Arcee: An OCM-Solver (Student Submission)

Kimon Boehmer
Université Paris-Saclay, France

Lukas Lee George
Technical University Berlin, Germany

Fanny Hauser
Technical University Berlin, Germany

Jesse Palarus
Technical University Berlin, Germany

Abstract

The 2024 PACE Challenge focused on the One-Sided Crossing Minimization (OCM) problem, which aims to minimize edge crossings in a bipartite graph with a fixed order in one partition and a free order in the other. We describe our OCM solver submission that utilizes various reduction rules for OCM and, for the heuristic track, employs local search approaches as well as techniques to escape local minima. The exact solver uses an ILP formulation and branch & bound to solve an equivalent Feedback Arc Set instance.

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

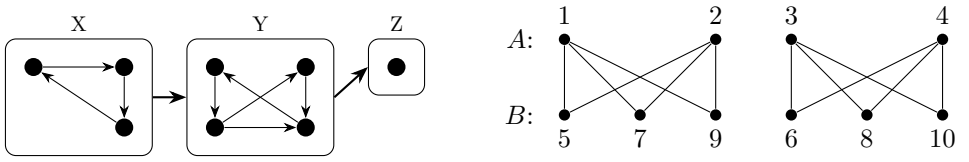
Keywords and phrases PACE 2024, One-Sided Crossing Minimization, OCM

Supplementary Material Source Code: `https://github.com/lucidLuckylee/pace_2024`

Acknowledgements We want to thank Mathias Weller and André Nichterlein for their advice and the *Internet Architecture and Management Research Group TU Berlin* for their hardware support.

1 Introduction

In the Parameterized Algorithms and Computational Experiments (PACE) Challenge of 2024, the problem of interest was One-Sided Crossing Minimization (OCM). In this problem, we are given a bipartite graph with vertex partitions A and B which are drawn horizontally and in parallel. A comes with a fixed linear order and is thus called the set of *fixed* vertices, while the ordering of the vertices in B is unknown (we call these vertices *free*). The goal is to find an ordering of the vertices in B that minimizes the total number of edge crossings when all edges are drawn with straight lines. Usually, exact and heuristic solvers for OCM will first require the computation of the so-called *crossing matrix* or *crossing numbers* [2, 5, 6, 10]. An entry c_{uv} with $u, v \in B$ denotes the number of edge crossings between edges incident to u and edges incident to v, when u appears before v in the ordering. The *penalty graph* of our OCM instance is a directed graph (B, E) where $E := \{(u, v) \in B^2 \mid c_{uv} < c_{vu}\}$ with edge weights $w((u, v)) = c_{vu} - c_{uv}$. Sugiyama et al. [14] observed a connection between OCM and the Feedback Arc Set of the penalty graph: An optimal ordering of the vertices B for OCM is equal to a topological ordering when an optimal Feedback Arc Set is removed from the penalty graph. In the following, we consider OCM instances and graphs to be *large* if the solver opts not to generate and store their crossing matrix and penalty graph due to memory limitations. In our submitted solver all instances with more than $10,000$ free vertices are considered large.



(a) Strongly connected components of a penalty graph.

(b) OCM instance splittable by partitioning the set of free vertices into $\{\{5, 7, 9\}, \{6, 8, 10\}\}$.

Figure 1 Graph splitting approaches.

2 Data Reduction Rules

The following methods are employed in the heuristic, exact and parameterized track. Before applying data reduction rules we try to split the OCM instance into several smaller instances. We can solve each strongly connected component of the instance's penalty graph individually and concatenate their solutions in the topological order of the penalty graph's strongly connected components (visualized in Figure 1a).

If the graph is large, then the above approach is infeasible. Instead, we try to split the graph by partitioning the free vertices B into non-empty subsets $\mathcal{B} = \{B_1, B_2, \dots B_k\}$ such that there are no vertices $u, v, w \in A$ with u before v before w in the fixed order of A and sets $B_i, B_j \in \mathcal{B}$ with $u, w \in N(B_i)$ and $v \in N(B_j)$. In other words the neighborhood intervals of the elements of $\mathcal{B}$ do not overlap in the set of fixed vertices We can split a graph into the induced subgraphs of each partition element and potentially further split these subgraphs with the aforementioned splitting approach. An example OCM instance that can be split via partitioning is Figure 1b.

