# PACE Solver Description: Bob

**Sergey Pupyrev**
Menlo Park, CA, USA

## Abstract

We present `Bob`, a heuristic solver for one-sided crossing minimization, submitted to the 2024 edition of the Parameterized Algorithms and Computational Experiments (PACE) challenge.

**2012 ACM Subject Classification** Theory of computation → Graph algorithms analysis

**Keywords and phrases** crossing minimization, 2-layered drawing, balanced graph partitioning

**Supplementary Material** clone the sources at `https://github.com/spupyrev/pace2024-bob` or download the code at `https://doi.org/10.5281/zenodo.11551133`

## 1 Preliminaries

The 2024 PACE challenge (`https://pacechallenge.org/2024`) is on the one-sided crossing minimization problem (OSCM). The problem is to layout a bipartite graph on two layers, with a fixed vertex order on one of the layers, so as to minimize the total number of edge crossings. OSCM is known to be NP-hard [6,13], even for trees [3], but admits constant-factor approximations [7,14] and can be solved in FPT time [4,5,11].

Let $G = (U \cup V, E)$ be a bipartite graph with a partition $U$ and $V$ of the vertices and edges $E \subseteq U \times V$. The linear order of vertices in $U$ is fixed to $(1, 2, \ldots, |U|)$. The task is to find a linear order on $V$, denoted $<_V$ or simply $<$, such that the drawing of $G$ with vertices lying on two parallel lines and edges represented by straight-line segments has the minimum number of edge crossings. The set of vertices adjacent to $v \in V$ is denoted by $\mathcal{N}(v)$ and $\deg(v) = |\mathcal{N}(v)|$ is the degree of $v$. It is well-known that the number of crossings is fully determined by the relative positions of pairs of vertices. That is, let $\sigma(x, y)$ for $x, y \in V$ be the number of crossings between edges adjacent to $x$ and edges adjacent to $y$ when $x < y$ in the order. Then the total number of crossings in a drawing is $\sum_{x \in V} \sum_{y \in V : x < y} \sigma(x, y)$.

## 2 Solver Overview

The solver contains three main ingredients: a set of pre-processing *kernelization* rules simplifying the instance, recursively applied *balanced graph partitioning*, and a collection of post-processing *adjustments*. The latter two steps are randomized and may produce different results depending on the seed of the random number generator. Therefore, we apply the algorithm multiple times with different seeds for each instance and output a solution with the minimum number of crossings. Next we provide details on each of the three steps.

### 2.1 Kernelization

The goal of the step is simplify the input instance while keeping the optimal solution unchanged. We start by removing all isolated vertices from both parts. Then we identify and merge *duplicate* vertices $x \in V, y \in V$ having $\mathcal{N}(x) = \mathcal{N}(y)$. Note that after the merging we have a weighted graph in which every edge, $e$, is assigned a positive integer weight; it is straightforward to adjust the definitions of the crossing number to the weighted case. The next rule is to merge near-duplicate vertices, called *twins*. A vertex $x \in V$ with $\mathcal{N}(x) = \{u_1^x, \ldots, u_d^x\}, d = \deg(x)$ is a *twin* of $y \in V$ with $\mathcal{N}(y) = \{u_1^y, \ldots, u_d^y\}$ if $|u_i^x - u_i^y| \leq 1$ for all $1 \leq i \leq d$ and all $u_1^x, u_1^y, \ldots, u_d^x, u_d^y$ are adjacent only to $x$ and $y$.

The last kernelization rule is based on the representation of OSCM as an instance of the minimum feedback arc set problem (FAS). Consider a directed graph $H$ whose vertices are $V$ and there is a (weighted) directed edge $(x, y)$ whenever $\sigma(x, y) > \sigma(y, x)$ and edge $(y, x)$ whenever $\sigma(x, y) < \sigma(y, x)$. One can verify that OSCM is equivalent to solving FAS on $H$ [1, 8]. Thus, we find strongly connected components on $H$, place the components in a topological order and solve OSCM on each of the components separately.

## 2.2 Balanced Partitioning

The main stage of `Bob` is based on the recursive balanced graph partitioning scheme, which is often used to find one-dimensional graph layouts [2, 9, 12]. The algorithm combines recursive graph bisection with a local search optimization at each step. Starting with an input graph $G = (U \cup V, E)$ with $|V| = n$, we apply a bisection algorithm to obtain two disjoint subsets, $V_1, V_2 \subseteq V$ with $V_1 \cup V_2 = V$, of (approximately) equal cardinality, $n/2$. Then $V_1$ is placed on the set $\{1, \ldots, n/2\}$ and $V_2$ on the set $\{n/2 + 1, \ldots, n\}$. By doing so, we divide the problem into two sub-problems, each of half the size, and recursively compute orders for the two subgraphs induced by vertices $V_1$ and $V_2$, and the incident edges.

