Journal of Graph Algorithms and Applications
http://jgaa.info/ vol. 27, no. 8, pp. 737–757 (2023)
DOI: 10.7155/jgaa.00641

Effective Computation of a Feedback Arc Set Using PageRank

Vasileios Geladaris[1] *Panagiotis Lionakis*[1] *Ioannis G. Tollis*[1]

[1]Computer Science Department
University of Crete, GREECE

Submitted: November 2022 Reviewed: April 2023 Revised: June 2023
Reviewed: August 2023 Revised: September 2023 Accepted: October 2023
Final: November 2023 Published: November 2023
Article type: Regular paper
Communicated by: P. Angelini and R. von Hanxleden

Abstract. Computing a minimum Feedback Arc Set (FAS) is important for visualizing directed graphs in hierarchical style. It is the first step of both known frameworks for hierarchical graph drawing of directed graphs and it is NP-hard. We present a new heuristic algorithm for computing a minimum FAS in directed graphs. The new technique produces solutions that, for graph drawing datasets, are better than the ones produced by the best previously known heuristics, often reducing the FAS size by more than 50%. The heuristic is based on computing the PageRank score of the nodes of the directed line graph of the input directed graph. Although the time required by our heuristic is heavily influenced by the size of the produced line graph, our experimental results show that it runs very fast even for very large graphs used in graph drawing. We compare results produced by our heuristic to known exact results for specific graphs used in a previous study and discuss the interesting trade-off. Finally, our experimental results on large web-graphs show that our technique found smaller FAS than it was known before for some web-graphs from a data set used in a recent study.

1 Introduction

Hierarchical graphs are important for many applications in several areas of research and business because they often represent hierarchical relationships between objects in a structure. They are

Special Issue on the 30th Int. Symposium on Graph Drawing and Network Visualization, GD 2022

E-mail addresses: csd3926@csd.uoc.gr (Vasileios Geladaris) lionakis@csd.uoc.gr (Panagiotis Lionakis) tollis@csd.uoc.gr (Ioannis G. Tollis)

typically directed (sometimes acyclic) graphs and their visualization has received significant attention [10, 15]. In a directed graph G, a Feedback Arc Set (FAS) is a set of edges whose removal leaves G acyclic. Computing a minimum FAS is important for visualizing directed graphs in hierarchical style [10, 15]. In fact, the first step of both known frameworks for hierarchical graph drawing is to compute a minimum FAS [16, 21]. Unfortunately, computing a minimum FAS is NP-hard and thus many heuristics have been presented in order to find a reasonably good solution, see for example [10, 15]. In this paper we present a new heuristic that uses a different approach and produces FAS, for graph drawing datasets, that contain about half the number of edges of the best previously known heuristics. Finding a minimum FAS has many additional applications beyond Graph Drawing, including misinformation removal, label propagation, and many application domains motivated by Social Network Analysis [9, 12, 19]. The new technique requires super-linear time, and hence it may not be suitable for computing FAS in very large social networks.

In hierarchical drawing algorithms the edges in a FAS are not removed, but instead their direction is inverted. Following the terminology of [10], a set of edges whose reversal makes the digraph acyclic is called a feedback set (FS). Notice that a FAS is not always a FS. For example, if all the edges of a cycle are in a FAS, when their direction is reversed then a new cycle is introduced. However, it is easy to see that every minimal cardinality FAS is also a FS. Hence it follows that the minimum FS problem is as hard as the well studied minimum FAS problem which is known to be NP-hard [13, 14]. Additionally, any heuristic for the minimum FAS problem based on finding a minimal FAS can be applied for solving the minimum FS problem, as discussed in [10, 15].

There are many heuristic algorithms in the literature for solving the FAS problem due to the multitude of its applications and its usefulness in many practical applications in different areas. An example of such a heuristic is *SimpleFAS* which utilizes a straightforward 2-approximation algorithm for solving the Maximum Acyclic Subgraph (MAS) problem. The time complexity of SimpleFAS is $O(n + m)$. Berger & Shor [4] presented a better heuristic which is also based on an approximation algorithm for the MAS problem that runs in in $O(n + m)$ time. The intuition behind this approach is that selecting either the incoming or the outgoing edges guarantees that the resulting graph is acyclic. Additionally, by selecting, at each step, the set of edges of bigger size, ensures that the resulting acyclic graph has a large number of edges, see also [20]. Clearly, a simple Depth-first search (DFS) traversal, can be used to compute a feedback arc set. The removal of all the back edges computed by any DFS removes all cycles, thus the resulting graph is acyclic. The time complexity of *dfsFAS* is clearly $O(n + m)$. Eades, Lin & Smyth [11] presented another $O(n+m)$ heuristic algorithm that calculates in a greedy manner a feedback arc set of a graph G, by first calculating a Linear Arrangement of the nodes of G, this is called *GreedyFAS*. *KwikSortFAS*, is a sorting-based heuristic originally introduced by Ailon et al. [1], as a 3-approximation algorithm for the FAS problem on tournaments. It is based on the classical sorting algorithm QuickSort and was later extended by Brandenburg and Hanauer [7] as a heuristic for general directed graphs, see also [20]. This new variant, which is called *SortFAS*, is based on insertion sort and its time complexity is $O(n^2)$. Based on the aforementioned heuristics, Simpson, Srinivasan & Thomo published an extensive experimental study for the FAS problem on large and very large graphs at web-scale (also called *web-graphs*) [20]. According to their study, the most efficient heuristics in terms of quality of solution and execution time are *GreedyFAS* and *SortFAS*, but due to its time complexity, only *GreedyFAS* is suitable to run on their extra large web-graphs. Our technique computes solutions that are superior to both, but since it requires slightly super-linear time, its execution time is higher than *GreedyFAS* but lower than *SortFAS*.

A novel exact method for solving the minimum FAS problem was recently proposed by Baharev et. al. in [2]. In their paper the authors describe extensively the problem, its complexity, and its applications to chemical engineering. Their proposed method uses Integer Linear Programming and enumerates simple cycles in a lazy fashion by extending an incomplete cycle matrix iteratively. Their method succeeds in finding a minimum FAS for several small and medium-sparse graphs, as shown by their extensive experiments. However, as it is expected, it fails to produce results for medium-dense graphs, and even for sparse larger graphs [2].

In this paper we present a new heuristic algorithm for computing a minimum FAS in directed graphs. The new technique produces solutions that are better than the ones produced by the best previous heuristics, sometimes even reducing the FAS size by more than 50%, for graph drawing datasets. It is based on computing the PageRank score of the nodes of a graph related to the input graph, and runs rather fast for graphs up to 4,000 nodes. However, it is slower than GreedyFAS for web-graphs. We also performed experiments using the graphs that required the highest execution time in [2] and compare the results produced by our heuristic to the exact results reported there. There is an interesting trade-off between computing the minimum FAS in excessive time or obtaining a slightly suboptimal solution in a few tenths of a second. Furthermore, our experimental results on the web-graphs of [20] show that our technique found smaller FAS than it was known before for three of them. For their larger web-graphs our technique did not finish in a reasonable amount of time (timeout).

2 Existing Algorithms

In this section we summarize and give a brief description of the two important heuristics that currently give the best results for the FAS problem, according to the new experimental study of Simpson, Srinivasan & Thomo [20]. They implemented and compared many heuristics for FAS, and performed experiments on several large and very large web-graphs.

