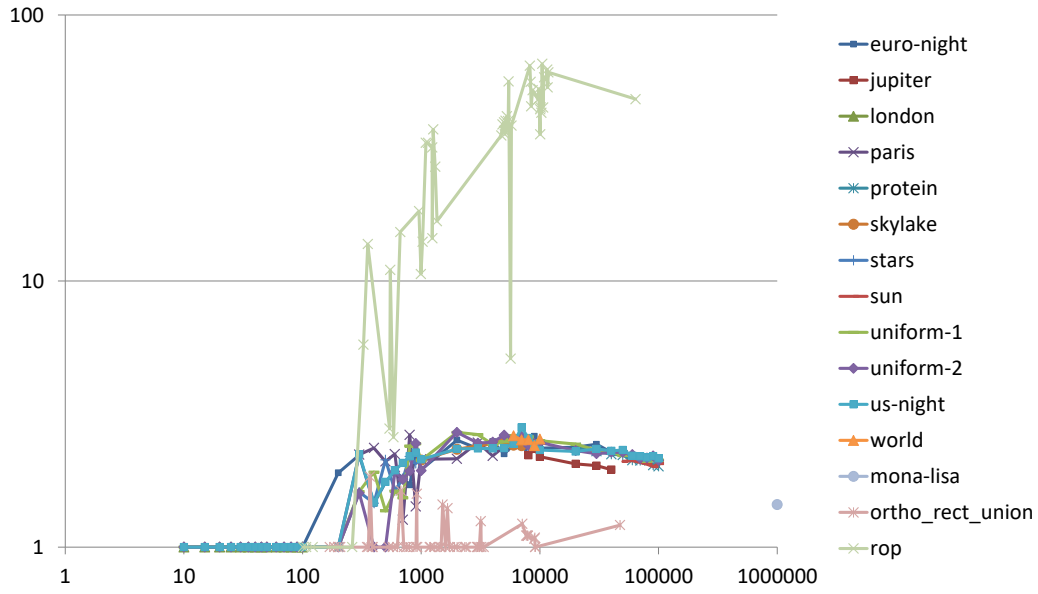


Figure 5 Evolution of $p_{impr} + 1$ (in ordinate) depending on the number of points of the instance (in abscissa). A logarithmic scale is used on both axes for clarity.

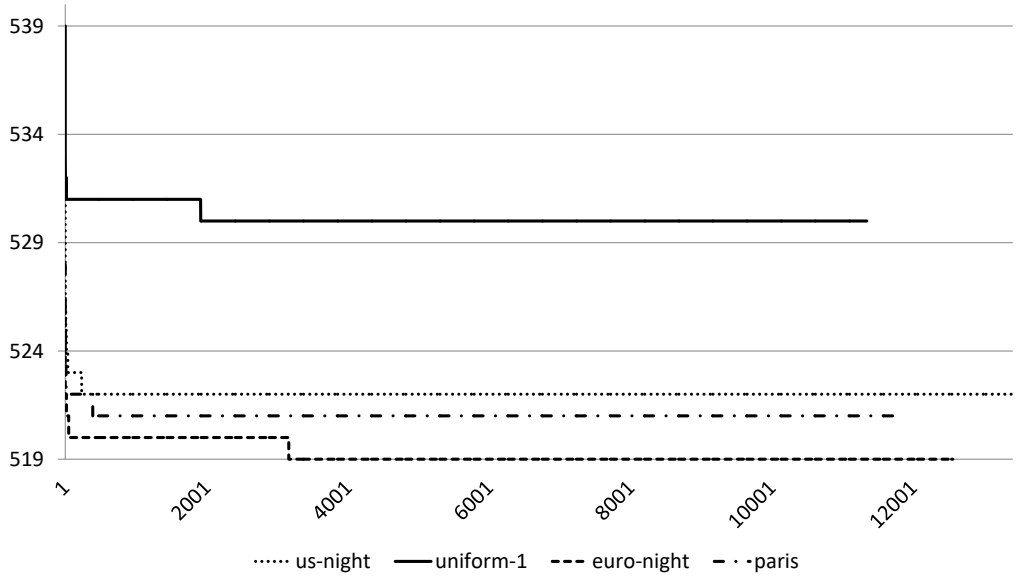


Figure 6 Convergence curve of the algorithm for 1,000 point instances over generations.

