PACE 2024 Challenge Report: Heuristic Approaches to One-Sided Crossing Minimization

1. Introduction

Graph drawing is a critical area of study in computational geometry and computer science, dealing with the visualization of graphs in a manner that is both aesthetically pleasing and functionally informative. Among the various problems in this field, the one-sided crossing minimization problem (OCM) presents a unique challenge. It involves arranging the nodes of a bipartite graph across two layers in such a way that the number of edge crossings is minimized. This problem is pivotal for drawing hierarchical graphs, where nodes represent entities and edges represent relationships between these entities. The goal is to produce a layout that is easy to understand, reducing the cognitive load on the viewer.

The Parameterized Algorithms and Computational Experiments Challenge (PACE) provides an annual platform for bringing theoretical ideas from parameterized complexity into practical application. The 2024 iteration of this challenge has selected the OCM problem as its focus, reflecting its significance and the complexity of finding efficient solutions. This problem is NP-hard, signifying that no polynomial-time algorithm is known for finding an optimal solution for all instances. However, the problem admits good heuristics, can be constant-factor approximated, and solved in fixed-parameter tractable (FPT) time for certain parameterizations.

Our participation in the Heuristic track of the PACE 2024 challenge is motivated by the desire to explore and implement heuristic solutions to the OCM problem. This approach is driven by practical considerations: heuristic algorithms can provide sufficiently good solutions within a reasonable time frame, making them highly relevant for real-world applications where exact solutions may be infeasible due to time or resource constraints.

In this report, we introduce three heuristic algorithms: the Median Heuristic, the Split Heuristic, and the Barycenter Heuristic. Each of these has been chosen for its potential to effectively reduce the number of edge crossings in bipartite graphs. The Median Heuristic aims to leverage the median position of nodes to determine an

optimal placement that minimizes crossings. The Split Heuristic, on the other hand, divides the graph into smaller, more manageable sections, applying localized strategies for minimizing crossings within each section. Lastly, the Barycenter Heuristic calculates the center of mass of nodes' positions, using this as a guide to arrange nodes in a way that is likely to reduce edge crossings.

By exploring these algorithms, we aim not only to contribute to the PACE challenge but also to advance the broader field of graph drawing. This report details our methodology, implementation, experimental setup, and the results of our endeavors, providing insights into the effectiveness and limitations of each heuristic approach.

2. Heuristic Track Overview & Literature Review

The Two-layer Crossing Minimization Problem is a well-known NP-hard problem even when the order of vertices in one layer is fixed [1]. This means no polynomial algorithms can actually be found to solve it exactly unless P=NP. All the existing exact algorithms need to take exponential time to give an optimal permutation of vertices that minimizes the number of crossings. So in practice, it could be extremely time-consuming, especially when dealing with large graphs with more than 60 vertices per layer [2]. So, to achieve balance between cost and performance, instead of finding exact algorithms, we turn to choose heuristic algorithms which are much more efficient.

There is a wealth of resources related to solving OCM. Many researchers have studied this NP-hard problem and published their well-performed heuristic algorithms in papers. Among all of them, the three heuristic algorithms we choose to explore in this report are Median, Barycenter and Split.

Barycenter is the most basic and might be one of the earliest heuristic algorithms for solving the two-layer crossing minimization problem. Its popularity maintains and is still one of the most commonly employed approximation algorithms on graphs. The Median is newer and is a variant of Barycenter [1]. They are both called "average heuristics" which compute positions by simply conducting average calculation. Though for the simplicity, they are surprisingly among the heuristic algorithms which often give the most promising results. Iterated barycenter especially performs very well on graphs with respect to the number of edges [4]. Further to Barycenter, Median can ensure the upper bound of its crossings is at most three times of the optimal ones with a solid theoretical proof. Though barycenter often outperforms it in actual practice [1].

Split is another heuristic algorithm first proposed by P. Eades and D. Kelly, which also proves its qualified performance on graphs [5]. It employs a strategy similar to quicksort, utilizing random pivots and recursively conducting pairwise crossing minimization between the pivot and each other vertex. Split often outputs better results compared to other heuristics. However, it requires significantly more time compared to the barycenter and median heuristics, which offer similar quality results [4].