Their experimental results show that two of the most important and practical heuristics/techniques are due to Eades, Lin & Smyth [11] and Brandenburg & Hanauer [7]. The first of the two heuristic algorithms that currently produce the best FAS size is called *GreedyFAS*. In [20] two different optimized implementations of GreedyFAS that run in $O(n + m)$ are presented and tested. These are the most efficient implementations in their study and are able to run even for their extra large web-graphs. The second algorithm is *SortFAS* of Brandenburg & Hanauer [7]. According to [20], SortFAS, as proposed runs in $O(n^3)$ time but Simpson et al. present an optimized implementation that runs in $O(n^2)$ time.

We will present experimental results that show that our new heuristic algorithm performs better than both of them in terms of the size of the produced FAS. On the other hand, it takes more time than GreedyFAS for large web-graphs. However, for graphs that are typically used in visualization applications (in terms of size), the running time is about the same whereas the produced FAS size is about half.

2.1 GreedyFAS

As discussed above, the GreedyFAS algorithm was introduced by Eades, Lin & Smyth in 1993 [11]. It efficiently calculates an approximation to the FAS problem on a graph G. In order to understand the algorithm, we first discuss the *Linear Arrangement Problem (LA)*, which is an equivalent formulation to the FAS problem. The LA problem produces an ordering of the nodes of a graph G for which the number of arcs pointing backwards is minimum. The set of backwards arcs is a FAS since removing them from G leaves the graph acyclic.

GreedyFAS calculates a feedback arc set of a graph G by first calculating a Linear Arrangement of G. More specifically, in each iteration, the algorithm removes all nodes of G that are sinks followed by all the nodes that are sources. It then removes a node u for which $\delta(u) = d^+(u) - d^-(u)$ is a maximum, where $d^+(u)$ denotes the out-degree of u and $d^-(u)$ denotes the in-degree of u. The algorithm also makes use of two sequences of nodes s_1 and s_2. When any node u is removed from G then it is either prepended to s_2 if it is a sink, or appended to s_1 if it is not. The above steps are repeated until G is left with no nodes, then the sequence $s = s_1 s_2$ is returned as a linear arrangement for which the backward arcs make up a feedback arc set. For more details see [10, 15]. Using the implementations of [20], GreedyFAS runs very fast, in $O(n + m)$ time, and is suitable for their extra large web-graphs. The pseudocode for GreedyFAS, as described in [10] and [20], is presented in Algorithm 1.

Algorithm 1 GreedyFAS

```
Input: Directed graph $G = (V, E)$
Output: Linear Arrangement A
$s_1 \leftarrow \emptyset$, $s_2 \leftarrow \emptyset$
while $G \neq \emptyset$ do
    while $G$ contains a sink do
        Choose a sink $u$
        $s_2 \leftarrow u s_2$
        $G \leftarrow G \backslash u$
    while $G$ contains a source do
        Choose a source $u$
        $s_1 \leftarrow s_1 u$
        $G \leftarrow G \backslash u$
    Choose a node $u$ for which $\delta(u)$ is a maximum
    $s_1 \leftarrow s_1 u$
    $G \leftarrow G \backslash u$
return $s = s_1 s_2$
```

2.2 SortFAS

The SortFAS algorithm was introduced in 2011 by Brandenburg & Hanauer [7]. The algorithm is an extension of the KwikSortFAS heuristic by Ailon et al. [1], which is an approximation algorithm for the FAS problem on tournaments. With SortFAS, Brandenburg & Hanauer extended the above heuristic to work for general directed graphs. It uses the underlying idea that the nodes of a graph can be sorted into a desirable Linear Arrangement based on the number of back arcs induced.

In the case of SortFAS, the nodes are processed in order of their ordering $(v_1 ... v_n)$. The algorithm goes through n iterations. In the the i-th iteration, node v_i is inserted into the linear arrangement in the best position based on the first $i - 1$ nodes which are already placed. The best position is the one with the least number of back arcs induced by v_i. In case of a tie the leftmost position is taken. Using the implementation of [20], SortFAS runs in $O(n^2)$ time. The pseudocode for SortFAS, as described in [20], is presented in Algorithm 2.

Algorithm 2 SortFAS

```
Input: Linear arrangement A
for each node $v$ in $A$ do
    $val \leftarrow 0$, $min \leftarrow 0$, $loc \leftarrow$ position of $v$
    for each position $j$ from $loc - 1$ down to 0 do
        $w \leftarrow$ node at position $j$
        if arc $(v, w)$ exists then
            $val \leftarrow val - 1$
        else if arc $(w, v)$ exists then
            $val \leftarrow val + 1$
        if $val \leq min$ then
            $min \leftarrow val, loc \leftarrow j$
    Insert $v$ at position $loc$
```

3 Our Approach

Our approach is based on running the well known PageRank algorithm [8, 17] on the directed line graph of the original directed graph. The *line graph* of an undirected graph G is another graph $L(G)$ that is constructed as follows: each edge in G corresponds to a node in $L(G)$ and for every two edges in G that are incident to a node v an edge is placed in $L(G)$ between the corresponding nodes. Clearly, the number of nodes of a line graph is m and the number of edges is proportional to the sum of squares of the degrees of the nodes in G, see [18]. If G is a directed graph, its *directed line graph* (or *line digraph*) $L(G)$ has one node for each edge of G. Two nodes representing directed edges incident upon v in G (one incoming into v, and one outgoing from v), called $L(u, v)$, and $L(v, w)$, are connected by a directed edge from $L(u, v)$ to $L(v, w)$ in $L(G)$. In other words, every edge in $L(G)$ represents a directed path in G of length two. Similarly, the number of nodes of a line digraph is m and the number of edges is proportional to $\sum_{u \in V}[d^+(u) \times d^-(u)]$. Hence, the size of $L(G)$ is $O(m + \sum_{u \in V}[d^+(u) \times d^-(u)])$. Figure 1 depicts a simple directed graph and its resulting line digraph.

Given a digraph $G = (V, E)$ our approach is to compute its line digraph, $L(G)$, run a number of iterations of PageRank on $L(G)$ and remove the node of highest PageRank in $L(G)$. Our experimental results indicate that PageRank values converge reasonably well within five iterations.

A digraph G is *strongly connected* if for every pair of vertices of G there is a cycle that contains them. If G is not strongly connected, it can be decomposed into its *strongly connected components (SCC)* in linear time [22]. A strongly connected component of a directed graph G is a maximal subgraph in which there is a path from every vertex to every other vertex. The term maximal means that no additional vertices or edges of G can be included in the subgraph and keep the subgraph strongly connected. If each SCC is contracted to a single vertex, the resulting graph is a directed acyclic graph (DAG). It follows that feedback arcs can exist only inside the SCCs of G. Hence we can apply this approach inside each SCC, using their corresponding line digraph, and remove the appropriate edges from each SCC. This approach will avoid performing several useless computations and thus reduce the running time of the algorithm.