Additonaly, we apply data reductions proposed by Dujmovic et al. [2]. In perticular their rules `RR1`, `RR2`, `RRLO1` in unmodified form and a modified version of their `RRlarge` rule that accounts not only for an upper bound but also for the trivial lower bound described by Nagamochi [11].

3 Heuristic Track

We use different approaches to find a heuristic solution depending on the size of the graph. For small and medium-sized graphs, the repeated application of our methods leads to better results, but on large graphs, even a single application may stress the resource limit. The first step in our heuristic for small to medium graphs is to compute an initial order with the median heuristic which was introduced by Eades and Wormald [3].

Local Search

Sifting To improve an order, we use *sifting*, which was first introduced by Rudell [13]. One vertex is taken at a time and placed at the position in the order which minimizes the total number of crossings. We do this exhaustively for all free vertices of the graph, but choose the sequence in which we sift the vertices randomly. After applying sifting on the order computed by the median heuristic, we apply it on random orders and store the order with the fewest crossings.

Swapping To escape local optima, we use *force swapping* if no better solution was found for 592 iterations[1] of using a random order with sifting. Force swapping is done by choosing two vertices and swapping their positions in the order. Then, we use sifting to improve the order while requiring that the relative position of the two vertices remains unchanged. We iterate through all vertices in a random order, swapping each vertex with its immediate right neighbor within the current best ordering. Subsequently, we increment the distance from the swapped vertices in each iteration by 9 until the distance is greater than 90.

Large Graphs

For large graphs, we first compute an order with the median heuristic and try to improve it by swapping neighboring vertices in the order until 10% of the available time is used. We repeat the same with the barycenter heuristic [14]. Then, we use a variation of sifting to improve the best order found so far. Instead of checking all possible positions for each vertex in the order, we skip the vertex if the total number of crossings increases by 20,000 or more. We sift the vertices until the time limit is reached.

4 Exact Track & Parameterized Track

For the exact solver we mainly use the idea described in the introduction to translate our OCM to an FAS instance. Our approach for solving the FAS instance is based on an algorithm by Grötschel et al [4]. The idea is to solve the FAS problem iteratively, by only looking at a subset $\mathcal{C}$ of all cycles from the input graph G and add cycles until one can guarantee that the solution, when only looking at the cycles $\mathcal{C}$, are also optimal for the initial graph.

ILP To now solve a partial instance of feedback arc set with the cycles $\mathcal{C}$, we use the following ILP formulation, which was widely used to solve FAS [4, 7, 12]:

$$\begin{aligned} \min \sum_{e \in E} w(e) \cdot y_e & \\ \text{s.t. } y_e \in \{0,1\} & \qquad \text{for all } e \in E \\ \sum_{e \in C} y_e \geq 1 & \qquad \text{for all } C \in \mathcal{C} \end{aligned}$$

Where we have a variable y_e for each $e \in E$, which is 1 if and only if the edge e is part of an FAS. As a solver we use SCIP v9.0 [1] and together with row generation and a warm startup so it can use results from the previous iteration, when solving the next iteration with more cycles.

Branching For graphs where the upper and lower bound differ by at most 10, we use a branch & bound algorithm to solve FAS. The algorithm branches on the edges of a cycle, uses a meta-heuristic by Lan et al. [8] as upper bound and a packing lower bound of the cycles.

The main reason for the use of the branching comes into play for the parameterized track. Here, our graphs decompose into a lot of small components using the splitting described in Section 2. When we then used SCIP to solve FAS, there was a large overhead in calling the ILP compared to the time the ILP needed to solve the instance.

