Approximation Methods for the Minimum Feedback Arc Set

by
Kevin Zhuo

Professor Shikha Singh, Advisor

Indepdent Study for
Computer Science

WILLIAMS COLLEGE
Williamstown, Massachusetts
December 6, 2024

Contents

Chapter 1

Introduction

1.1 Problem Overview

The feedback arc set (FAS) problem in a directed graph G is a subset of the edges that includes at least one edge from every cycle in G. A feedback arc set with the minimum possible number of edges is referred to as the minimum feedback arc set. By removing this set from G, the resulting graph becomes a directed acyclic graph (DAG), enabling the application of topological sorting to derive a linear ordering of the vertices. This problem has numerous practical applications, including determining rankings in round-robin tournaments[1] and aggregating preferences in ranked voting systems[2]. Moreover, with the increasing prevalence of social networks in modern society, the ability to produce acyclic graphs that closely approximate the structure of large-scale networks holds significant potential for improving computational efficiency in solving related problems[3].

1.2 Accomplishments

In this independent study, we investigated approximation methods for the minimum feedback arc set problem, focusing on achieving a balance between minimizing the size of the feedback arc set and maintaining a relatively fast runtime. By evaluating a range of approaches, we found that introducing a "warm-start" through an initial greedy ordering of the graph provided a preliminary ordering in $O(m + n)$ time. We were then able to construct a "sorted order" of the vertices from which we could obtain a feedback arc set that effectively optimized the trade-off between runtime efficiency and the size of the feedback arc set.

Chapter 2

Background

We will begin by introducing the Minimum Feedback Arc Set problem, followed by an overview of prior work addressing this problem, and concluding with a detailed discussion of the algorithms employed in this independent study.

2.1 Feedback Arc Set Background

As mentioned in the introduction, the Minimum Feedback Arc Set is the smallest subset of edges in a directed graph G such that removing this subset from G results in a DAG. The problem's significance lies in its broad applicability in areas such as ranking systems, preference aggregation, and the analysis of network structures. For example in round-robin tournaments[1], the outcomes of each game can be represented by constructing a directed graph where an edge is drawn from the loser to the winner of each match. By identifying a minimum feedback arc set, reversing the directions of its edges, and performing a topological sort on the resulting graph, it is possible to generate a ranking of all competitors that minimizes the total number of upsets. Another example is in the context of large scale social networks[3], where cyclic relationships can make it difficult to rank users based on metrics like influence and trust. Similar to the round-robin problem, transforming the social network into a DAG can enable the establishment of clear hierarchies and consistent global rankings.

The Minimum-FAS problem was demonstrated to be NP-complete by Karp [4]. For some subsets of the problem such as planar graphs, weakly acyclic graphs, and reducible flow graphs, the Minimum-FAS problem is able to be solved in polynomial time. However, there are important subsets of the problem such as tournament graphs where the problem still remains NP-Complete. Furthermore, the Minimum-FAS problem is also approximation resistant. Unless P = NP, the problem is unable to be approximated to a constant factor in polynomial time. Instead, it is shown that the feedback arc set can be approximated to within a polylogarithmic approximation ratio of $O(\log n \log \log n)$ [5].

2.2 State-of-the-Art Approaches

2.2.1 Integer Programming Techniques

Given the computational complexity of determining the exact minimum feedback arc set, some solutions to this problem have relied on **integer programming techniques**. Bahrev et al. [6] proposed an exact method of finding the minimum feedback arc set problem for sparse graphs that relies on enumerating cycles in a lazy fashion. Their approach begins with the application of a greedy heuristic to identify an initial feedback arc set. Breadth-First Search (BFS) is then used to locate the smallest simple cycles that include this feedback arc set. The cycle matrix is then extended with all the simple cycles that were found. After all the simple cycles are enumerated and the cycle matrix is complete, the integer program with the complete cycle matrix can be fed to a general-purpose integer programming solver.