Besides analysis of a single heuristic's performance, many previous studies also have proposed hybrid approaches and tried to combine different heuristics to achieve better results. This is also the direction that we attempt to explore. Greedy Switch algorithm repeatedly adjusts the relative positions of consecutive pairs from left to right to decrease the crossing number. Which is suitable for being incorporated as the post-process step of our chosen heuristics. It also has shown achievement in some hybrid approaches with several heuristics [3]. We try it with barycenter and median in this report to see improvements.

In addition to heuristic algorithms, we also incorporate some efficient methods proposed in some papers of functions we need to implement in our algorithm. In this way we further increase our algorithm's efficiency and reduce the running time in practice. For example, we use the efficient counting DP (Dynamic Programming) algorithm proposed by Nagamochi & Nobuyasu. We use it to count the total crossings in the middle of iteration process, which limits the costing time to $O(n_1 n_2)$ (n_1, n_2, the number of vertices in two layers respectively) [2].

3. Methodology

Our research embarked on an analytical journey to explore heuristic strategies aimed at minimizing edge crossings in bipartite graphs, a challenge central to the field of graph drawing. This section outlines the methodology employed in evaluating the effectiveness of these heuristic algorithms, specifically the Median, Barycenter, and Split Heuristics, and the role of the Greedy Switch technique as a means of post-processing optimization.

Heuristic Algorithm Selection and Application

Median Heuristic: This strategy prioritizes the median position of nodes' adjacent counterparts to guide its placement, aiming to align nodes directly across layers with their median connections, thereby minimizing potential edge crossings. It is proven to be a *3-approximation* algorithm [1]. Time complexity: $O(n_2 log(n_2) + |E|)$, space complexity: $O(n_2 + |E|)$.

Barycenter Heuristic: Focusing on the "center of gravity," this approach calculates a barycenter for each node based on the average position of its connections. Nodes are then arranged to approximate these barycenter positions as closely as possible, striving for a balanced and optimized layout. Time complexity: $O(n_2 log(n_2) + |E|)$, space complexity: $O(n_2 + |E|)$.

Split Heuristic: Inspired by the quicksort algorithm, this method employs a divide-and-conquer strategy. It selects a pivot node and organizes other nodes around it to minimize pairwise crossings, recursively applying this logic to refine the graph's arrangement. Time complexity: worst case $O(n_2^2)$ and average $O((n_2 + |E|)log(n_2))$, space complexity: $O(n_2 + |E|)$

Iterative Optimization of the Split Heuristic

Unique to our methodology is the iterative optimization process applied to the Split Heuristic. Recognizing the potential for further refinement, we adopted a strategy where the heuristic is applied repeatedly, allowing for an incremental adjustment and optimization of node positions with each iteration. This process aimed to progressively reduce the total number of edge crossings through successive approximations towards an local optimal arrangement.

Greedy Switch Post-Processing

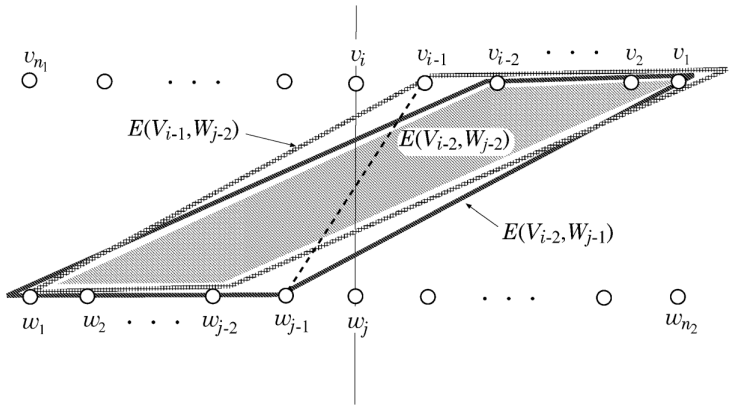
To complement our heuristic algorithms in addressing the one-sided crossing minimization problem for the PACE 2024 challenge, we incorporated a "Greedy Switch" post-processing technique. This method, reminiscent of the bubble sort algorithm, iteratively evaluates and potentially swaps adjacent nodes to reduce edge crossings. In each iteration, the technique traverses the node sequence from left to right, examining pairs of adjacent nodes. If swapping a pair results in fewer crossings, the swap is executed, iteratively refining the graph's layout towards minimization of edge crossings. This process enhances our heuristic solutions by

leveraging local adjustments for incremental improvements, thus proving to be an effective strategy in our experimental framework.