[1] The exact values were found with SMAC3 using a subset of the public instances for training [9]

References

1 Suresh Bolusani, Mathieu Besançon, Ksenia Bestuzheva, Antonia Chmiela, João Dionísio, Tim Donkiewicz, Jasper van Doornmalen, Leon Eifler, Mohammed Ghannam, Ambros Gleixner, Christoph Graczyk, Katrin Halbig, Ivo Hedtke, Alexander Hoen, Christopher Hojny, Rolf van der Hulst, Dominik Kamp, Thorsten Koch, Kevin Kofler, Jurgen Lentz, Julian Manns, Gioni Mexi, Erik Mühmer, Marc E. Pfetsch, Franziska Schlösser, Felipe Serrano, Yuji Shinano, Mark Turner, Stefan Vigerske, Dieter Weninger, and Lixing Xu. The SCIP Optimization Suite 9.0. Technical report, Optimization Online, February 2024. URL: https://optimization-online.org/2024/02/the-scip-optimization-suite-9-0/.

2 Vida Dujmovic, Henning Fernau, and Michael Kaufmann. Fixed parameter algorithms for one-sided crossing minimization revisited. *J. Discrete Algorithms*, 6(2):313–323, 2008. doi:10.1016/J.JDA.2006.12.008.

3 Peter Eades and Nicholas C Wormald. Edge crossings in drawings of bipartite graphs. *Algorithmica*, 11:379–403, 1994.

4 Martin Grötschel, Michael Jünger, and Gerhard Reinelt. Comments on "an exact method for the minimum feedback arc set problem". *ACM J. Exp. Algorithmics*, 27:1.3:1–1.3:4, 2022. doi:10.1145/3545001.

5 Michael Jünger and Petra Mutzel. 2-layer straightline crossing minimization: Performance of exact and heuristic algorithms. In *Graph algorithms and applications i*, pages 3–27. World Scientific, 2002.

6 Yasuaki Kobayashi and Hisao Tamaki. A fast and simple subexponential fixed parameter algorithm for one-sided crossing minimization. *Algorithmica*, 72(3):778–790, 2015. doi:10.1007/S00453-014-9872-X.

7 Pichayapan Kongpanna, Deenesh K. Babi, Varong Pavarajarn, Suttichai Assabumrungrat, and Rafiqul Gani. Systematic methods and tools for design of sustainable chemical processes for co_2 utilization. *Comput. Chem. Eng.*, 87:125–144, 2016. URL: https://doi.org/10.1016/j.compchemeng.2016.01.006, doi:10.1016/J.COMPCHEMENG.2016.01.006.

8 Guanghui Lan, Gail W. DePuy, and Gary E. Whitehouse. An effective and simple heuristic for the set covering problem. *Eur. J. Oper. Res.*, 176(3):1387–1403, 2007. doi:10.1016/J.EJOR.2005.09.028.

9 Marius Lindauer, Katharina Eggensperger, Matthias Feurer, André Biedenkapp, Difan Deng, Carolin Benjamins, Tim Ruhkopf, René Sass, and Frank Hutter. Smac3: A versatile bayesian optimization package for hyperparameter optimization. *Journal of Machine Learning Research*, 23(54):1–9, 2022. URL: http://jmlr.org/papers/v23/21-0888.html.

10 Christian Matuszewski, Robby Schönfeld, and Paul Molitor. Using sifting for k -layer straightline crossing minimization. In Jan Kratochvíl, editor, *Graph Drawing, 7th International Symposium, GD'99, Stirín Castle, Czech Republic, September 1999, Proceedings*, volume 1731 of *Lecture Notes in Computer Science*, pages 217–224. Springer, 1999. doi:10.1007/3-540-46648-7_22.

11 Hiroshi Nagamochi. On the one-sided crossing minimization in a bipartite graph with large degrees. *Theor. Comput. Sci.*, 332(1-3):417–446, 2005. doi:10.1016/J.TCS.2004.10.042.

12 TK Pho and L Lapidus. Topics in computer-aided design: Part i. an optimum tearing algorithm for recycle systems. *AIChE Journal*, 19(6):1170–1181, 1973.

13 R. Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *Proceedings of 1993 International Conference on Computer Aided Design (ICCAD)*, pages 42–47, 1993. doi:10.1109/ICCAD.1993.580029.

14 Kozo Sugiyama, Shojiro Tagawa, and Mitsuhiko Toda. Methods for visual understanding of hierarchical system structures. *IEEE Trans. Syst. Man Cybern.*, 11(2):109–125, 1981. doi:10.1109/TSMC.1981.4308636.