2.2.2 Approximation Algorithms

Other approaches have relied on **approximation algorithms** which aim to reduce the algorithmic runtime at the cost of a potentially larger feedback arc set. The Minimum-FAS problem is approximation resistant, which implies that for every $C > 0$, it is NP-Hard to find a C-approximation to the problem. Despite this limitation, several attempts have been made to address the problem by forgoing guarantees on the feedback arc set size within approximation algorithms. Simpson et al. [3] presents methods that adopt this strategy, introducing algorithms that lack the guarantee on the size-approximation of the resulting feedback arc set but are both computationally efficient and scalable, making them particularly well-suited for large-scale datasets.

2.2.3 Depth-First Search

An alternative method for detecting cycles in graphs which can be extended to identify a feedback arc set involves employing **Depth-First Search (DFS)** and incorporating all back edges encountered during the traversal into the feedback arc set. Park et al. [7] introduced modifications to the original graph along with a cutting technique based on DFS to develop an efficient method for calculating the feedback arc set. This approach achieves a 90% reduction in the size of the resulting feedback arc set, compared to the 50% reduction using prior methods mentioned in the paper. There has also been prior research into using machine learning methods in order to “warm-start” the DFS and begin off with a set of learned predictions for the ordering [8]. By utilizing predictions about the input data, these algorithms can potentially achieve faster runtimes compared to traditional approaches. Instead of depending solely on the worst-case analysis, the performance of such algorithms is influenced by the accuracy of the predictions, which can lead to significant reductions in computational complexity when the predictions are reliable.

2.3 Algorithms Tested

Initially, we tested a DFS approach that starts from a randomly selected root vertex and builds the feedback arc set by visiting the neighbors of each vertex in random order and keeping track of all backward edges encountered during the traversal. We then developed an optimized version of the initial DFS that prioritizes vertices with the smallest total-degree (in-degree minus out-degree) to visit first during the traversal, while continuously updating the total-degree counter of vertices after traversing through them. Other than DFS, we drew inspiration from Simpson et al., examining the GreedyFAS and SortFAS algorithms mentioned in the paper. Subsequently, we explored an enhanced SortFAS that was "warm-started" with a preliminary ordering derived from GreedyFAS.

The first DFS algorithm implementation employs recursion, starting from a randomly selected vertex and traversing the graph by recursively calling the DFS function on each of its neighbors that is chosen at random without replacement. We maintain an array representing a "coloring" of each vertex, where the indices correspond to vertex numbers and the values are mapped to 0 (vertex has not been seen), 1 (vertex is currently being explored), or 2 (vertex has been fully visited). Back edges are detected when we encounter a vertex with a color of 1. Building upon this approach, the optimized DFS begins at the vertex with the smallest total degree and visits its neighbors in the sequential order based on their total degrees. Similar to the initial method, we maintain the vertex coloring array to detect back edges in the same manner. Due to memory constraints, this implementation is performed iteratively using a stack rather than recursively. In both DFS algorithms, if the resulting feedback arc set comprises more than half of the total edges in the original graph, we can instead return the complement of the feedback arc set to achieve a smaller set. The runtime and space for both algorithms is $O(m+n)$, and there is an approximation guarantee of $\frac{1}{2}|E|$.

GreedyFAS was introduced by Eades et. al [9] as an efficient greedy approximation algorithm for the feedback arc set problem. In every iteration, the algorithm removes vertices from the graph that are sources/sinks, appending source vertices to a sequence $s1$ and prepending sink vertices to a sequence $s2$. Furthermore, the vertex with the largest out-degree - in-degree is also removed from the graph and appended/prepended to $s1$ or $s2$ depending on its status as a source or sink. Once all the vertices in the original graph are removed, the sequence $s = s1s2$ is a linear sequence containing all the vertices in the original graph where the back edges make up a feedback arc set. The greedy approach of the algorithm relies on the intuition that the back edges in a linear sequence of vertices is more likely to be minimized if the vertices most closely resembling source vertices are visited before the vertices most closely resembling sink vertices. For the implementation of GreedyFAS, three arrays are used that denote the vertices that are sources, sinks, and neither, with the arrays being updated whenever vertices are moved. The runtime and space of GreedyFAS is $O(m+n)$, and there is an approximation guarantee of $\frac{1}{2}|E| - \frac{1}{6}|V|$.

