heiCross – Student Submission

Adil Chhabra
University of Heidelberg, Germany

Marlon Dittes
University of Heidelberg, Germany

Alvaro Garmendia
University of Heidelberg, Germany

Ernestine Großmann
University of Heidelberg, Germany

Tomer Haham
University of Heidelberg, Germany

Shai Peretz
University of Heidelberg, Germany

Henrik Reinstädtler
University of Heidelberg, Germany

Antonie Wagner
University of Heidelberg, Germany

Henning Woydt
University of Heidelberg, Germany

Abstract

The one-sided crossing minimization problem (OCM) asks for an order of one layer of a bipartite graph. The goal is to minimize the number of edge crossings with the fixed order of vertices in the other layer. Our heiCross solver employs a reduce and peel approach specifically designed for the heuristic track of the challenge. It starts by partitioning the graph trough articulation points. We reduce the graph size by iteratively applying reduction rules and partitioning, which still results in optimal solutions. If this approach can no longer be executed, we use a heuristic reduction to build a heuristic solution. When a partition reaches a small enough size, we solve it exactly.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases PACE2024, crossing number, heuristics, OCM

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

Supplementary Material *Source Code (Zenodo)*: `https://doi.org/10.5281/zenodo.12507916`
Source Code (GitHub): `https://github.com/MarlonDittes/heiCross`

Acknowledgements We would like to thank the challenge organizers and other participants for their time and continuous support throughout the challenge. We would also like to thank Professor Schulz for his guidance and help during this time.

1 Introduction

The PACE2024 challenge asks for a solution to the one-sided crossing minimization problem (OCM), an NP-hard graph-drawing problem. Consider a bipartite graph $G = ((A \cup B) = V, E)$ with a fixed linear order of the nodes in A. The goal is to find an order of the moveable nodes in B, such that the number of edge crossings in a drawing of G with A and B on parallel lines is minimized. Note, that two edges (a_1, b_1) and (a_2, b_2), with $a_1, a_2 \in A$, $b_1, b_2 \in B$ and $a_1 < a_2$, produce a crossing in a drawing of G if and only if $b_2 < b_1$. If $x < y$ for two vertices in A or B, then x is to the left of y.
The heiCross solver focuses on the heuristic track of the challenge. There are several heuristics known for delivering already good solutions to the OCM problem. The so-called barycenter heuristic [3] finds an $O(\sqrt{|V|})$-approximation solution, while the median heuristic [2] computes a 3-approximation.
In Section 2.1 we present a local improvement of the median heuristic as the initial start of our algorithm. Sections 2.2 and 2.3 focus on the partitioning and reduction strategies. In Section 2.4 the approach for computing a heuristic solution is explained.

2 Solver Description

2.1 Initial Start

First, we decided to explore the performance of the median heuristic [2]. To then further improve the solution quality, we apply a greedy strategy to find local improvements.
The median heuristic sorts the movable nodes based on the median node of its neighbourhood. For a node of degree 0 the median is defined as 0. Another consideration is how to solve conflicts between nodes of the same median. In this case, we opted to shift even-degree nodes to the right and uneven ones to the left.
This scheme yields a 3-approximation, however, it has been proven by Eades et. al. [2] that the results approximate the optimal one if the density $|E|/(|A||B|)$ becomes 1. Moreover, by going through all vertices to calculate the medians and then sorting them, we only need $O(|B|\log(|B|))$ time to get an improved ordering.
One of these improvements consists of our greedy heuristic. When building the graph, we compute two scores for each node u, each of those reflecting the amount of crossings generated with nodes to the right ($\text{right}(u)$) and left ($\text{left}(u)$) of it. We use these scores to define the following 3 rules among adjacent nodes $u, v \in B$ with $u < v$:

$$\text{(R1): } \text{left}(u) = 0 \wedge \text{left}(v) > 0, \tag{1}$$

$$\text{(R2): } \text{right}(u) > \text{left}(u) \wedge \text{left}(v) > \text{right}(v), \tag{2}$$

$$\text{(R3): } \text{right}(u) > \text{left}(v). \tag{3}$$

Intuitively, these rules try to balance the number of crossings on each side of a node, but we only swap u and v if it improves our solution. Our algorithm scans the permutation, from left to right, and checks for these rules for every pair of adjacent nodes until no rules can be applied. After a rule triggers, the crossings caused by the ordering $u < v$ and $v < u$ are computed and the best result is taken.