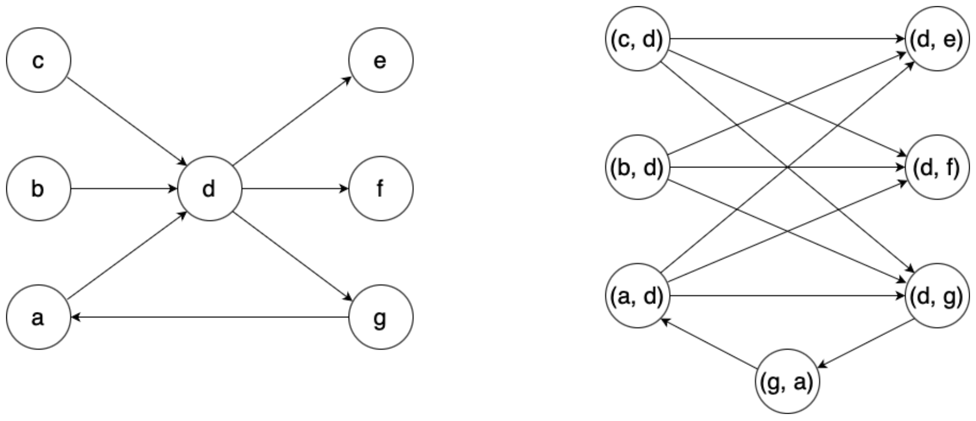


Figure 1: A simple directed graph (left) and its resulting line digraph (right).

3.1 Line Graph

In order to obtain the line digraph of G, we use a DFS-based approach. First, for each edge (u, v) of G, we create a node (u, v) in $L(G)$ and then run the following recursive procedure. For a node v, we mark it as visited and iterate through each of its outgoing edges. For each outgoing edge (v, u) of v, we add an edge in $L(G)$ from the *prev* $L(G)$ node that was processed before the procedure's call to the node (v, u). Afterwards we call the same procedure for u if it is not visited with (v, u) as *prev*. If u is visited we add an edge from (v, u) to each one of $L(G)$'s nodes corresponding from u. Since this technique is based on DFS, the running time is $O(n + m + |L(G)|)$. The pseudocode for computing a line digraph is presented in Algorithm 3.

Algorithm 3 LineDigraph

```
Input: Digraph $G = (V, E)$
Output: Line Digraph $L(G)$ of $G$
Create a line digraph $L(G)$ with every edge of $G$ as a node
$v \leftarrow$ random node of $G$
GetLineGraph($G, L(G), v, null$)
return $L(G)$

procedure GETLINEGRAPH($G, L(G), v, prev$)
    Mark $v$ as visited
    for each edge $e = (v, u)$ outgoing of $v$ do
        $z \leftarrow$ node of $L(G)$ representing $e$
        Create an edge in $L(G)$ from $prev$ to $z$          ▷ Given that $prev$ is not nill
        if $u$ is not visited then
            GetLineGraph($G, L(G), u, z$)
        else
            for each node $k$ in $L(G)$ that originates from $u$ do
                Create an edge in $L(G)$ from $z$ to $k$
```

3.2 PageRank

PageRank was first introduced by Brin & Page in 1998 [8, 17]. It was developed in order to determine a measure of importance of web pages in a hyperlinked network of web pages. The basic idea is that PageRank will assign a score of importance to every node (web page) in the network. The underlying assumption is that important nodes are those that receive many "recommendations" (in-links) from other important nodes (web pages). In other words, it is a link analysis algorithm that assigns numerical scores to the nodes of a graph in order to measure the importance of each node in the graph. PageRank works by counting the number and quality/importance of edges pointing to a node and then estimate the importance of that node. We use a similar approach in order to determine the importance of edges in a directed graph. Our technique shows that the number of cycles that contain a specific edge e will be reflected in the PageRank score of e, see Section 3.3. Thus the removal of edges with high PageRank score is likely to break the most cycles in the graph.

Given a graph with n nodes and m edges, PageRank starts by assigning an initial score of $1/n$ to all the nodes of a graph. Then for a predefined number of iterations each node divides its current score equally amongst its outgoing edges and then passes these values to the nodes it is pointing to. If a node has no outgoing links then it keeps its score to itself. Afterwards, each node updates its new score to be the sum of the incoming values. It is obvious that after enough iterations all PageRank values will inevitably gather in the sinks of the graph. In use cases where that is a problem (e.g., when sinks exist) a damping factor is used, where each node gets a percentage of its designated score and the rest gets passed to all other nodes of the graph. For our use case we have no need for this damping factor because we apply PageRank on the Strongly Connected Components of the graph. Hence, the scores of the nodes truly reflect their importance. The number of iterations depends on the size and structure of a graph. We found that for small and medium graphs, which is the case in the scenario for graph visualization, about five iterations were enough for the scores of the nodes to converge. Depending on the implementation, PageRank can run in $O(k(n+m))$ time, where k is the number of iterations. The pseudocode for PageRank is presented in Algorithm 4.

Algorithm 4 PageRank

```
Input: Digraph $G = (V, E)$, number of iterations $k$
Output: PageRank scores of $G$
for each node $v$ in $G$ do
    $PR(v) \leftarrow \frac{1}{|V|}$
for $k$ iterations do
    for each node $v$ in $G$ do
        $PR(v) \leftarrow \sum_{u \in in(v)} \frac{PR_{old}(u)}{|out(u)|}$
return $PR$
```

3.3 PageRank and Number of Cycles

Our approach is based on the observation that the number of cycles that involve an edge is proportional to the edge's PageRank score. In Figure 2 we show a small example in which as we increase the number of cycles the PageRank score of the edges changes. We observe that the change of score in an edge reflects the number of cycles that contain this edge. In other words, the number of cycles that would be broken if the edge of highest PageRank score is removed.

We start with a simple graph that contains one 2-cycle between nodes c and d. At this point the score of both edges is equal to 0.5 (Figure 2a). Then we manually add back edges in order to increase the number of cycles and observe the PageRank score of the edges (i.e., the PageRank score of the nodes of the line graph of the graph). We show the PageRank scores of the nodes of the line graph on the figure. As we add edge (d, b) the score of the edges has changed as shown in Figure 2b. Notice that edge (c, d) has the highest score now, as it participates in two cycles, i.e., it is clear that its removal will break both cycles. The addition of another back edge (d, a) increases the number of cycles, and the score of edge (c, d) is still the highest (although it is a bit lower than before) as it participates in three cycles (Figure 2c). The minimum feedback arc set size is still one, edge (c, d). Finally, the addition of another back edge (c, b) changes the PageRank scores of the edges significantly. We observe that the minimum feedback arc set size is two now. Edges (b, c) and (c, d) now participate in three cycles (Figure 2d). Taking a look at the PageRank scores, we see that edge (b, c) now has the highest score, while edges (c, d) and (c, b) have the same score, 0.190. This shows that the edge with most involvements in cycles does have the highest PageRank score and we can remove it, but we cannot do the same with the second highest score, as it is not clear how the removal of edge (b, c) will affect the rest of the scores.

3.4 PageRankFAS

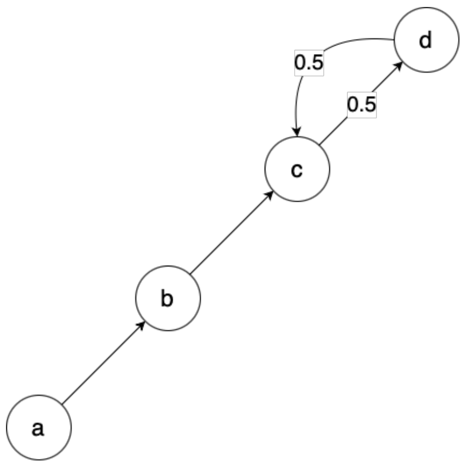
Our proposed algorithm is based on the concepts of PageRank and Line Digraphs. The idea behind *PageRankFAS* is that we can score the edges of G based on their involvement in cycles: For each strongly connected component $(s_1, s_2, ..., s_j)$ of G, it computes the line digraph $L(s_i)$ of the i-th strongly connected component, to transform edges to nodes; next it runs the PageRank algorithm on $L(s_i)$ to obtain a score for each edge of s_i in G.