SortFAS was introduced by Brandenburg et. al [10] and extends sorting algorithms to the Minimum-FAS problem by applying a sorting heuristic to an initial linear sequence of the vertices and outputting a “sorted” linear sequence of the vertices. The implementation that was explored in the independent study was an insertion sort version of the algorithm. The vertices in SortFAS are processed according to an initial orderin; in the i-th iteration, v_i is placed in the most optimal position within the already sorted subset of the first $i-1$ vertices. If v_i encounters a tie, it is placed at the leftmost position of the other tied vertices. The algorithm determines the optimal position for a vertex as the one that minimizes the number of back edges created when v_i is added to the sequence. SortFAS is able to determine the optimal position of v_i with a single pass using a counter variable to keep track of the number of backward edges induced or removed by a vertex at a location. The optimal location would be the index where the minimum value was achieved by the counter variable. The runtime of SortFAS is $O(n^2)$ with this approach and the space of SortFAS is $O(1)$.

Since SortFAS requires a linear sequence of vertices to start off with, we explored a potential improvement to the algorithm by having the initial linear sequence be the output of GreedyFAS. Given that SortFAS has the limitation where in the i-th iteration the vertex v_i only considers the first $i-1$ vertices, an initial ordering that is already ”close” to the optimal can lead to a better final result and also reduce the number of comparisons performed. By combining the two algorithms, we believe that this approach not only enhances computational efficiency but may also result in a smaller feedback arc set size.

Chapter 3

Results

We will begin by detailing the experimental setup including any datasets that were used. Afterwards, a comprehensive analysis of our results will be provided, concluding with our main takeaways and suggestions for future work.

3.1 Experiment

For the datasets used in the experimentation, we used a set of real-world large social networks as well as randomly generated synthetic graphs. The real-world datasets were taken from SNAP (Stanford Large Dataset Collection) and consists of two snapshots of the Gnutella peer-to-peer file sharing network. We also used a social network that represents the facebook pages with links to each other and a larger social network of the general consumer review site EOpinions where vetices are sites and a directed edge represents that one site "trusts" another site. The synthetic graphs were created by defining a vertex count and assigning each vertex a randomly generated threshold value between 0.01 and 0.3. Each vertex was then connected to every other node in the graph with a probability equal to its threshold value. Table 1 shows the statistics of the graphs that were tested.

Name	$\lvert V \rvert$	$\lvert E \rvert$
Gnutella-small	6301	20777
Gnutella-large	62586	147878
Facebook Page-Page	22470	171002
EOpinions	75879	508837
Random Graph 1	100	1888
Random Graph 2	300	14027

Table 3.1: Graph Dataset Statistics

All graphs used in the experiments were stored as adjacency lists. Since the implementation was primarily done in Python, these adjacency lists were represented as dictionaries where each key corresponds to a vertex and the associated value was a list of vertices to which the key vertex had directed edges to.

3.2 Results

The following results illustrate the performance among the algorithms tested on the numerous graph datasets. Due to some of the graphs being disconnected, it was not possible to obtain the DFS results for all the datasets.