2.2 Graph Partitioning

After obtaining an initial solution that provides a 3-approximation, our algorithm partitions the graph into independent blocks while maintaining optimality. This approach simplifies the problem into smaller, more manageable sub-problems, each of which can be solved independently, thereby improving overall computational efficiency and solution quality. The partitioning is done at each connected component and at every vertex $v \in A$ known as an articulation point. A vertex v in a graph G is called an articulation point if its removal increases the number of connected components of G. We find these articulation points in $O(|V| + |E|)$ using Tarjan's algorithm [4], where we perform a depth-first search (DFS).
Once the articulation points are identified, we proceed to determine all connected components by removing these articulation points from the graph. The newly formed connected components constitute new sub-graphs, where with the help of vertex splitting, articulation points are added only to their original and newly caused independent sub-graphs. In the newly formed sub-graph, split vertices only contain edges between nodes of the same sub-graphs.

2.3 Data Reductions

To reduce the size of the partitions, multiple data reduction strategies were tried, but only three of them lead to better solutions. We call the first two *naive reductions* because their execution already solves a partition optimally such that it can not be reduced further and

will not take part in the heuristic reduction step. The first naive reduction simply checks if a partition has zero edges, as in such cases, no permutation of the nodes will alter the number of crossings. The other naive reduction detects the completeness of a partition, since then the node order does not affect the number of crossings within the partition.
The twin reduction [1] finds nodes in A with the same neighborhood. For a node v its neighborhood $N(v)$ is defined as the set of nodes it is adjacent to. If for two nodes $u, v \in A$, $N(u) = N(v)$, they are reduced to one node with increased edge weights. Note, that this property can affect more than two nodes at the same time, such that multiple nodes could be reduced to one. In the final ordering the twins are always positioned next to each other.

2.4 Heuristic Reduction

Once we are unable to further partition the graph and there are no applicable reductions, we decrease the instance heuristically. This is done by temporarily removing one moveable node $v \in B$ of the current graph and computing the solution for the remaining vertices using our algorithm recursively. The removed node v is then placed back into the graph by calculating the crossings it would partake in at each possible position in the ordering of B. Afterwards, we greedily choose the lowest crossing setting.
Regarding the heuristic removal of a node $v \in B$, we use three different methods for ranking, upon which we then remove the top-ranking node:

$$\text{(M1): } \deg(v) \tag{4}$$

$$\text{(M2): } \operatorname{span}(v) \tag{5}$$

$$\text{(M3): } \frac{\operatorname{span}(v)}{\deg(v)} \tag{6}$$

where $\deg(v)$ is the degree of v. Given the neighborhood N_v of a node $v \in B$, we define the $\operatorname{span}(v)$ as follows:

$$\operatorname{span}(v) = \max_{u \in N_v} \operatorname{ind}(u) - \min_{w \in N_v} \operatorname{ind}(w)$$

where $\operatorname{ind}(v)$ is the index of node v.

When the size of $|A|$ in the graph is smaller or equal to certain bound L, we use an exact solver [5] to compute the exact ordering of the remaining vertices. This exact algorithm works by reducing the graph size using other exact reduction rules and then exhaustively searching the permutation tree with a branch-and-cut strategy to find the optimal vertex orderings.

2.5 Running the overall algorithm

In the following, we will explain how we combine all components to make the best use of the five minutes available during the heuristic track.
We first use the median heuristic [2] to create a good initial solution. We then use the approach described in Section 2.1 to improve upon the found solution. Afterwards, we then apply the reduce and peel approach described in Sections 2.2 and 2.3 together with the span method and bound $L = 0$ discussed in Section 2.4, since using no exact solver sometimes yielded better results. Subsequently, we run the same approach with the span method again whilst using bound $L = 10$.
To utilize the rest of the time, we repeatedly apply the reduce and peel technique using a random heuristic removal method whilst increasing the bound L by two after each call.
If the time limit is reached during any point of this procedure, we output the best solution found so far.

References

1 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.

2 Peter Eades and Nicholas C. Wormald. Edge crossing in drawing bipartite graphs. *Algorithmica*, 11(4):379–403, 1994. doi:10.1007/BF01187020.

3 Kozo Sugiyama, Shojiro Tagawa, and Mitsuhiko Toda. Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man, and Cybernetics*, 11:109–125, 1981. URL: https://api.semanticscholar.org/CorpusID:8367756.

4 Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972. arXiv:https://doi.org/10.1137/0201010, doi:10.1137/0201010.

5 Henning Martin Woydt. Crossingguard: An exact solver for the one-sided crossing minimization problem. https://github.com/HenningWoydt/PACE2024Exact/blob/master/CrossingGuard.pdf, 2024.