Every bisection step is a variant of the local search optimization. We start by (randomly) splitting $V$ into two sets, $V_1$ and $V_2$, of roughly equal size. Then, we iteratively exchange pairs of vertices between $V_1$ and $V_2$ to improve a certain objective. The process is repeated until a convergence criterion is met or the maximum number of iterations is reached. The final order is obtained by concatenating the two recursively computed orders for $V_1$ and $V_2$.

An important aspect of the algorithm is the objective to optimize at each bisection step. Note that at this step the order of vertices in the two parts is not yet determined. Hence, we define and utilize the *expected number edge crossings* for splitting $V$ into $V_1$ and $V_2$. For $x \in V_1$ and $y \in V_2$, this number can be computed exactly; this is $\sigma(x, y)$. For $x, y \in V_1$ (or symmetrically, $x, y \in V_2$), we estimate the number of crossings via $1.05 \times \min(\sigma(x, y), \sigma(y, x))$; the motivation is that in most practical instances, there exists a solution achieving the near-optimal bound for all pairs of vertices [10].

Finally, we stress that the lowest levels of the recursion with a few vertices can be solved optimally via a technique described in Section 3.

## 2.3 Adjustments

Given an order of $V$, we apply several heuristics trying to reduce the number of crossings. First heuristic considers (continuous) intervals of $V$ of a small size, e.g., $k \leq 20$, and re-orders the vertices of each interval optimally; refer to Section 3 for our implementation of the exhaustive search. Note that the process is iterative: once an interval is re-ordered, all overlapping intervals might become sub-optimal. To avoid excessive runtime of the iterative optimization, we dynamically maintain a queue of potentially sub-optimal intervals, avoiding re-optimization of unchanged intervals. The second heuristic places single vertices of $V$ into their (locally) optimal positions. Consider $x \in V$; the vertex produces $\sum_{y \in V : y < x} \sigma(y, x) + \sum_{y \in V : y > x} \sigma(x, y)$ crossings in the solution. We find the best position of $x$ in the order while keeping all the remaining vertices unchanged. Similarly, to the first heuristic, the process is iterative and the optimal position of a vertex might change while other vertices are re-ordered.

In practice, the two heuristics are the most effective for reducing crossings at the post-processing step. A number of less effective (but sometimes useful) heuristics are applied to the solution; we refer to the implementation of the solver for details.

## 3 Optimal Algorithms

This section describes two important sub-routines that are utilized throughout the algorithm and responsible for the majority of computation cycles while running the solver. Each of them describes an optimal algorithm for solving a special case of OSCM.

▶ **Lemma 1.** *Let $G = (U \cup V, E)$ be an instance of OSCM with $|V| = n$ and pre-computed values $\sigma(x, y)$ for all $x, y \in V$. Then the problem can be solved optimally in $O(n^2 2^n)$ time using $O(2^n)$ space.*

**Proof.** We use dynamic programming to compute the optimal number of crossings for each subset $S \subseteq V$; let $\mathrm{F}[S]$ denote the value. Clearly, $\mathrm{F}[S] = 0$ for all subsets with $|S| = 1$. For $|S| \geq 2$, we can choose which vertex $x \in S$ is placed the last in the order. Thus, $\mathrm{F}[S] = \min_{x \in S} \Big( \mathrm{F}[S \setminus \{x\}] + \sum_{y \in S \setminus \{x\}} \sigma(y, x) \Big)$. There are $2^n$ subsets of $V$ and choosing the last vertex and counting the corresponding crossings can be done in $O(n^2)$ steps. It is easy to reconstruct the optimal vertex order for OSCM, given F values for all subsets. ◀

The next result describes an algorithm to optimally "merge" two lists of vertices. Note that a special case of the algorithm (with $|V_1| = 1$) is an optimal insertion of a vertex into an existing vertex order.

▶ **Lemma 2.** *Let $G = (U \cup V, E)$ be an instance of OSCM with $|V| = n$, pre-computed values $\sigma(x, y)$ for all $x, y \in V$, and a partitioning of $V$ into (ordered) lists $V_1$ and $V_2$. There exists an $O(n^2)$-time algorithm finding an optimal order, $<$, in which $x < y$ whenever $x <_{V_1} y$ if $x, y \in V_1$ and $x <_{V_2} y$ if $x, y \in V_2$.*