We observed that the nodes of the line digraphs with the highest PageRank score correspond to edges that are involved in the most cycles of G. We also observed that the nodes of the line digraphs with lower score correspond to edges of G with low involvement in cycles. Using this knowledge, we run PageRankFAS for a number of iterations. In each iteration, we use PageRank to calculate the node scores of each $L(s_i)$ and remove the node(s) with the highest PageRank score, also removing the corresponding edge(s) from G. Please notice that PageRankFAS removes one edge at a time and another iteration of our algorithm is needed only if a cycle still exists. Once we remove one last edge and the last cycle breaks, the algorithm finishes. The time complexity for each iteration of PageRankFAS is linear with respect to the size of the line digraph of each component. In other words it is $O(m + \sum_{u \in V}[d^+(u) \times d^-(u)])$, where m is the number of nodes and $\sum_{u \in V}[d^+(u) \times d^-(u)]$ the number of edges of each line digraph $L(s_i)$. We repeat this process until G becomes acyclic. The pseudocode is presented in Algorithm 5.

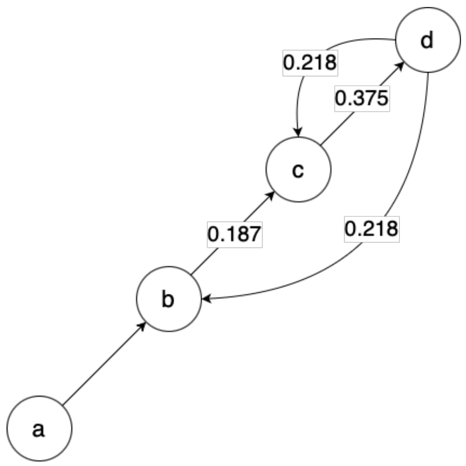
4 Experiments and Discussion

Here we report our experimental results and describe some details of our setup. All the experimented algorithms are implemented in Java 17 using the web-graph framework [5, 6] and tested on a single machine with Apple's M1 processor, 8GB of RAM and running macOS Monterey 12.

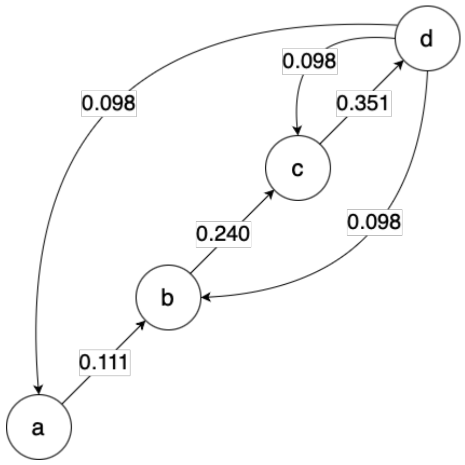
Datasets: In order to evaluate our heuristic algorithm we used five different datasets:



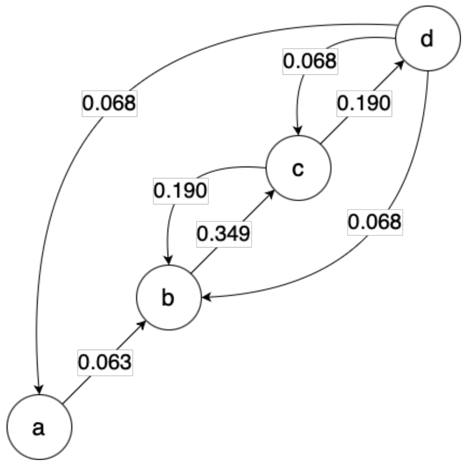
(a) PageRank scores of a simple graph with one back edge.



(b) PageRank scores of a simple graph with two back edges.



(c) PageRank scores of a simple graph with three back edges.



(d) PageRank scores of a simple graph with four back edges.

Figure 2: Example illustrating how the involvement in cycles affects the PageRank score of each edge.

Algorithm 5 PageRankFAS

```
Input: Digraph $G = (V, E)$
Output: Feedback Arc Set of $G$
$fas \leftarrow \emptyset$
while $G$ has cycles do
    Let $(s_1, s_2, ..., s_j)$ be the strongly connected components of $G$
    for each strongly connected component $s_i$ do
        Create a line digraph $L(s_i)$ with every edge of $s_i$ as a node
        $v \leftarrow$ random node of $s_i$
        GetLineGraph($s_i, L(s_i), v, null$)
        PageRank($L(s_i)$, 5)
        $u \leftarrow$ node of $L(s_i)$ with highest PageRank value
        $e \leftarrow$ edge of $G$ corresponding to $u$
        Add $e$ to $fas$
        Remove $e$ from $G$
return $fas$
```

1. Randomly generated graphs with 100, 200, 400, 1000, 2000, 4000 nodes and an average out-degree of 1.5, 3 and 5 each. This is done in order to see how our technique performs with respect to the size of graphs.

2. Three directed graphs from the datasets in graphdrawing.org, suitably modified in order to contain cycles (since the originals are DAGs). This is done in order to observe the performance of our technique with respect to graphs commonly used in graph drawing.

3. Randomly generated graphs with 50, 100 and 150 nodes and average out-degrees of 1.5, 3, 5, 8, 10 and 15 each. This is done in order to see how our technique performs with respect to the density of graphs.

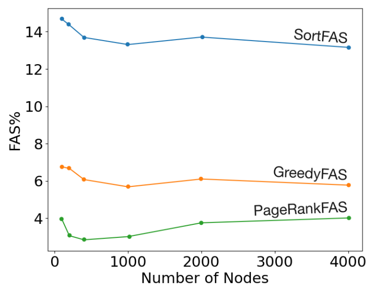
4. In [2] the authors conducted extensive experiments on 4,468 test graphs of varying size and density. We use their graphs since the minimum FAS is produced by the proposed method of [2]. We run our PageRank heuristic on the graphs that required the highest time to compute the optimum FAS in order to compare the results in terms of FAS size and execution time.

5. Three web-graphs from the Laboratory of Web Algorithmics[1], that are used in [20].

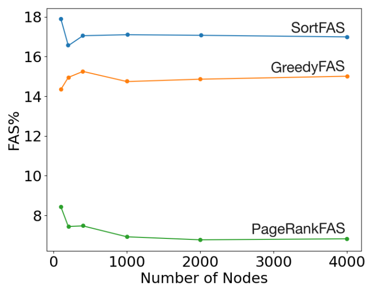
We generated a set of randomly constructed graphs, comprising a total of 36 graphs used as a baseline. The baseline graphs were created using predetermined values for the number of nodes, average out-degree, and back edge percentage. One advantage of this constructed model is that we have prior knowledge of an upper limit for the size of a minimum feedback arc set (FAS). This upper limit is determined by dividing the number of randomly generated back edges by the total number of edges in the graph. To avoid the impact of randomness and ensure consistent results, we executed the three algorithms on 10 randomly generated graphs for each of the baseline graph. Subsequently, we calculated the average values from these multiple runs. This helps to smooth out any abrupt variations in our curves and provides more reliable results.