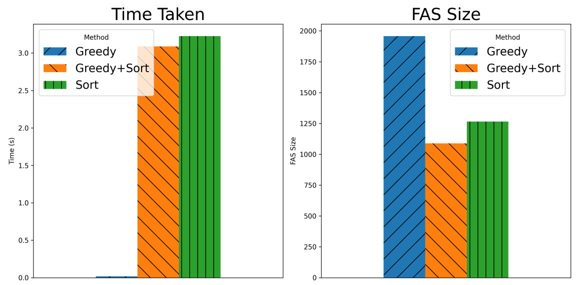

Figure 3.1: GNutella Small

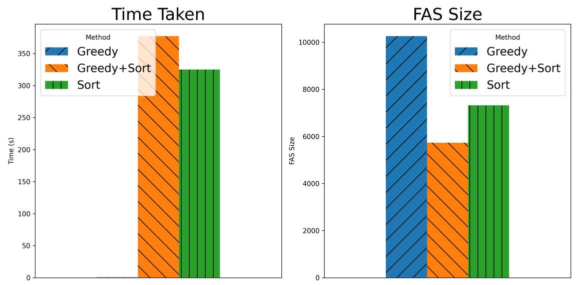

Figure 3.2: GNutella Large

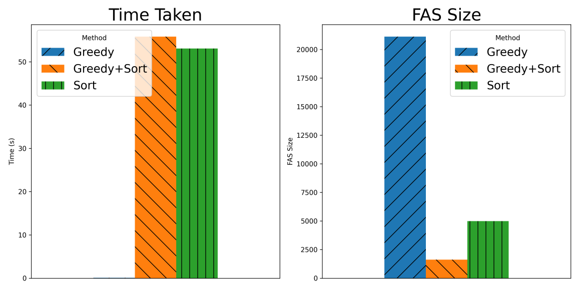

Figure 3.3: Facebook

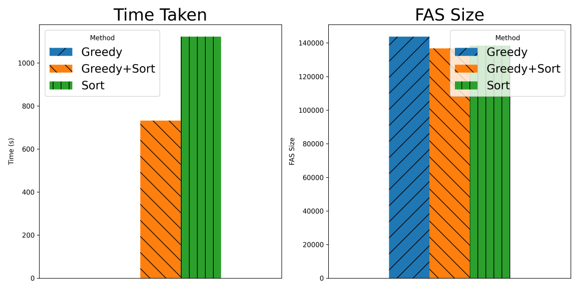

Figure 3.4: EOpinions

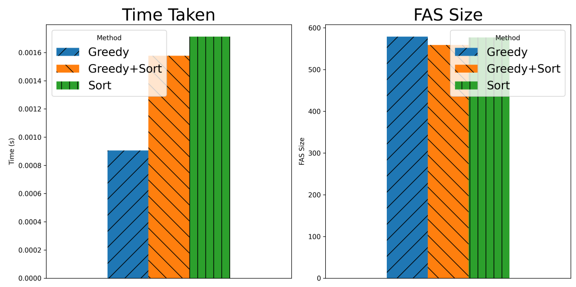

Figure 3.5: Random Graph 1

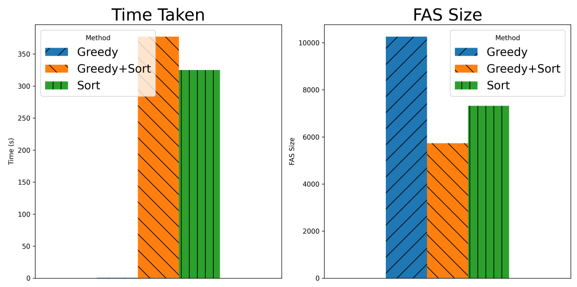

Figure 3.6: Random Graph 2

Name	Sort	Greedy+Sort
Gnutella-small	15509910	9834898
Gnutella-large	1710160533	1066316958
Facebook Page-Page	84888193	108525026
EOpinions	2280853611	1925976892
Random Graph 1	4278	2412
Random Graph 2	30941	13899

Table 3.2: Comparisons Done Per Algorithm

We can see that from the results that the GreedyFAS + SortFAS combined algorithm results in the smallest feedback arc size in the datasets that were tested. As Simpson et. al notes, GreedyFAS performs well when there are many source and sink-like vertices present in the original graph. They also note that unlike traditional sorting problems where there is a total ordering on the data, there is a lack of transitivity on the Minimum-FAS problem which causes sorting trouble on sparse graphs where in every iteration many vertices will be deemed "equal", which can lead to problems arising from the fact that large sections of the ordering will not be modified in any meaningful way. However, both GreedyFAS and SortFAS achieved relatively similar results in the datasets that we tested on, other than the Facebook Page-Page dataset. However, across the board the warm-started SortFAS that starts with the GreedyFAS ordering results in the smallest FAS size for all the different experiments.

In terms of the overall runtime, GreedyFAS outperformed the combined GreedyFAS + SortFAS as well as just SortFAS by itself, which makes sense because GreedyFAS has a runtime of $O(n + m)$ while both SortFAS and the combined version of the two algorithms have a runtime of $O(n^2)$. This also validates the results of Simpson et. al, since they also observed that sorting approaches took longer than the GreedyFAS approach. When comparing the warm-started SortFAS to the naive SortFAS, we observe that, overall, warm-starting SortFAS generally results in a reduced number of comparisons during the sorting process. Even though the runtime in seconds for the naive SortFAS may appear lower than warm-started SortFAS, that may be explained by external factors such as network speed, since the overall number of comparisons is generally lower for the warm-started SortFAS. However, the beginning order does matter which can be seen through Facebook Page-Page GreedyFAS ordering being not ideal, which led to the warm-started SortFAS performing more comparisons than the naive SortFAS. This can be explained by the fact that GreedyFAS algorithm itself produced a very non-optimal result on that dataset, which caused more comparisons to be done.