Dynamic Programming algorithm for Counting total Crossings

Inside our implementation of algorithms, we often need to count total crossings based on current ordering of vertices. For example, we have incorporated randomness into the split algorithm because the pivot is always chosen randomly for each recursion. So we decide to run the split algorithms for a limited iteration and find the optimal ordering which achieves the minimum crossing number among all the iterations. It is the strategy that our 'Iterative Split Heuristic' employs. But between each iteration, we need to count the total crossings for the current resulting ordering of vertices, which can significantly increase the time burden on our execution. So an efficient counting algorithm is of great significance.

The counting strategy we employ is a dynamic programming algorithm proposed by Nagamochi & Nobuyasu [2]. It orders the vertices in first layer from right to left and orders the vertices in the second layer from left to right. We set $|E(V_i, W_j)|$ be the number of crossings created by the edge $\{v_i, w_j\}$ (if exists) and all the edges created by vertices v and w with smaller indices that i and j respectively. Each time we can compute $|E(V_i, W_j)|$ in constant time based on previous calculations. After finishing all the calculations, we sum up all the $|E(V_i, W_j)|$ for $1 \le i \le n_1$ and $1 \le j \le n_2$ as our final output. In this way, the time complexity is limited to be $O(n_1 n_2)$.



$$\begin{aligned}&\left|E(V_{i-1}, W_{j-1})\right| \\ &= \left|E(V_{i-1}, W_{j-2})\right| + \left|E(V_{i-2}, W_{j-1})\right| \\ &\quad - \left|E(V_{i-2}, W_{j-2})\right| + \left|E\left(\{v_{i-1}\}, \{w_{j-1}\}\right)\right|. \quad (1)\end{aligned}$$

Fig. 2. Illustration for notations of $E(V_i, W_j)$ in a bipartite graph $G = (V, W, E)$.

4. Experimental Setup and Evaluation Criteria

Software Environment

Our computational experiments were underpinned by Ubuntu 22.04 LTS, chosen for its stability and long-term support, ensuring a reliable platform for our work. Complementing this, SageMath 10.2 offered an advanced suite of tools for mathematical computation, particularly in graph theory and optimization. This combination allowed us to leverage state-of-the-art developments in algorithm implementation and evaluation, providing a solid foundation for our experimental analysis.

Graph Instances and Team Strategy

The "tiny set instances" and "medium size instances" provided by the challenge organizers were instrumental in evaluating our heuristics across different complexity scales. The tiny sets enabled detailed analysis and optimization in simpler scenarios, while the medium sets challenged our algorithms' scalability and effectiveness, offering insights into their applicability to complex real-world problems.

Our team's collaborative effort was strategically segmented among the core heuristic algorithms:

Jack led the Barycenter Heuristic, enhancing its reflection of the graph's connectivity patterns through adaptive weighting, optimizing edge crossing minimization.

Lingyue directed the Median Heuristic, refining median position calculations with edge weights to prioritize significant connections, thus improving performance in densely connected graphs.

Alice oversaw the Split Heuristic, creating a dynamic partitioning algorithm that adjusted based on graph density and node distribution, allowing for tailored heuristic application.

Our experiments spanned a diverse collection of medium-sized bipartite graphs, enabling a thorough assessment of each heuristic's adaptability and efficacy across varied instances. The performance of each heuristic, both pre and post-optimization, was evaluated based on:

- Reduction in Crossings: The primary metric for effectiveness was the decrease in the total number of edge crossings.
- Computational Efficiency: We also considered the computation time as a secondary metric, balancing performance against the practical applicability of each heuristic approach.

Comparative and Correlation Analysis

A comparative analysis was conducted to contextualize the performance of each heuristic relative to the others, incorporating statistical examination to identify trends and outliers. Additionally, we investigated correlations between heuristic performance and specific graph characteristics, aiming to uncover insights that could guide the selection and application of heuristic strategies in future scenarios.