[1]https://law.di.unimi.it/datasets.php

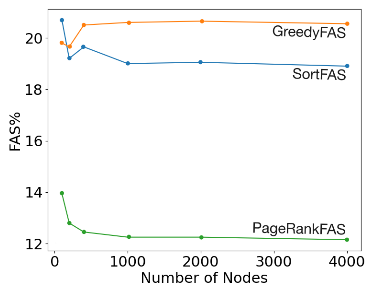
4.1 FAS with Respect to the Number of Nodes



(a) Graphs with average out-degree 1.5



(b) Graphs with average out-degree 3

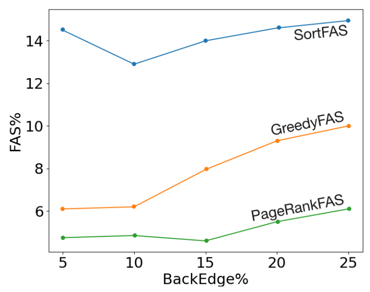


(c) Graphs with average out-degree 5

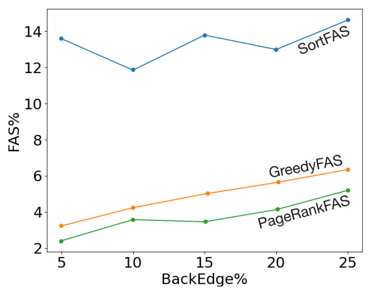
Figure 3: FAS percentage for graphs with increasing number of nodes and three different average out-degrees.

The first set of experiments gives us an idea of how PageRankFAS performs on graphs, with varying number of nodes in comparison to the other two algorithms. It is noteworthy that in most cases the FAS found by PageRankFAS is less than 50% of the FAS found by GreedyFAS and SortFAS. As a matter of fact, for large visualization graphs with 4,000 nodes and 12,000 edges the reduction in the FAS size is almost 55% with respect to the FAS produced by GreedyFAS. The execution time taken by PageRankFAS is less than one second for graphs up to 1,000 nodes, which is similar to the time of the other two heuristics. For the larger graphs, even up to 4,000 nodes the time required is less than 8 seconds, whereas, the other heuristics run in about 1-2 seconds. The results of this experiment are shown in Figure 3. It is interesting to note that the performance of SortFAS is better than the performance of GreedyFAS as the graphs become denser, and in fact, SortFAS actually out-performs GreedyFAS when the graphs have an average out-degree 5 and above, see Figure 3c.

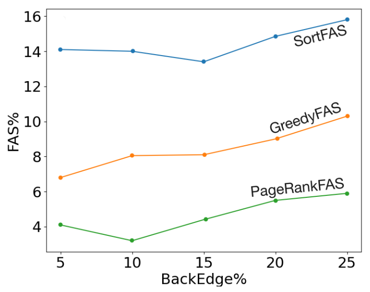
4.2 FAS with Respect to The Number of Back Edges



(a) Graph with 50 nodes and 75 edges before modification



(b) Graph with 75 nodes and 86 edges before modification

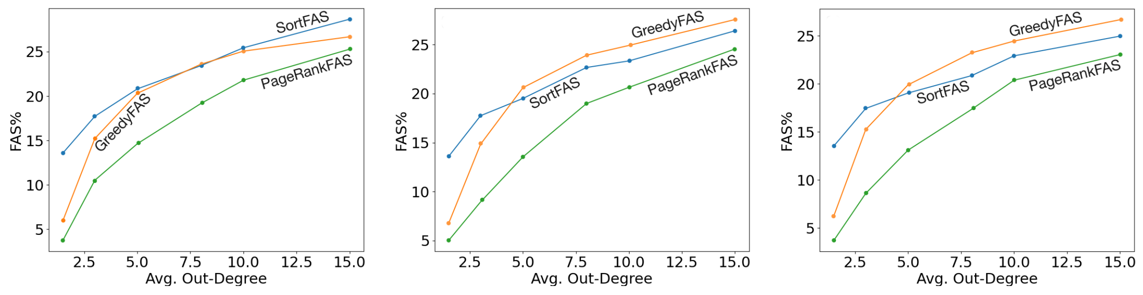


(c) Graph with 99 nodes and 154 edges before modification

Figure 4: FAS percentage for 3 types of graphs from graphdrawing.org and for various numbers of back edges.

The second type of experiments make use of three graphs from graphdrawing.org. Since these graphs are directed acyclic, we randomly added back edges in different percentages of the total number of edges. We did this in a controlled manner in order to know in advance an upper bound of FAS. PageRankFAS gave by far the best FAS results and GreedyFAS also produced FAS with sizes mostly below 10%. SortFAS was not competitive in this dataset. The results are shown in Figure 4. The execution time taken by PageRankFAS is well below 0.15 of a second for all graphs, which is similar to the other two heuristics.

4.3 FAS with Respect to the Average Out-Degree



(a) Graphs with 50 nodes (b) Graphs with 100 nodes (c) Graphs with 150 nodes

Figure 5: FAS percentage depending on the average out-degree of three different types of graphs.

Motivated by the results shown in Figure 3c we decided to investigate the correlation between the density of a graph and its potential FAS percentage. In this experiment, we created 18 different graphs, six of them with 50 nodes, six with 100 nodes and six with 150 nodes as follows: For each node size (i.e., 50, 100, 150) six graphs with average out-degrees 1.5, 3, 5, 8, 10 and 15. Again, as with our previous experiments, the results reported here are the averages of 10 runs in order to compensate for the randomness of each graph and to get smoother curves. The results of this experiment are shown in Figure 5.

The results of PageRankFAS are consistently better than the results of GreedyFAS and SortFAS for all graphs. The results of GreedyFAS and SortFAS are very close to each other, for the graphs with 50 nodes. Notice however that, SortFAS outperforms GreedyFAS when the number of nodes exceeds 100 and the average out-degree exceeds five. This is aligned with the results shown in Figure 3c. Furthermore, as expected, when the average out-degree increases the FAS size clearly increases. Consequently, all techniques seem to converge at higher percentages of FAS size. Again, PageRankFAS runs in a small fraction of a second for all graphs, which is similar to the running times of the other two heuristics.

4.4 Comparing the Results of PageRankFAS to the Optimum

In this section we present the results obtained by PageRankFAS, GreedyFAS and SortFAS and compare them to the optimum as produced by the proposed method of [2]. The authors presented extensive experiments on 4,468 test graphs of varying size and density. Their aim was to test the scalability of their exact algorithms over a wide range. From the point of view of our heuristics, the size of these graphs is small to medium. Hence, the heuristics found solutions in a few tenths of a second, and therefore it is not worth comparing the various heuristics in terms of time. However, it is important to compare the solutions in terms of the size of a FAS produced by them. For the first set of comparisons, we test the three heuristics on the Erdős–Rényi graphs of [2]. While the authors used a total of 3,738 of these graphs, we decided to focus on the twenty graphs that took the longest time to compute the optimum solution. For these twenty graphs, the optimum solution took anywhere from 5,600 seconds to 85,000 seconds, depending on their size and density, to compute using the approach of [2] but as mentioned, all three heuristics took only a few tenths of a second to execute. We can see from the results that PageRankFAS gives the better results consistently on all of the graphs, often splitting the difference between the optimal result and the results of the other heuristics. The results are presented in Figure 6 and Table 1.

