# A penalty graph based heuristic solver for the One-Sided Crossing Minimization problem

Carolin Rehs
Technical University of Dortmund, Germany

Eric Weidner[1]
Technical University of Dortmund, Germany

**Abstract**

The 2024 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 the Heuristic Track. The presented algorithm is based on the notion of penalty graphs, which describe the generation of intersections within the arrangement of two specific vertices relative to each other in the form of a directed graph. While previous algorithms with penalty graphs aim for exact solutions, we extend the idea to heuristic results using both Feedback Arc Set Problem (FAS) heuristics and OSCM heuristics. Obtained solutions are improved by an extended greedy-switch heuristic.

**2012 ACM Subject Classification** Theory of computation $\to$ Computational geometry

**Keywords and phrases** One-Sided Crossing Minimization, Graph drawing

**Supplementary Material** Source Code: 10.5281/zenodo.11542915

## 1 PACE Challenge

▶ **Definition 1.** *Let $G = (V_f \mathbin{\dot\cup} V_\ell, E)$ be a bipartite graph and let $\sigma_f$ be a linear ordering of the* fixed vertices *in $V_f$. The One-Sided Crossing Minimization Problem (OSCM) asks for an ordering $\sigma_\ell$ of the* loose vertices *in $V_\ell$ such that the number of edge crossings in a straight-line drawing of $G$ with $V_f$ and $V_\ell$ on two parallel lines following their linear ordering is minimized.*

*By $c[u_1, \ldots, u_k]$ we denote the number of crossings in a straight-line drawing with $V_f$ on the first line following its ordering and $u_1, \ldots, u_k$ on the second line in this order. The OSCM thus asks for some $\sigma_\ell$ such that $c[\sigma_\ell]$ is minimal.*

The requirements of the PACE challenge under the heuristic path are to solve a series of OSCM instances heuristically while not exceeding the memory limit of 8GB. After a maximum period of 5 minutes, a solution must be returned.

## 2 Initial solution with FAS and OSCM heuristics

To reduce the initial size of the given problem, we preprocess the given input by excluding all vertices with degree$= 0$, since they do not contribute to the generation of edge crossings and are thus appended in any order to a computed ordering.

[1] This submission is eligible for the Student Submission Reward. A clear majority of conceptual and implementation work was done by the student co-author.

▶ **Definition 2** (Penalty Digraph [6]). *Let $G = (V_f \dot{\cup} V_\ell, E)$ be a bipartite graph and $\sigma_f$ an ordering on $V_f$ The* weighted penalty digraph *$H = (W, F, p)$ is then defined by:*

$$
\begin{aligned}
&W = V_\ell \\
&F = \{(u,v) \in W \times W | c[u,v] < c[v,u]\} \\
&p : F \to \mathbb{N} \textit{ with } p(u,v) = c[v,u] - c[u,v] \textit{ for each } (u,v) \in F,
\end{aligned}
\qquad (1)
$$

Sugiyama et al. [6] introduce an exact approach to solving OSCM using a topological ordering of the penalty graph. As this graph is not necessarily acyclic, it is modified by reversing a minimum number of edges. Those edges can be found by solving the Minimum Feedback Arc Set Problem, i.e. finding the minimal set of edges that, when deleted, render the graph acyclic. While computing the penalty graph seems expensive compared to other heuristic approaches ($c_{u,v}$ needs to be computed for every pair of vertices in $V_\ell$), it allows the decomposition in strongly connected components and thus in smaller subproblems which can be regarded separately. Finding strongly connected components in the penalty graph is possible in $\Theta(|W| + |F|)$ [5], while topological sorting the resulting acyclic graph is $\mathcal{O}(|W| + |F|)$ with Kahn's algorithm [3].

We considered two possibilities to obtain efficient heuristics instead of exact results using penalty graphs. In a first approach, we used Feedback Arc Set heuristics to approximate a FAS in the penalty graph. In a second approach, we computed solutions for the OSCM subproblems based on the strongly connected components of the penalty graph with the barycenter heuristic (Where each Vertex is assigned a position based on the average position of its neighbours. The barycenter heuristic is a widely used heuristic for OSCM due to its simplicity and good performance [4]. In the submitted algorithm, we combine both approaches using the one that generates less edge crossings for a certain strongly connected component.

**FAS-heuristics** Many FAS heuristics are based on sorting the vertices according to some comparison criterion $d(v)$. Comparative experiments on the test set provided for PACE 2024 encourage the following modification of the criterion mentioned in [1]:

$$d(v) = \text{outdegree}(v)/\text{indegree}(v) \qquad (2)$$

In our experiments, this criterion produced the most accurate results amongst all considered FAS-heuristics. While it produced slightly better results than the approach using the barycenter heuristic, it does not deliver results with less edge crossings for every input graph of the PACE test set. Thus, our algorithm also calculates the barycenter heuristics and selects the ordering for the respective strongly connected component that generates the lower number of crossings.

