Crossing minimisation heuristic solver for PACE 2024: NV_OCM

André Nusser
CNRS, Inria Center at Université Côte d'Azur, France

Juliette Vlieghe
Technical University of Denmark, Denmark

Abstract

This is the description of our solver for the heuristic track of the PACE Challenge 2024.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases PACE 2024 Challenge, heuristics

Supplementary Material The code source is available at `https://github.com/JulietteVl/crossing-minimization` and is associated with the DOI `https://zenodo.org/doi/10.5281/zenodo.11400797`

1 Our Approach

This is the solver description of our solver for the heuristic track of the PACE challenge 2024. It solves the problem of One-Sided Crossing Minimisation (OCM). In the following, we consider a graph G as input and denote with n its number of vertices and with m its number of edges. We call the vertices for which we have to choose the order to be the *free vertices*.

We employed two main approaches.

1.1 Ordering Heuristics

In a first step, we use three heuristics known as the median heuristic, the barycenter heuristic and the greedy switching heuristic. The median heuristic was introduced in [1] and it works as follows. Each free vertex is assigned the median value of the position of its neighbours, and the free vertices are ordered accordingly (breaking ties arbitrarily). This heuristic is a 3-approximation. In particular, it finds an optimal solution when there exist an optimal solution with no crossings. The barycenter heuristic was introduced in [3] and it works as follows. Each free vertex is assigned the average value of the position of its neighbours, and the free vertices are ordered accordingly (breaking ties arbitrarily). The experiments performed in [2] suggests that it performs better than the median heuristic in practice, both in general and for sparse graphs. The greedy switching heuristic [1] switches adjacent vertices when it improves the crossing count. This requires that we can easily evaluate whether the crossing count is improved. One way to do this is to compute the crossing count for each pair of free vertices: this is the difference in the number of crossings when switching these two vertices. If we want to obtain all crossing counts, then we can compute what is called the crossing matrix: a matrix that for every pair of free vertices contains their crossing count. On the provided instances, the space to store the crossing matrix can be a problem, but the time was no problem. Therefore we compute the crossing counts naively in $O(m^2)$ time.

The median and barycenter heuristics are global heuristics, while the greedy switching is a local heuristic, i.e., its result will depend on the solution that we start with. Therefore we try the greedy switching on a graph ordered with the median heuristic and on a graph ordered with the barycenter heuristic. We keep the best of the two results and proceed with the heuristic described in the next part.

1.2 Iterative Approach

Assume we can merge vertices. Each new edge is assigned a weight that corresponds to the sum of the original edges that it includes. There is an intra-edge crossing count that comes from edge crossings with endpoints in the set of vertices that was merged. The intra-edge crossing count is not affected by the ordering of the vertices. There is also an inter-edge crossing count coming from edge crossings with only one endpoint in the set of merged vertices. If two such edges cross, the corresponding contribution is the product of their weights.

To improve the previous result, we apply the greedy switching heuristic on different "scales". We start by splitting the free vertices in 2 parts. We create a new graph where each half of the vertices is merged into a vertex, and apply the greedy switching algorithm on this graph. Then we zoom in: we split the original graph in 4 parts instead, and apply the greedy switching. We double the number of groups until we finish with a final greedy switching on the original graph.

Intuitively, we do not expect the first iterations to produce changes if a first heuristic has been applied in a preprocessing step. However, this method allows to move smaller groups of vertices locally, instead of single vertices only, which results in a small improvement. On the medium test set, this increases the score from 59.53 to 59.64, so the error is reduced by almost a quarter.

2 Experiments

Here we give some implementation details and the results of some experiments. Our implementation was written in C++.

The repository includes a workflow which among other things assess the performance on the code on the medium data set. It output the ratio of our crossing count to the crossing count of the provided solution, and their sum, as well as the running time.

Running Time

The running time on the medium set is less than two seconds. It does not seem relevant to compare variations of the solution based on their time on this dataset. In the public set, a typical running time would be between a fraction of seconds to 20 seconds, suggesting that significantly more complex algorithms could run during the allocated time. There are still a couple of instances that take significantly longer than 5 minutes: instance 10, that has an unusually high amount of vertices, and instance 44, that has an unusually high amount of edges. For those, the output will be an intermediate step.

Solution Quality

The medium test set results in a score of 59.64 on 60 instances. The biggest deviation from the provided solution is of about 5%.

Hard Instances

The instances 38, 39, 40 of the medium test set contribute for almost half of the difference of the sum of ratios to the ideal sum. Visually, it is possible to see that a few vertices are misplaced in a way that greatly increase their crossing count. If such vertices can be

identified, it would make sense to remove such vertices from the order an greedily reinsert them.

References

1 Peter Eades and Nicholas C. Wormald. Edge crossings in drawings of bipartite graphs. *Algorithmica*, 11(4):379–403, Apr 1994. doi:10.1007/BF01187020.

2 Michael Jünger and Petra Mutzel. *2-Layer Straightline Crossing Minimization: Performance of Exact and Heuristic Algorithms*, pages 3–27. URL: https://worldscientific.com/doi/abs/10.1142/9789812777638_0001, arXiv:https://worldscientific.com/doi/pdf/10.1142/9789812777638_0001, doi:10.1142/9789812777638_0001.

3 Kozo Sugiyama, Shojiro Tagawa, and Mitsuhiko Toda. Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(2):109–125, 1981. doi:10.1109/TSMC.1981.4308636.