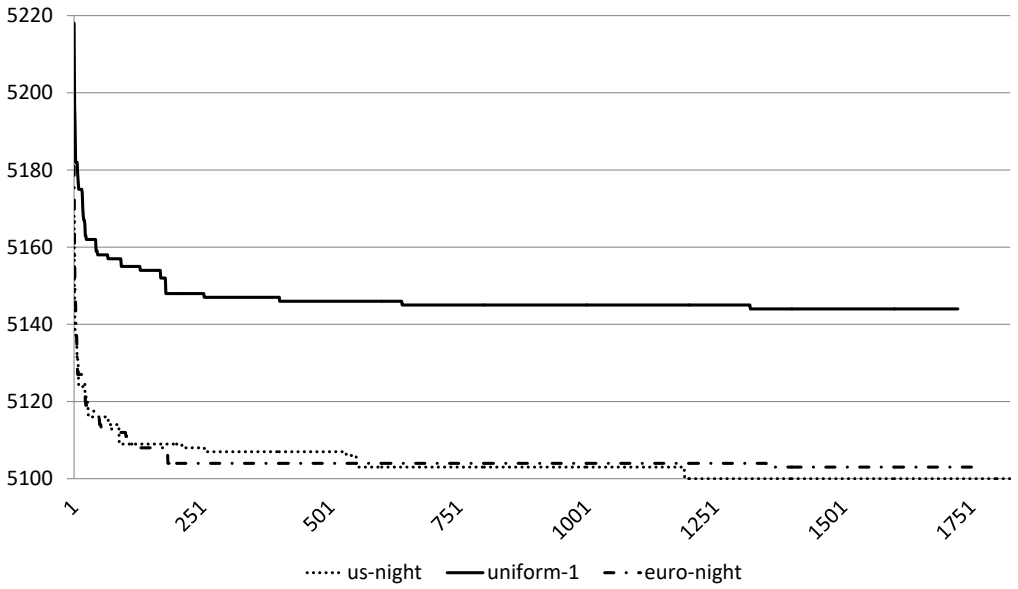


Figure 7 Convergence curve of the algorithm for 10,000 point instances over generations.

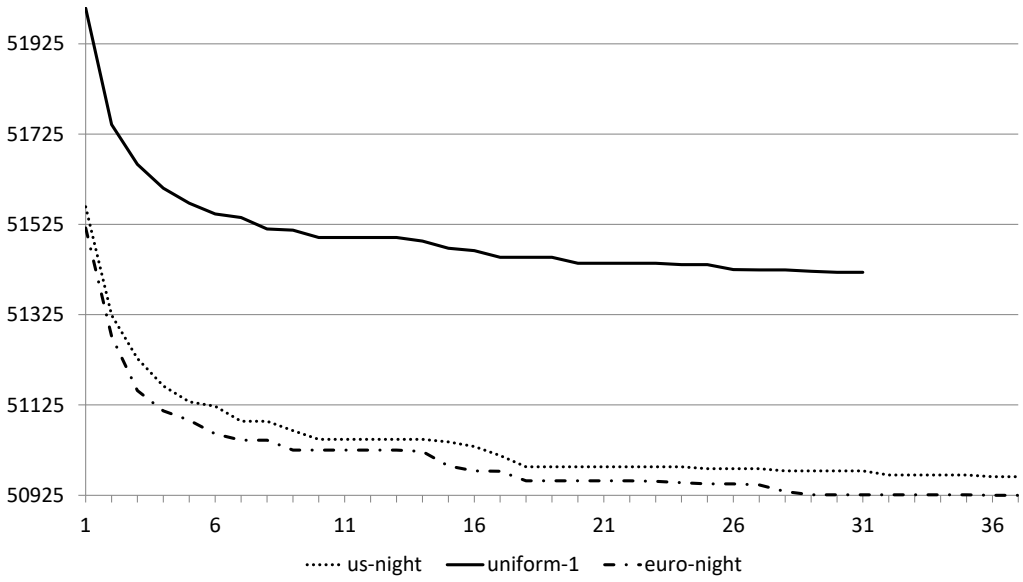


Figure 8 Convergence curve of the algorithm for 100,000 point instances over generations.

4 Conclusion

The proposed memetic algorithm has proven its overall effectiveness by ranking second among the best algorithms competing at the 2020 Computational Geometry Challenge on the Minimum Convex Partition problem. As pointed out in the analysis of section 3.3, significant fitness differences are observed between instances, belonging to the same class or not. In spite of this, the proposed algorithm does not have instance-specific parameter settings. These differences between instances should be studied, as well as the stability of the algorithm on each class of instances, to lead to an improved variant of our approach.

References

1 CGAL, Computational Geometry Algorithms Library. http://www.cgal.org.

2 Erik D. Demaine, Sándor P. Fekete, Phillip Keldenich, Dominik Krupke, and Joseph S. B. Mitchell. Computing convex partitions for point sets in the plane: The cg:shop challenge 2020, 2020. arXiv:2004.04207.

3 Nicolas Grelier. Minimum convex partition of point sets is NP-hard, 2019. arXiv:1911.07697.

4 Laurent Moalic and Alexandre Gondran. Variations on memetic algorithms for graph coloring problems. *Journal of Heuristics*, 24(1):1–24, 2018. doi:10.1007/s10732-017-9354-9.

5 Pablo Moscato and Carlos Cotta. A Gentle Introduction to Memetic Algorithms. In F. Glover and G. A. Kochenberger, editors, *Handbook of Metaheuristics*, pages 105–144. Springer, 2003.