The second set of comparisons makes use of the Tournament graphs used in [2]. Again, the authors used a total of 708 of these graphs but we focus on the twenty that took the longest time to compute the optimum solution. Surprisingly, we observe that GreedyFAS gives the worse results on all graphs while PageRankFAS and SortFAS give the best. It is known that tournament graphs are Hamiltonian and the Hamiltonian paths are in a one-to-one correspondence with the minimal FAS of the tournament [3]. That may be an explanation for the improved performance of SortFAS, as that property may make the placement of nodes in the linear arrangement easier. The results are presented in Figure 7 and Table 2.

There is an interesting question with respect to the tradeoff of computing an exact FAS solution in several hours or days versus a solution that is between 10% and 20% of the optimum in a few tenths of a second. Clearly, there may be some applications that absolutely require the optimum solution and every effort will be made to compute it. But in many applications a good-enough solution is acceptable. Furthermore, in cases where the graphs are larger than an exact algorithm can handle, say above the maximum number of nodes (120) used in the experiments of [2] a given application has no other option but to use a heuristic solution that according to previous experiments is between 10% and 20% of the optimum solution.

Baharev et. al. report experimental results on some other types of graphs in [2]. In order not to disturb the flow of our paper, we present these additional comparisons in the Appendix. There we show the experimental results obtained by the three heuristics comparing them with the exact solutions on twelve De Bruijn graphs and twelve Imase–Itoh graphs provided by the authors of [2].

4.5 PageRankFAS on web-graphs

The experiments reported in [20] use large and extra large benchmark web-graphs. Their smaller benchmarks are *wordassociation-2011* (with 10,617 nodes, 72,172 edges, which implies an average degree 6.80) and *enron* (with 69,244 nodes, 276,143 edges, which implies an average degree 3.86).

The authors report that the size of FAS found by GreedyFAS and SortFAS for wordassociation-2011 are 18.89% and 20.17%, respectively [20]. We ran PageRankFAS for wordassociation-2011 and obtained a FAS of size 14.85%. Similarly, for web-graph enron they report a FAS of 12.54% and 14.16% respectively. We ran PageRankFAS on web-graph enron and obtained a FAS of size 11.05%. The results are shown in Figures 8a and 8b. As expected, and consistent with our experimental observations of the previous subsections, the FAS size of the denser web-graph (wordassociation-2011) is larger than the FAS size of the sparser graph (enron), as computed by all heuristics.

In addition, we also ran PageRankFAS on a medium benchmark from [20], uk-2007-05@100000 (with 100,000 nodes, 3,050,615 edges, which implies an average degree 30.5) and obtained a FAS size of 9.17%, while the result of GreedyFAS is 10.23%. The authors of [20] did not report the result obtained by SortFAS on this dataset because of its high execution time. The result is shown in Figure 8c.

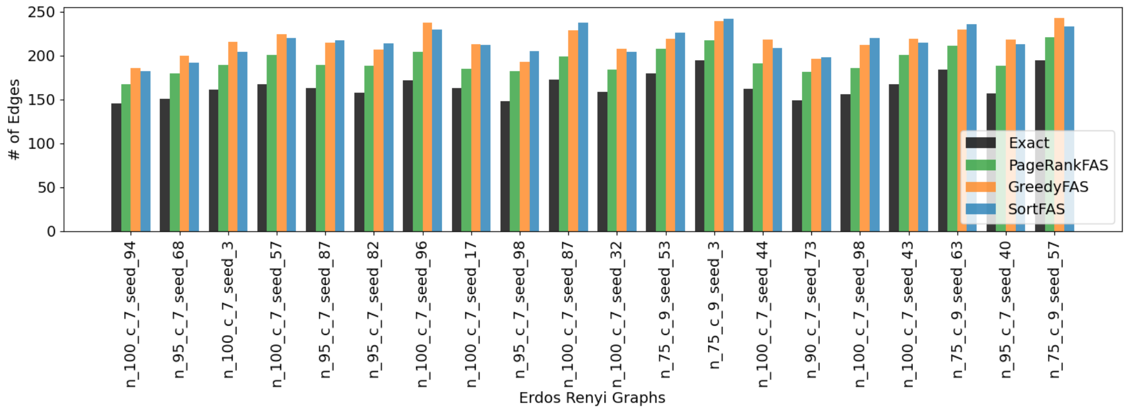


Figure 6: Results of twenty Erdős–Rényi graphs where the exact solution took the longest time to compute.

Graph	Exact Time (1000s)	Exact	PageRankFAS	GreedyFAS	SortFAS
n_100_c_7_seed_94	85.08	145	167	186	182
n_95_c_7_seed_68	35.24	151	180	200	192
n_100_c_7_seed_3	29.21	161	189	216	204
n_100_c_7_seed_57	27.19	167	201	224	220
n_95_c_7_seed_87	18.26	163	189	215	217
n_95_c_7_seed_82	17.62	158	188	207	214
n_100_c_7_seed_96	16.45	172	204	238	230
n_100_c_7_seed_17	16.18	163	185	213	212
n_95_c_7_seed_98	13.60	148	182	193	205
n_100_c_7_seed_87	12.43	173	199	229	238
n_100_c_7_seed_32	10.31	159	184	208	204
n_75_c_9_seed_53	9.11	180	208	219	226
n_75_c_9_seed_3	8.77	195	217	239	242
n_100_c_7_seed_44	8.34	162	191	218	209
n_90_c_7_seed_73	7.68	149	181	196	198
n_100_c_7_seed_98	7.67	156	186	212	220
n_100_c_7_seed_43	6.50	167	201	219	215
n_75_c_9_seed_63	5.94	184	211	230	236
n_95_c_7_seed_40	5.71	157	188	218	213
n_75_c_9_seed_57	5.61	195	221	243	233

Table 1: Results of twenty Erdős–Rényi graphs where the exact solution took the longest time to compute.

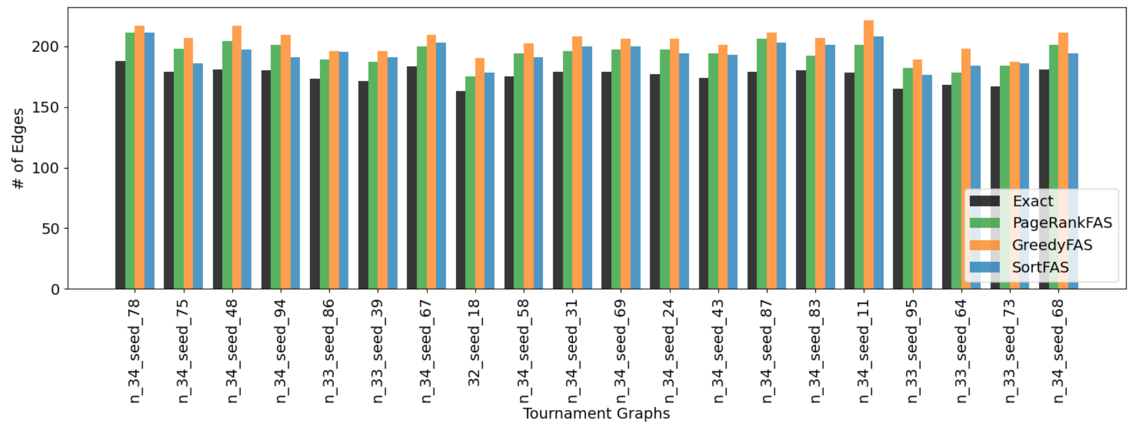
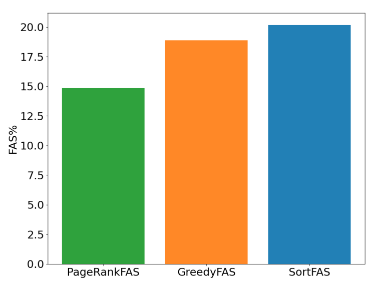


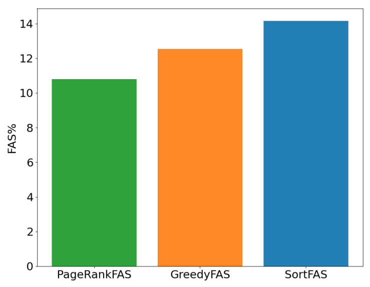
Figure 7: Results of twenty Tournament graphs where the exact solution took the longest time to compute.