3.3 Conclusion

In conclusion, the independent study analyzed and experimented with various algorithms for the Minimum-FAS problem. For analysis, we mainly looked into the approximation algorithms of DFS, GreedyFAS, SortFAS, and a new approach of "warm-starting" SortFAS with a linear ordering resulting from GreedyFAS. For experimentation, DFS had some issues with unconnected graphs and memory, so most of the experimentation was focused on GreedyFAS, SortFAS, and the warm-started SortFAS. Our results found that the warm-started SortFAS generally produces the smallest FAS size on a variety of graphs, but it is highly dependent on the intial ordering produced by GreedyFAS. Meanwhile, we observed that SortFAS generally produces slightly better results than GreedyFAS, but having the initial linear ordering of the vertices in SortFAS being the output of GreedyFAS yields even better results and takes fewer comparisons than a naive SortFAS. The experimental results we produced matched the observations of the Simpson et. al paper, with GreedyFAS being an algorithm that is a good balance of scalability and solution quality, while SortFAS is less scalable but has the potential to produce smaller FAS sizes as the output. As mentioned, GreedyFAS tends to perform better on graphs with many source and sink-like vertices, while SortFAS performs better on graphs that are denser. By introducing the "warm-start" method and combining both GreedyFAS and SortFAS, we hope to mitigate some of the shortfalls of the individual methods and present an algorithm that is both scalable and has a high solution quality.

Some future work to be done in this area of research would be to introduce a machine-learning approach to achieve an even better "warm-start" for SortFAS. Since the algorithm is highly susceptible to the initial ordering that is provided, having an ordering that would mitigate some of the transitivity issues experienced with the SortFAS technique would allow the algorithm to make more meaningful decisions with vertices at every step.

Bibliography

[1] L. Hubert, British Journal of Mathematical and Statistical Psychology **29**, 32 (1976), https://bpspsychub.onlinelibrary.wiley.com/doi/pdf/10.1111/j.2044-8317.1976.tb00701.x, URL https://bpspsychub.onlinelibrary.wiley.com/doi/abs/10.1111/j.2044-8317.1976.tb00701.x.

[2] J. G. Kemeny, Daedalus **88**, 577 (1959), ISSN 00115266, URL http://www.jstor.org/stable/20026529.

[3] M. Simpson, V. Srinivasan, and A. Thomo, Proc. VLDB Endow. **10**, 133–144 (2016), ISSN 2150-8097, URL https://doi.org/10.14778/3021924.3021930.

[4] R. Karp (1972), vol. 40, pp. 85–103, ISBN 978-3-540-68274-5.

[5] P. D. Seymour, Combinatorica **15**, 281 (1995), URL https://doi.org/10.1007/BF01200760.

[6] A. Baharev, H. Schichl, A. Neumaier, and T. Achterberg, ACM J. Exp. Algorithmics **26** (2021), ISSN 1084-6654, URL https://doi.org/10.1145/3446429.

[7] S. Park and S. Akers, in *[Proceedings] 1992 IEEE International Symposium on Circuits and Systems* (1992), vol. 4, pp. 1863–1866 vol.4.

[8] S. McCauley, B. Moseley, A. Niaparast, and S. Singh, *Incremental topological ordering and cycle detection with predictions* (2024), 2402.11028, URL https://arxiv.org/abs/2402.11028.

[9] P. Eades, X. Lin, and W. Smyth, Information Processing Letters **47**, 319 (1993), ISSN 0020-0190, URL https://www.sciencedirect.com/science/article/pii/002001909390079O.

[10] F.-J. Brandenburg and K. Hanauer (2011), URL https://api.semanticscholar.org/CorpusID:27641780.