A Simulated Annealing based heuristic solver for the One-Sided Crossing Minimization Problem

tlopez - Student Submission

Toan Lopez
Student of Université Paris Dauphine, France

Florian Sikora
Lamsade, Université Paris Dauphine, France

Abstract

The 2024 edition of the Parameterized Algorithms and Computational Experiments Challenge (PACE) focuses on the One-Sided Crossing Minimization Problem (OSCM) in graph drawing. In the following, we outline the main functionalities of our contribution to its Heuristic Track. Our algorithm is based on OSCM heuristics refined by the Simulated Annealing metaheuristic (SA), in what is called a Two-Stage Simulated Annealing (TSSA). It also features a graph reduction rule.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases One-Sided Crossing Minimization, Graph drawing, Heuristics, Simulated Annealing, Two-Stage Simulated Annealing

Related Version doi.org/10.5281/zenodo.11539309

Supplementary Material (*Source code*) github.com/toanlpz/PACE2024-TLPZ

1 Description

Please check the challenge description on their website.
Our solver was implemented in C. The *pacegraph.c* file handles graph loading from *stdin* to memory. The variables *uint32_t nA* and *uint32_t nB* stores the size of set A and B, the variable *uint32_t n* stores *nA + nB* and the variable *uint32_t m* the number of edges in our graph. The graph is stored using contiguous adjacency lists in *uint32_t* all_adj* while *uint32_t** adjacency_arr* stores the beginning of each individual adjacency list. We initialize them from the edge list we get from *stdin* that we pass to *build_graph()*, that sorts each adjacency lists. We also count the degree of each vertex, which we use in the median heuristic developed P. Eades and N. C. Wormald [3].
The first crucial technique is the reduction of our graph. We group together the different vertices from B that have the exact same neighbours in A. This process is done in *build_reduced_graph()*, that builds the corresponding edge list of our reduced graph and pass it again to *build_graph()*. Also, if there is too few vertices that are equivalent to each other, we halt this process as it would take too much time for nothing. The *float threshold* parameter in *build_reduced_graph()* is there to control this. We set it to 0.75, so only if the graph is reduced to below 75% of its original size will this reduction occur. While it was developed independently, this technique reminds us of one of the kernel reduction rules developed by V. Dujmović, H. Fernau and M. Kaufmann[2].
Then, we compute the barycenter and median heuristics by P. Eades and N. C. Wormald [3]. We make use of a segment tree based algorithm [1] to compare both heuristics and select the better one.
We then apply our SA procedure. First, we use the work of James M. Varanelli [5] on TSSA to compute a starting temperature based on mean and deviation of the total crossing number on a few random solutions (We ran tests to make sure it would not impede on the

SA performance to compute these on a limited set of random solutions).
During our SA, we use the variable temperature decrement rule developed by Aarts and Van Laarhoven[4]. Taking the time limit into account, we had to put a limit on how slow the temperature could decrease (by taking the minimum between the value given by the variable decrement rule and 0.999). We enabled our SA to restart from our best solution at a high temperature, or else it would sometimes stay in local minima, not being able to maintain the quasi-equilibrium needed for convergence, due to a too large decreasing rate. This reset occurs if, for a 100 consecutive iterations (of the Metropolis-Hastings algorithm that makes up the SA), the number of crossings in our current solution stays the same.
Finally, the last addition to our SA is a way to counteract some overshooting that sometimes happen due to a too high starting temperature, which ruins our initial solution. If we find, as the SA resets, that the cost of our current solution is at least two times larger than the cost of our best solution, the next iteration will start at half its starting temperature. We then half our temperature down until it is sufficiently low for it to not scramble our initial solution beyond recovery.
During our SA, we make use of the *int64_t cost_uv_to_vu()* function to tell us the difference between c_{uv} and c_{vu}. It computes this going through both u's and v's adjacency lists only one time, so we can quickly computes the evolution of the cost of our solution if we swap two side by side vertices in B.
Finally, if we reduced our graph in the beginning, we convert our solution back to a corresponding one in the non-reduced graph.

References

1 Wilhelm Barth, Michael Jünger, and Petra Mutzel. Simple and efficient bilayer cross counting. In Michael T. Goodrich and Stephen G. Kobourov, editors, *Graph Drawing*, pages 130–141, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

2 Vida Dujmović, Henning Fernau, and Michael Kaufmann. Fixed parameter algorithms for one-sided crossing minimization revisited. *Journal of Discrete Algorithms*, 6(2):313–323, 2008. Selected papers from CompBioNets 2004. URL: https://www.sciencedirect.com/science/article/pii/S1570866707000469, doi:10.1016/j.jda.2006.12.008.

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

4 Peter J. M. Van Laarhoven and Emile H. L. Aarts. *Simulated Annealing: Theory and Applications*. Springer, Dordrecht, 1987. doi:10.1007/978-94-015-7744-1.

5 James Matthew Varanelli. *On the acceleration of simulated annealing*. PhD thesis, USA, 1996. AAI9701322.