Graph	Exact Time (1000s)	Exact	PageRankFAS	GreedyFAS	SortFAS
n_34_seed_78	82.76	188	211	217	211
n_34_seed_75	72.16	179	198	207	186
n_34_seed_48	41.21	181	204	217	197
n_34_seed_94	15.74	180	201	209	191
n_33_seed_86	14.93	173	189	196	195
n_33_seed_39	13.29	171	187	196	191
n_34_seed_67	12.63	183	200	209	203
n_32_seed_18	11.22	163	175	190	178
n_34_seed_58	10.41	175	194	202	191
n_34_seed_31	10.20	179	196	208	200
n_34_seed_69	8.03	179	197	206	200
n_34_seed_24	7.08	177	197	206	194
n_34_seed_43	5.39	174	194	201	193
n_34_seed_87	4.88	179	206	211	203
n_34_seed_83	3.60	180	192	207	201
n_34_seed_11	3.55	178	201	221	208
n_33_seed_95	3.28	165	182	189	176
n_33_seed_64	3.23	168	178	198	184
n_33_seed_73	3.18	167	184	187	186
n_34_seed_68	2.95	181	201	211	194

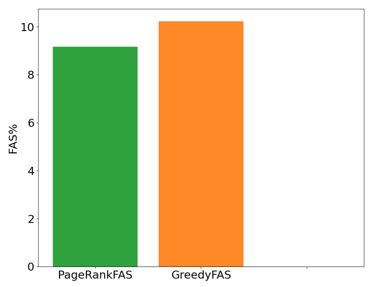
Table 2: Results of twenty Tournament graphs where the exact solution took the longest time to compute.



(a) wordassociation-2011



(b) enron



(c) uk-2007-05@100000

Figure 8: FAS percentage on three web-graphs.

Unfortunately, the required execution time of our heuristic algorithm does not allow us to test it on the larger web-graphs used in [20]. However, it is interesting that there exist FAS of smaller size for these large graphs, which, to the best of our knowledge, was not known before.

5 Conclusions

We presented a heuristic algorithm for computing a FAS of minimum size based on PageRank. Our experimental results show that the size of a FAS computed by our heuristic algorithm is typically about 50% smaller than the sizes obtained by the best previous heuristics, for typical graph drawing datasets. Our algorithm is more time consuming than the best previous heuristics, but it's running time is reasonable for graphs up to 4,000 nodes. For smaller graphs, up to 1,000 nodes, the execution time is well below one second, which is similar to the running times of the other two heuristics. Therefore, this is acceptable for graph drawing applications. We also performed experiments using the graphs that required the highest execution time to compute the minimum FAS and compared the results produced by our heuristic to the exact results reported in [2]. There is an interesting trade-off between computing the minimum FAS in excessive time or obtaining a slightly suboptimal solution in a few tenths of a second. Finally, an interesting side result is that we found out that the FAS-size of three large web-graphs is significantly less than it was known before. Since it is NP-hard to compute the minimum FAS, the optimum solution for these web-graphs is unknown. Hence, we do not know how close our solutions are to the optimum. It would be interesting to investigate techniques to speedup PageRankFAS in order to make it more applicable to larger web-graphs.

Acknowledgements

The authors wish to thank Dr. Ali Baharev for useful discussions regarding the method of [2] and for pointing us to the datasets used in their paper.

References

[1] N. Ailon, M. Charikar, and A. Newman. Aggregating inconsistent information: ranking and clustering. *Journal of the ACM (JACM)*, 55(5):1–27, 2008. doi:10.1145/1411509.1411513.

[2] A. Baharev, H. Schichl, A. Neumaier, and T. Achterberg. An exact method for the minimum feedback arc set problem. *ACM J. Exp. Algorithmics*, 26, apr 2021. doi:10.1145/3446429.

[3] A. Bar-Noy and J. Naor. Sorting, minimal feedback sets, and hamilton paths in tournaments. *SIAM J. Discret. Math.*, 3(1):7–20, 1990. doi:10.1137/0403002.

[4] B. Berger and P. W. Shor. Approximation alogorithms for the maximum acyclic subgraph problem. In *Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms*, pages 236–243, 1990.

[5] P. Boldi, M. Rosa, M. Santini, and S. Vigna. Layered label propagation: a multiresolution coordinate-free ordering for compressing social networks. In S. Srinivasan, K. Ramamritham, A. Kumar, M. P. Ravindra, E. Bertino, and R. Kumar, editors, *Proceedings of the 20th International Conference on World Wide Web, WWW 2011*, pages 587–596. ACM, 2011. doi:10.1145/1963405.1963488.

[6] P. Boldi and S. Vigna. The webgraph framework I: compression techniques. In S. I. Feldman, M. Uretsky, M. Najork, and C. E. Wills, editors, *Proceedings of the 13th international conference on World Wide Web, WWW 2004*, pages 595–602. ACM, 2004. doi:10.1145/988672.988752.

[7] F. J. Brandenburg and K. Hanauer. Sorting heuristics for the feedback arc set problem. In *Technical Report MIP-1104*. University of Passau Germany, 2011.

[8] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Comput. Networks*, 30(1-7):107–117, 1998. doi:10.1016/S0169-7552(98)00110-X.

[9] C. Budak, D. Agrawal, and A. El Abbadi. Limiting the spread of misinformation in social networks. In *Proceedings of the 20th international conference on World wide web*, pages 665–674, 2011. doi:10.1145/1963405.1963499.

[10] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs.* Prentice-Hall, 1999.

[11] P. Eades, X. Lin, and W. F. Smyth. A fast and effective heuristic for the feedback arc set problem. *Inf. Process. Lett.*, 47(6):319–323, 1993. doi:10.1016/0020-0190(93)90079-O.

[12] X. He, G. Song, W. Chen, and Q. Jiang. Influence blocking maximization in social networks under the competitive linear threshold model. In *Proceedings of the Twelfth SIAM International Conference on Data Mining, Anaheim, California, USA, April 26-28, 2012*, pages 463–474. SIAM / Omnipress, 2012. doi:10.1137/1.9781611972825.40.

[13] D. S. Johnson. The NP-completeness column: An ongoing guide. *J. Algorithms*, 3(4):381–395, 1982. doi:10.1016/0196-6774(82)90032-3.

[14] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Proceedings of a symposium on the Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972. doi:10.1007/978-1-4684-2001-2_9.

