PACE Heuristic Solver Description: UAIC_OCM*

Andrei Arhire
Alexandru Ioan Cuza University of Iași, Romania

Eugen Croitoru
Alexandru Ioan Cuza University of Iași, Romania

Matei Chiriac
Alexandru Ioan Cuza University of Iași, Romania

Alex Dumitrescu
Alexandru Ioan Cuza University of Iași, Romania

Abstract

The One-Sided Crossing Minimization (OCM) problem involves reducing the number of edge crossings in a bipartite graph with a fixed vertex set. This optimization is crucial for applications in areas such as circuit layout, network visualization, and data interpretation, where clear graph layouts enhance usability and understanding. In the PACE 2024 competition, we developed UAIC_OCM, a heuristic solver that combines greedy and local search techniques to address the OCM problem efficiently.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases One-Sided Crossing Minimization, Heuristic Solver, Greedy Algorithm, Local Search, Graph Drawing, Crossing Minimization, Optimization

Supplementary Material The source code is available on Zenodo and GitHub.

1 Introduction

Graph drawing is a fundamental area in computer science and related fields, involving the creation of visual representations of graphs to facilitate understanding and analysis. A particular challenge within this domain is the One-Sided Crossing Minimization (OCM) problem, which focuses on reducing edge crossings in bipartite graphs where one set of vertices is fixed. This problem has significant implications for various applications, including circuit layout design, network visualization, and biological data interpretation, where minimizing visual clutter is essential for clarity and effectiveness.

The OCM problem can be formally described as follows: given a bipartite graph G=(U,V,E) where U is a fixed set of vertices and V is a set of mobile vertices, the objective is to find a permutation of V that minimizes the number of edge crossings when edges E are drawn between U and V. This combinatorial optimization problem is known to be NP-hard, and exact solutions are often impractical for large graphs due to their computational complexity.

In the PACE 2024 competition, participants were invited to develop solvers for the OCM problem that are not only effective in reducing crossings but also efficient enough to handle large instances within reasonable time constraints. Our contribution to this competition is

* STUDENT SUBMISSION

the solver UAIC_OCM, which employs a heuristic approach combining greedy algorithms and local search techniques to iteratively refine the arrangement of the mobile vertices.

The core strategy of UAIC_OCM involves two main operations: a greedy repositioning of vertices and a local search optimization. Initially, vertices are repositioned one at a time to their locally optimal positions in a way that minimizes crossings relative to the fixed set. This is followed by a local search that focuses on optimizing short segments of the vertex sequence, further refining the overall layout. To enhance the effectiveness of these operations, we apply various rearrangement strategies to subsequences, such as reversing the order, sorting by neighboring means, and random shuffling, ensuring diverse exploration of possible configurations.

This paper details the design and implementation of UAIC_OCM, outlining the heuristic techniques employed and their integration into a cohesive algorithm. By balancing computational efficiency with solution quality, our solver provides a practical approach to tackling the OCM problem, demonstrating competitive performance in the PACE 2024 competition. The source code for UAIC_OCM is available for further exploration and development.

2 Algorithm Description

2.1 Component Decomposition

To simplify the problem, we first decompose the graph into components. Each component consists of a subset of the mobile vertices and their associated fixed vertices such that there are no crossings between any two components. This decomposition allows us to independently optimize each component, reducing the computational complexity of the problem. Within each component, we merge mobile vertices that have identical adjacency lists.

2.2 Operations on the Mobile Partition

2.2.1 Greedy Operation

The greedy operation repositions each vertex in the mobile partition to its locally optimal position within the sequence. For each vertex v, we remove it from its current position and reinsert it at the position that minimizes the number of crossings. This process is repeated for every vertex from left to right in the sequence.

2.2.2 Exact Operation

Following the greedy operation, an exact algorithm is performed on short segments of the vertex sequence. This involves finding the optimal arrangement for fixed-length subsequences and reordering them to further minimize crossings. The exact algorithm is applied sequentially to each L-length subsequence in the sequence.

2.2.3 Iteration Process

The solver alternates between the greedy operation and the exact operation, applying them sequentially until no further improvements in crossing reduction are observed. This iterative process ensures that the solution progressively approaches a local minimum for the number of crossings.

2.2.4 Subsequence Optimization

To explore diverse configurations and avoid local optima, we apply various rearrangement strategies to subsequences. For each subsequence, we test the following rearrangements and apply the greedy and exact operations to each configuration, selecting the one that results in the least number of crossings:

- Reverse Order: Invert the order of the vertices in the subsequence.
- Sort by Mean of Neighbors Ascending: Sort vertices based on the average position of their neighbors in ascending order.
- Sort by Mean of Neighbors Descending: Sort vertices based on the average position of their neighbors in descending order.
- Random Shuffle: Randomly shuffle the vertices to explore unbiased configurations.

2.2.5 Convergence and Final Optimization

The solver iteratively applies these optimization strategies, focusing on progressively larger subsequences, within the available time constraints. This hierarchical approach ensures thorough exploration of possible arrangements, balancing the depth of search with computational efficiency. The process continues until no significant improvement can be achieved, providing a near-optimal solution for the OCM problem.