5. Results.

Basic Statistics:

Median and Range: The performance of the algorithms across different graph instances exhibits a wide range, with the number of crossings varying from as low as 240 to as high as nearly 200,000 in the more complex cases.

Except for the Median heuristic, all other algorithms consistently computed the most optimal results on tiny instances.

For median instances,the Barycenter heuristic displayed the highest average percentage error at 6.91%, despite its high structural similarity to the Median heuristic, suggesting that its approach, while similar in design, may be less effective in practical applications. On the other hand, the Median heuristic outperformed the Barycenter with a lower average error of 5.08%, indicating that its simpler strategy of aligning nodes based on median connections could more effectively reduce

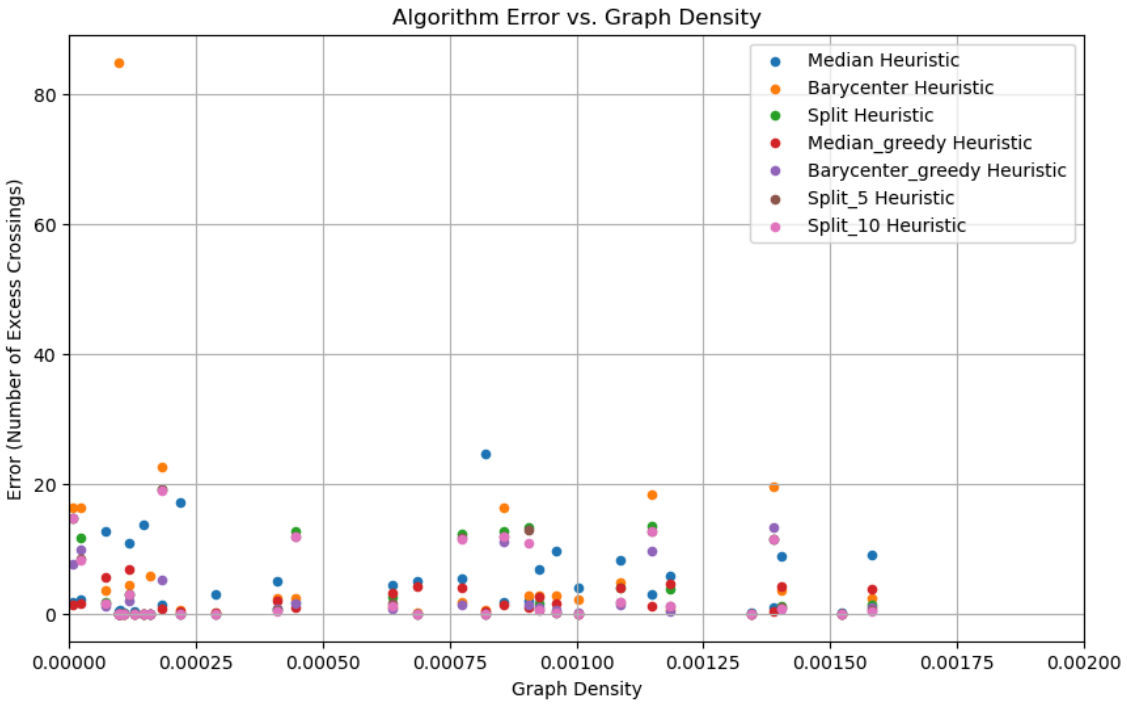
crossings. The most efficient initial performance was observed with the Split heuristic, which had the lowest average error of 4.46%.

Improvement Through Greedy Switch: The mean values of percentage error indicate that the application of the greedy switch post-processing step generally results in a reduction of crossings for both the Median (from 5.08% to 1.73) and Barycenter Heuristics (from 6.91% to 2.07%). This suggests that the greedy switch is effective in further optimizing the solutions. The performance of the Split Heuristic across 5 and 10 iterations shows a very high correlation (0.9999 between Split-5 iteration and Split-10 iteration). The percentage errors are reduced 3.96% and 3.86% correspondingly, indicating that additional iterations beyond a certain point may not yield significant improvements. What's more, the split algorithm is not enhanced by greedy post-processing.The details of our results are put in the Pace 2024 results.xsl for reference.