[15] N. S. Nikolov and P. Healy. *Hierarchical Drawing Algorithms, in Handbook of Graph Drawing and Visualization, ed. Roberto Tamassia.* CRC Press, 2014. pp. 409-453.

[16] G. Ortali and I. G. Tollis. A new framework for hierarchical drawings. *Journal of Graph Algorithms and Applications*, 23(3):553–578, 2019. doi:10.7155/jgaa.00502.

[17] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

[18] S. Pemmaraju and S. Skiena. *Computational Discrete Mathematics: Combinatorics and Graph Theory with Mathematica ®*. Cambridge University Press, 1990.

[19] M. Simpson, V. Srinivasan, and A. Thomo. Clearing contamination in large networks. *IEEE Transactions on Knowledge and Data Engineering*, 28(6):1435–1448, 2016. doi:10.1109/TKDE.2016.2525993.

[20] M. Simpson, V. Srinivasan, and A. Thomo. Efficient computation of feedback arc set at web-scale. *Proceedings of the VLDB Endowment*, 10(3):133–144, 2016. doi:10.14778/3021924.3021930.

[21] K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(2):109–125, 1981. doi:10.1109/TSMC.1981.4308636.

[22] R. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972. doi:10.1137/0201010.

Appendix

A. Additional Comparisons of PageRankFAS to the Optimum

For completeness we present here comparisons of the experimental results obtained by the three heuristics and compare them to the optimum results obtained by the method of [2].

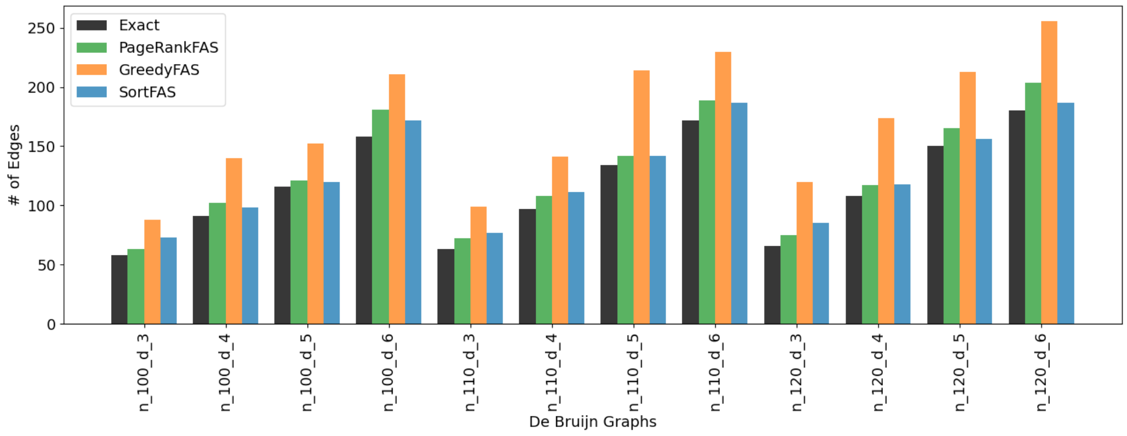


Figure 9: Results of the three algorithms compared to the exact solution for De Bruijn graphs.

Graph	Exact Time (s)	Exact	PageRankFAS	GreedyFAS	SortFAS
n_100_d_3	37.95	58	63	88	73
n_100_d_4	2.07	91	102	140	98
n_100_d_5	1.79	116	121	152	120
n_100_d_6	31.03	158	181	211	172
n_110_d_3	1.90	63	72	99	77
n_110_d_4	332.61	97	108	141	111
n_110_d_5	39.95	134	142	214	142
n_110_d_6	10,606.27	172	189	230	187
n_120_d_3	1.98	66	75	120	85
n_120_d_4	6.53	108	117	174	118
n_120_d_5	1.95	150	165	213	156
n_120_d_6	6,699.13	180	204	256	187

Table 3: Results of the three algorithms compared to the exact solution for De Bruijn graphs.

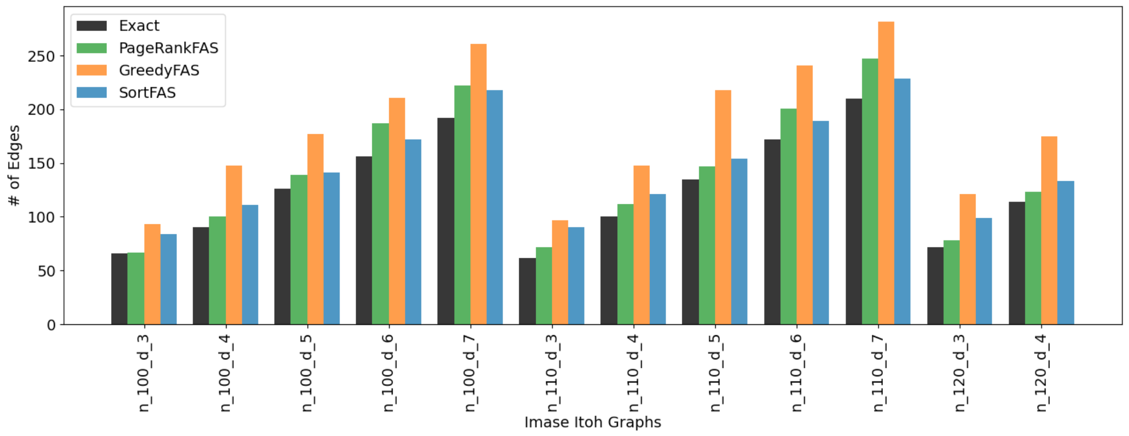


Figure 10: Results of the three algorithms compared to the exact solution for Imase–Itoh graphs.

Graph	Exact Time (s)	Exact	PageRankFAS	GreedyFAS	SortFAS
n_100_d_3	0.78	66	67	93	84
n_100_d_4	1.38	90	100	148	111
n_100_d_5	1.42	126	139	177	141
n_100_d_6	32.60	156	187	211	172
n_100_d_7	25.89	192	222	261	218
n_110_d_3	3.08	62	72	97	90
n_110_d_4	5.93	100	112	148	121
n_110_d_5	7.14	135	147	218	154
n_110_d_6	59.04	172	201	241	189
n_110_d_7	6,821.63	210	247	282	229
n_120_d_3	11.98	72	78	121	99
n_120_d_4	Timeout	114	123	175	133

Table 4: Results of the three algorithms compared to the exact solution for Imase–Itoh graphs.

In the first set of comparisons, we use the eight De Bruijn graphs from [2]. For this dataset PageRankFAS and SortFAS proved to be most effective with both heuristics coming close to the exact solution, while GreedyFAS was not competitive. We observe that PageRankFAS performs better on the graphs with lower density while SortFAS performs better on the graphs with higher density. De Bruijn graphs are known to be Eulerian and Hamiltonian. It is possible that this fact makes SortFAS find better solutions by computing a better placement of nodes into a linear arrangement easier. The results are presented in Figure 9 and Table 3. In any case, as shown in Table 3 the exact method runs rather fast with the exception of only two graphs.

We also compare the results of the three heuristics on the twelve Imase–Itoh graphs from [2]. As with the previous set of comparisons, we observe that PageRankFAS and SortFAS perform best depending on the density of the graphs, while GreedyFAS remains noncompetitive. The results are presented in Figure 10 and Table 4. In any case, as shown in Table 4 the exact method runs rather fast with the exception of only two graphs.