# PACE Exact 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 focuses on rearranging the mobile vertices of a bipartite graph to minimize edge crossings with a fixed vertex set. This optimization is crucial for applications like circuit layout and network visualization, where clear graph representations are essential. In response to the PACE 2024 challenge, we developed UAIC_OCM, an exact solver leveraging a popular Integer Linear Programming (ILP) model and an efficient Branch-and-Bound algorithm.

Our ILP approach [1], which utilizes the HiGHS[2] solver for its high performance and accuracy, adapts a well-known formulation to address specific problem constraints, providing optimal solutions for small to medium-sized graphs. To manage larger graphs and avoid exceeding memory limits, our Branch-and-Bound algorithm incrementally constructs solutions, applying prefix hashing and lower bound calculations to prune infeasible paths early. This combination allows UAIC_OCM to balance accuracy and scalability, delivering competitive performance across various graph sizes and complexities.

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

**Keywords and phrases** One-Sided Crossing Minimization, Exact Solver, Integer Linear Programming, Branch and bound, Graph Drawing, Crossing Minimization

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

## 1 Introduction

Graph drawing is a crucial area in computer science, enabling the creation of visual representations that facilitate understanding and analysis of complex networks. A significant challenge within this domain is the One-Sided Crossing Minimization (OCM) problem, which focuses on reducing edge crossings in bipartite graphs. In this problem, one set of vertices is fixed, while the other set is permutable, and the objective is to find an ordering of the mobile vertices that minimizes the number of crossings between edges.

Formally, 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 goal is to determine a permutation of V that results in the fewest possible edge crossings when edges E are drawn between U and V. This combinatorial

* STUDENT SUBMISSION

optimization problem is known to be NP-hard, making it computationally infeasible to solve exactly for large graphs using straightforward methods.

In response to the PACE 2024 competition, which called for efficient solvers for the OCM problem, we developed UAIC_OCM, an exact solver that combines a well-established Integer Linear Programming (ILP) approach with a tailored Branch-and-Bound algorithm. The ILP solver is adapted to handle specific constraints of the problem, providing exact solutions for small to medium-sized graphs. For larger graphs, where memory constraints limit the effectiveness of the ILP approach, our Branch-and-Bound algorithm constructs the permutation incrementally, employing advanced pruning techniques to maintain efficiency and scalability.

This paper details the design and implementation of UAIC_OCM, describing how it leverages both ILP and Branch-and-Bound methodologies to effectively minimize edge crossings. By integrating these approaches, UAIC_OCM balances computational efficiency with solution accuracy, demonstrating competitive performance across a range of graph sizes and complexities.

## 2 Algorithm Description

This section details the algorithms employed by UAIC_OCM to solve the One-Sided Crossing Minimization (OCM) problem: an adapted Integer Linear Programming (ILP) solver for small to medium-sized graphs, and a Branch-and-Bound algorithm for handling larger and sparser graphs efficiently.

### 2.1 ILP Solver

To solve the One-Sided Crossing Minimization (OCM) problem exactly, we adapted an Integer Linear Programming (ILP) formulation. The ILP model is designed to minimize edge crossings by optimizing the permutation of the mobile vertices relative to a fixed vertex set.

**ILP Formulation:**

$$\min \sum_{\substack{u,v \in V_b \\ u<v}} [c(v,u) + x_{u,v} \cdot (c(u,v) - c(v,u))] \tag{1}$$

**Subject to:**

$$0 \le x_{u,v} + x_{v,w} - x_{u,w} \le 1 \quad \forall u,v,w \in V_b, u<v<w \tag{2}$$

$$x_{u,v} \in \{0,1\} \quad \forall u,v \in V_b, u<v \tag{3}$$

In this formulation: - $c(u,v)$ represents the number of crossings if vertex $u$ is placed before vertex $v$. - $x_{u,v}$ is a binary variable indicating whether vertex $u$ precedes vertex $v$.

**Optimization and Constraints:**

The objective function aggregates the crossing contributions based on the vertex ordering. Constraints ensure the consistency and binary nature of the placement decisions.

In practice, we found that certain vertex pairs $(x,y)$ have a fixed order, meaning vertex $x$ must always precede vertex $y$. This prior knowledge allows us to omit the corresponding constraints for these pairs, reducing the complexity of the ILP model and improving computational efficiency.

The ILP solver is particularly effective for small to medium-sized graphs with intricate crossing patterns. However, it may encounter memory limitations (up to 8 GB) for larger graphs due to the extensive number of constraints.

## 2.2 Branch-and-Bound Algorithm

For larger and sparser graphs where memory constraints limit the feasibility of the ILP solver, we developed a Branch-and-Bound algorithm. This method incrementally constructs the solution permutation while employing advanced pruning techniques to ensure efficiency and scalability.

**Algorithm Overview:**

1. **Incremental Construction:** Vertices are added to the current permutation incrementally, ensuring each addition is locally optimal.
2. **Exact Verification:** For the last few vertices (typically 10), a naive exact solver is used to find the arrangement that minimizes crossings.
3. **Positional Constraints:** Known constraints are respected during the construction (e.g., vertex $x$ must precede vertex $y$).
4. **Pruning:** Prefix hashing and lower bound calculations are used to prune suboptimal branches early in the process.

**Detailed Steps:**

- **Prefix Hashing:** Each prefix is hashed with its minimum crossing value. If a prefix can be rearranged to result in fewer crossings than previously computed, the branch is pruned.
- **Lower Bound Calculation:** A lower bound for the remaining vertices is computed. If this bound plus the current crossings equals or exceeds the best-known solution, the branch is terminated.

**Lower Bound Computation:**

The lower bound is initially computed by summing the minimum crossings for each pair $(x, y)$:

$$\sum \min(\text{cross}(x, y), \text{cross}(y, x)) \tag{4}$$

It is then refined by considering positional relationships. For a pair $(a, b)$ where $a$ precedes $b$, if $\text{cross}(b, a) < \text{cross}(a, b)$, we analyze intermediate vertices to further refine the lower bound.

The Branch-and-Bound algorithm is efficient for large, sparse graphs due to its linear memory usage and effective pruning strategies. While it may not perform as well for dense graphs or instances where the initial heuristic configuration is far from optimal, it excels in handling complex vertex arrangements with fewer computational resources compared to the ILP solver.

## References

1 Michael Jünger and Petra Mutzel. Exact and heuristic algorithms for 2-layer straightline crossing minimization. In *International Symposium on Graph Drawing*, pages 337–348. Springer, 1995.

2 Florian Schwendinger and Dirk Schumacher. *highs: 'HiGHS' Optimization Solver*, 2023. R package version 0.1-10. URL: `https://CRAN.R-project.org/package=highs`.