Insights:

Effectiveness of Greedy Switch: The reduction in crossings achieved by applying the greedy switch suggests it's a valuable post-processing step, particularly for the Median and Barycenter Heuristics. It appears to fine-tune the initial arrangement, smoothing out local irregularities that the broader strokes of the heuristic approaches might miss.

Split Heuristic Saturation: The lack of significant improvement between 5 and 10 iterations of the Split Heuristic suggests there might be a point of diminishing returns. After a certain number of iterations, the algorithm may converge to a solution that cannot be significantly improved by further iterations.



6. Discussion

The Median Heuristic's resilience against outliers contrasts sharply with the Barycenter Heuristic's vulnerability, where outlier nodes can disproportionately skew results. What's more, on some very dense graphs, median heuristic also results in a better result than split algorithm which is limited by local modification. This underscores the importance of understanding graph nuances when selecting a heuristic approach.

In comparative performance, the Split Heuristic, inspired by the quicksort algorithm and characterized by its use of a random pivot, outshined both the Barycenter and Median Heuristics on average . Its strength lies in the balance between execution speed and accuracy, achieved through iterative refinement—a strategy that proved less effective for the Barycenter and Median due to their different operational foundations.

However, when we subjected the outputs of the Median and Barycenter Heuristics to greedy-swap post-processing, the results improved significantly, often surpassing the Split Heuristic's performance. This improvement suggests that while the Median and Barycenter may not initially match the Split Heuristic's efficiency, their simplicity forms a solid base for optimization through greedy swapping. Interestingly, applying the same greedy-swap technique to the Split Heuristic's outputs did not yield notable

improvements, indicating a redundancy due to the inherent optimization present in both the Split approach and the greedy-swap method.

This exploration into heuristic solutions for graph drawing, particularly the nuanced application and combination of heuristics and optimization techniques, highlights a complex yet richly rewarding avenue for research. The synergistic potential between different heuristic strategies and post-processing optimizations promises avenues for achieving near-optimal solutions, suggesting a focus on hybrid approaches for future explorations in the field.

7. Conclusion and Future work

In this paper, we choose three popular heuristics including barycenter, median and split as basis of our algorithms to approximate the optimal result of the one-sided crossing minimization problem (OCM). They are among the heuristics which are proved to show the best performance in practice. After incorporating greedy switch as a post-processing algorithm, barycenter and median may even outperform iterative-split on graphs with large number of vertices. And they retain the advantage of efficiency that are still much faster than split. So we can draw a conclusion that iterative split is suitable for graphs of small and medium size. But when graphs are especially large (e.g. $|V| > 100000$), hybrid approaches of barycenter/median & greedy switch are better.

In the future, there are several potential directions for us to explore further in order to continue enhancing our algorithms. We may try other combinations of heuristics which don’t achieve local optimal structure with greedy switches. And we may explore more relations between features of a graph and performance of different heuristic algorithms, besides the size of the graph. Then we may design a more fine branching algorithm depending on specific features of graphs to achieve an improved approximate result for the OCM problem.

References

[1] P. Eades, N.C. Wormald, Edge crossing in drawing bipartite graphs, Algorithmica 11 (1994) 379–403.

[2] H. Nagamochi, N. Yamada, Counting edge crossings in a 2-layered drawing, Information Processing Letters, Volume 91, Issue 5, 2004, Pages 221-225,

[3] Marti, Rafael & Laguna, Manuel. (2003). Heuristics and Meta-heuristics for 2-layer Straight Line Crossing Minimization.. Discrete Applied Mathematics. 127. 665-678. 10.1016/S0166-218X(02)00397-9.

[4] Jünger, Michael & Mutzel, Petra. (1997). 2-Layer Straightline Crossing Minimization: Performance of Exact and Heuristic Algorithms. Journal of Graph Algorithms and Applications. 1. 10.1142/9789812777638_0001.

[5] P. Eades and D. Kelly. “Heuristics for reducing crossings in 2-layered networks, Ars Combinatoria, vol. 21, 1986, pp. 89-98.

