Local Optimization with Pre-Processing(LOPP): A heuristic solver for one-sided crossing minimization

Arijeet Pramanik
Department of Computer Science and Engineering, IIIT Guwahati, India

Rishabh Dev
Department of Computer Science and Engineering, IIIT Guwahati, India

Vimal Narassimmane
Department of Computer Science and Engineering, IIIT Guwahati, India

Srinibas Swain
Department of Computer Science and Engineering, IIIT Guwahati, India

Abstract

One-sided crossing minimization(OCM) is arranging the nodes of a bipartite graph on two layers (typically horizontal), with one of the layers fixed, and the goal is to minimize the number of edge crossings.

"Local Optimization with Pre-Processing(LOPP)" is a novel heuristic solver for computing the OCM. The solver can be accessed through this **link**[1]. In LOPP, we first pre-process the input graph by eliminating all degree zero vertices. We then use combination of backtracking and local search to identify potential pair of vertices suitable for swapping to minimize the number of crossings. We iteratively apply the aforementioned technique untill no new pairs were discovered.

Keywords and phrases Crossing number, bipartite graph, backtracking, pre-processing

1 LOPP

LOPP's algorithm comprises of two major phases:

- Pre-processing
- Backtracking local search

1.1 Pre-processing

Pre-processing is implemented to make sure the local search algorithm has a good start and does not perform unnecessary checks. We apply the barycenter and median heuristic defined in Eades et al. in [2], choose the ordering that gives lesser crossings, and pass the ordering to the local search algorithm giving it a good initial state. Additionally, we ignore isolated vertices in the free layer as they provide no contribution to the total crossings.

1.2 Backtracking local search

The algorithm is a variant of the sifting algorithm introduced in Matuszewski et al. 1999 [3]. Our idea in this algorithm is to maintain a window size of $k > 0$ in the free layer and perform a swap between the first and the last vertex in the window if and only if it reduces the overall crossings within the window. We start with a window size of 1 and maintain a stride of 1. We use the lemma defined in Eades et al. in [2] and the partially filled crossing matrix

[1] DOI for the code is available at: zenodo

to calculate the total crossings in a window i.e. $\text{cross}(G, x_0, x_1) = \sum_{x_1(u)<x_1(v)} c_{uv}$. The window size is incremented from k to $k+1$ if and only if there are no swaps performed in all windows sized from 1 to k, performing a backtrack search in every step of the local search.

1.2.1 Crossing matrix

Crossing number and crossing matrix are defined in [1]. For any 2 vertices u and v, we can find $cr_{u,v}$ by finding the sum of indices at which neighbors of u should be inserted in the array to maintain sorted order of neighbors of v. Finding $cr_{u,v}$ for any u, v is $O(n\log(m))$ where n is $deg(u)$ and m is $deg(v)$ Crossing matrix is 2D vector where $C[u][v]$ represents $cr_{u,v}$.

2 Algorithm for LOPP

The steps of our algorithm are described as follows:

1. Perform the median and barycenter heuristic [2] and choose the ordering that gives the least crossings.
2. Obtain isolated vertices in the free layer i.e. $v \in B$ such that $deg(v) = 0$ and shift them to the start of the ordering.
3. Set window size $k = 1$ and move to step **4**.
4. For each window of size k in the free layer (ignoring isolated vertices at the start), calculate the number of crossings in the window using the formula mentioned in Section 1.2. This calculation is performed before and after swapping the first and last vertices in the window. If there is a reduction in the number of crossings, keep the new order, else revert back to previous order.
5. Repeat step **4** if there is a suitable swap performed, otherwise set $k = k+1$ and go to step **6**.
6. If any suitable swap is performed during the repeated process of step **4** and **5**, set $k = k-2$ and repeat step **4**, otherwise go to step **7**.
7. If $k < n$ where $n = |B|$, repeat step **4** else **exit**.

References

1 Vida Dujmovic and Sue Whitesides. An efficient fixed parameter tractable algorithm for 1-sided crossing minimization. *Algorithmica*, 40(1):15–31, 2004.

2 Peter Eades and Nicholas C Wormald. Edge crossings in drawings of bipartite graphs. *Algorithmica*, 11:379–403, 1994.

3 Christian Matuszewski, Robby Schönfeld, and Paul Molitor. Using sifting for k -layer straightline crossing minimization. In *International Symposium Graph Drawing and Network Visualization*, 1999. URL: `https://api.semanticscholar.org/CorpusID:6573370`.