**Tiebreaker-barycenter heuristic for OSCM** The classical definition [6] of the barycenter heuristic uses input i.e., lexicographical ordering for cases where multiple vertices are assigned to the same position. Based on benchmarks of the public test set, by using the related Median heuristic [4], (where every vertex is positioned according to the median position of its neighbours) as a tiebreaker, the resulting crossings are reduced by 8% compared to the classical barycenter heuristic. In pathological cases where the barycenter heuristic performs much worse than the Median, using the Median as a tiebreaker results in crossing counts that are at least on the same level or better than the Median heuristic.

## 3 Improving solutions with extended greedy-switch heuristic

After the initial solution has been computed, a series of local searches are performed on the nontrivial SCC in order to further minimize the number of generated crossings. The greedy switch heuristic [4] reverses the position of two neighbouring vertices if this reduces the number of crossings until no further relative improvement can be made. Here we use the following extension of this heuristic:

▶ **Definition 3** ($k$-step greedy switch). *Let $G = (V_f \dot\cup V_\ell, E)$ with $\sigma_f$ be an instance of OSCM and $\sigma_\ell$ a previously generated solution. Let further $k \in \mathbb{N}^+$. For all pairs of vertices $u_0$ and $u_k$ such that $\sigma_\ell = (\dots, u_0, u_1, u_2, \dots, u_{k-1}, u_k, \dots)$, exchange the positions of $u_0$ and $u_k$ such that $\sigma_\ell = (\dots, u_k, u_1, u_2, \dots, u_{k-1}, u_0, \dots)$ if and only if $c[u_k, u_1, u_2, \dots, u_{k-1}, u_0] < c[u_0, u_1, u_2, \dots, u_{k-1}, u_k]$ until no further improvement can be made.*

Comparing over $k$ steps also does not increase the asymptotic complexity as long as $k$ is constant: $c[u_k, u_1, u_2, \dots, u_{k-1}, u_0] < c[u_0, u_1, u_2, \dots, u_{k-1}, u_k]$ can be calculated in $\mathcal{O}(k \cdot C)$ time where $C$ is the complexity of determining $c[u, v]$ for arbitrary vertices $u$ and $v$, since the number of crossings generated between the examined vertices is the same on both sides and does not need to be computed.

Performing multiple iterations of this heuristic for different values of $k$ may result in finding improvements that were previously undetectable due to reaching a local minimum. Therefore, for each non-trivial strongly connected component we perform several repeated applications of the $k$-step greedy switch (for different values of $k$) until again no further improvement is observed.

For the sequence of values for k within these batches we found that, based on the public test data set, starting from $k = 2$ and incrementing up to a value of 30 followed by one application of the classical greedy switch heuristic to be a good compromise between runtime and local search range.

### Handling big instances

In order for the algorithm to comply with the restriction set in place by the PACE-Challenge, the algorithm has a maximum limit for the number of vertices a graph can have for the algorithm to stay within the memory bounds of 8 GB. If an instance is expected to exceed this limit, the algorithm performs the local search routine on the whole set of vertices that has been initialized with the tiebreaker-barycenter heuristic. This fallback ordering is computed in advance and also returned should the time limit be reached while computing the penalty graph.

### Runtime

Calculating the penalty graph takes $\mathcal{O}(|V_l|^2 \cdot C)$ time. Determining this value is linear with respect to the degree of the considered vertices [4], so in the case of a dense graph calculating the penalty graph takes $\mathcal{O}(|V_l|^2 \cdot |V_f|)$ time as a worst case estimate. The runtime complexity for the local search depends not only on $C$ but on the respective choice of $k$. Since PACE regulations specify that optimizations must conclude after 5 minutes for big instances, the algorithm might not even be able to finish a single run for $k = 1$ which is $\mathcal{O}(|V_l|^2 \cdot C)$ in the worst case [2].

## References

1. Ali Baharev, Hermann Schichl, Arnold Neumaier, and Tobias Achterberg. An exact method for the minimum feedback arc set problem. *ACM J. Exp. Algorithmics*, 26, apr 2021. `doi:10.1145/3446429`.
2. Peter Eades and David Kelly. Heuristics for reducing crossings in 2-layered networks. *Ars Combinatoria*, 21(A):89–98, 1986.
3. A. B. Kahn. Topological sorting of large networks. *Commun. ACM*, 5(11):558–562, nov 1962. `doi:10.1145/368996.369025`.
4. Erkki Mäkinen. Experiments on drawing 2-level hierarchical graphs. *International Journal of Computer Mathematics*, 36(3-4):175–181, 1990. `doi:10.1080/00207169008803921`.
5. M. Sharir. A strong-connectivity algorithm and its applications in data flow analysis. *Computers Mathematics with Applications*, 7(1):67–72, 1981. URL: `https://www.sciencedirect.com/science/article/pii/0898122181900080`, `doi:10.1016/0898-1221(81)90008-0`.
6. K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical system structures. *IEEE Trans. Syst. Man Cybern.*, 11(2):109–125, 1981.