Appendix: Results on tiny set & Part of medium set

Graph	ave_degree	**Solution(Opt)**	median	median/greedy switch	barycenter	barycenter/greedy switch	split	split-5 iteration	split-10 iteration
complete_4_5		**60**	60	60	60	60	60	60	60
cycle_8_shuffled		**4**	5	4	4	4	4	4	4
cycle_8_sorted		**3**	3	3	3	3	3	3	3
grid_9_shuffled		**17**	19	17	17	17	17	17	17
ladder_4_4_shuffled		**11**	14	11	11	11	11	11	11
ladder_4_4_sorted		**3**	5	3	3	3	3	3	3
matching_4_4		**0**	0	0	0	0	0	0	0
path_9_shuffled		**6**	7	6	6	6	6	6	6
path_9_sorted		**0**	0	0	0	0	0	0	0
plane_5_6		**0**	0	0	0	0	0	0	0
star_6		**0**	0	0	0	0	0	0	0
tree_6_10		**13**	15	13	13	13	13	13	13
website_20		**17**	17	17	17	17	17	17	17

Part of Mediu	ave_degree	**Solution(Opt)**	median	median/greedy switch	barycenter	barycenter/greedy switch	split	split-5 iteration	split-10 iteration

m size Graphs									
1.gr	1.999	**240**	240	240	254	240	240	240	240
2.gr	1.999	**650**	650	650	1202	650	650	650	650
3.gr	1.999	**489**	513	509	490	489	489	489	489
4.gr	2.076	**3341**	3472	3341	3412	3345	3341	3341	3341
5.gr	2.035	**11450**	11783	11470	11471	11450	11451	11451	11451
6.gr	2.086	**3141**	3233	3175	3719	3442	3562	3541	3541
7.gr	2.006	**6641**	6732	6690	8145	6993	7919	7915	7904
8.gr	2.095	**16859**	17035	16929	20153	19103	18792	18783	18783
9.gr	1.999	**24661**	25195	25061	28698	27102	27532	26765	26715
10.gr	1.999	**20653**	21032	20951	24009	22248	23680	23676	23676
11.gr	2.524	**34126**	34737	34575	39682	37887	38441	38164	38194
12.gr	2.518	**6236**	6569	6487	6347	6320	6997	6960	6948
13.gr	2.496	**61515**	64228	63526	62161	62003	63101	62477	62235
14.gr	18.223	**189865**	199295	197431	191682	191275	192461	192086	192086
15,gr	31.378	**4434**	4540	4479	4512	4478	5036	4927	4927
16.gr	1.999	**8817**	8998	8900	9058	8932	9989	9951	9770
17.gr	1.999	**17373**	17665	17532	17774	17644	19564	19419	19419
18.gr	1.999	**6858**	7198	6996	7018	6904	6906	6901	6887
19.gr	3.91	**6958**	7426	7139	7150	7020	7064	7001	7001
20.gr	1.998	**8572**	9351	8890	8771	8661	8687	8611	8601
21.gr	1.998	**1828**	1933	1911	1839	1833	1898	1850	1850
22.gr	3	**1168**	1272	1218	1211	1177	1182	1178	1177

23.gr	3	**1328**	1455	1349	1366	1335	1329	1329	1329
24.gr	3	**353**	440	354	355	353	353	353	353
25.gr	2.987	**759**	889	762	763	759	759	759	759
26.gr	2.988	**601**	683	601	601	601	601	601	601
27.gr	1.989	**1106**	1110	1106	1106	1106	1106	1106	1106
28.gr	1.995	**2485**	2494	2485	2485	2485	2485	2485	2485
29.gr	1.999	**8546**	8567	8546	8546	8546	8546	8546	8546
30.gr	1.999	**3790**	3809	3790	3790	3790	3790	3790	3790
31.gr	2	**12216**	12240	12216	12216	12216	12216	12216	12216
32.gr	2	**16792**	16826	16792	16792	16792	16792	16792	16792
33.gr	2	**7064**	7640	7339	7404	7163	7347	7183	7183
34.gr	1.765	**23072**	25588	24639	24098	23545	23766	23766	23766
35.gr	1.764	**16917**	19070	17862	17526	17126	17215	17198	17174