**Proof.** Again, we use dynamic programming to compute the minimum number of crossings of merging two lists, $\mathrm{F}[i, j]$ where $0 \leq i \leq |V_1|$ and $0 \leq j \leq |V_2|$. It is convenient to count only inter-list crossings between vertices of $V_1$ and $V_2$, as the crossings between the vertices of $V_1$ (or $V_2$) are not affected by the merging. Thus, we have $\mathrm{F}[i, 0] = \mathrm{F}[0, j] = 0$ as the base case for all $i, j \geq 1$. When $i \geq 1$ and $j \geq 1$, we consider $i$-th and $j$-th vertices from the lists, $x = V_1[i]$ and $y = V_2[j]$, and observe that one of them is the last in the result. That is, $\mathrm{F}[i, j] = \min \Big( \mathrm{F}[i-1, j] + \sum_{1 \leq k \leq j} \sigma(V_2[k], x), \mathrm{F}[i, j-1] + \sum_{1 \leq k \leq i} \sigma(V_1[k], y) \Big)$. The optimal order is reconstructed from the computed F values. ◀

## 4 Implementation Details

While `Bob` was originally designed to participate in the HEURISTIC track of the PACE challenge, it can be applied for the EXACT and the PARAMETERIZED tracks too. For the former track, we run the solver multiple times (32, in our implementation) with different seeds and output the best solution only if the optimum is found in a pre-defined number of runs (10 or more, in our implementation). For the latter track, no changes to the algorithm in comparison to the HEURISTIC version is necessary. We use different time limits for the tracks (5 minutes for EXACT and HEURISTIC; 1 minute for PARAMETERIZED), and stop the execution once the limit is exceeded.

We want to emphasize another variant of the solver, called LITE, which skips the *adjustment* techniques (Section 2.3) of the algorithm. The version is able to process very large instances of OSCM at the cost of slightly worsened results. For example, every instance from the public dataset (containing up to $10^6$ vertices) is processed within 10 seconds; the result contains less than 0.5% extra crossings on top of the known optimum.

## References


1 Camil Demetrescu and Irene Finocchi. Removing cycles for minimizing crossings. *ACM J. Exp. Algorithmics*, 6:2, 2001. doi:10.1145/945394.945396.

2 Laxman Dhulipala, Igor Kabiljo, Brian Karrer, Giuseppe Ottaviano, Sergey Pupyrev, and Alon Shalita. Compressing graphs and indexes with recursive graph bisection. In *International Conference on Knowledge Discovery and Data Mining, KDD*, pages 1535–1544. ACM, 2016. doi:10.1145/2939672.2939862.

3 Alexander Dobler. A note on the complexity of one-sided crossing minimization of trees. *arXiv preprint arXiv:2306.15339*, 2023.

4 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.

5 Vida Dujmovic and Sue Whitesides. An efficient fixed parameter tractable algorithm for 1-sided crossing minimization. *Algorithmica*, 40(1):15–31, 2004. doi:10.1007/S00453-004-1093-2.

6 Peter Eades and Sue Whitesides. Drawing graphs in two layers. *Theor. Comput. Sci.*, 131(2):361–374, 1994. doi:10.1016/0304-3975(94)90179-1.

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

8 Henning Fernau, Fedor V. Fomin, Daniel Lokshtanov, Matthias Mnich, Geevarghese Philip, and Saket Saurabh. Ranking and drawing in subexponential time. In *International Workshop on Combinatorial Algorithms, IWOCA*, volume 6460 of *Lecture Notes in Computer Science*, pages 337–348. Springer, 2010. doi:10.1007/978-3-642-19222-7_34.

9 Ellis Hoag, Kyungwoo Lee, Julian Mestre, Sergey Pupyrev, and Yongkang Zhu. Reordering functions in mobiles apps for reduced size and faster start-up. *ACM Trans. Embed. Comput. Syst.*, 23(4), 2024. doi:10.1145/3660635.

10 Michael Jünger and Petra Mutzel. 2-layer straightline crossing minimization: Performance of exact and heuristic algorithms. *J. Graph Algorithms Appl.*, 1(1):1–25, 1997. doi:10.7155/JGAA.00001.

11 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.

12 Julián Mestre and Sergey Pupyrev. Approximating the minimum logarithmic arrangement problem. In *International Symposium on Algorithms and Computation, ISAAC*, volume 248 of *LIPIcs*, pages 7:1–7:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.ISAAC.2022.7.

13 Xavier Muñoz, Walter Unger, and Imrich Vrto. One sided crossing minimization is NP-hard for sparse graphs. In *International Symposium on Graph Drawing, GD*, volume 2265 of *Lecture Notes in Computer Science*, pages 115–123. Springer, 2001. doi:10.1007/3-540-45848-4_10.

14 Hiroshi Nagamochi. An improved bound on the one-sided minimum crossing number in two-layered drawings. *Discret. Comput. Geom.*, 33(4):569–591, 2005. doi:10.1007/S00454-005-1168-0.