Influence of Graph Characteristics on Solutions of Feedback Arc Set Problem

Ema Leila Grošelj
eg61487@student.uni-lj.si
University of Ljubljana
Faculty of Computer
and Information science
Večna pot 113
SI-1000 Ljubljana, Slovenija

Tomaž Poljanšek
tp51101@student.uni-lj.si
University of Ljubljana
Faculty of Computer
and Information science
Večna pot 113
SI-1000 Ljubljana, Slovenija

ABSTRACT

In this article we present Feedback Arc Set problem and how certain graph characteristics impact the results of heuristic algorithms. We then inspect how the most promising characteristic (treewidth) helps in choosing the most appropriate heuristics for our graph.

KEYWORDS

graph, FAS, heuristics, treewidth, random forest classifier

1 INTRODUCTION

In this article, we tackled the *feedback arc set* problem, where the goal is to find the smallest set of directed edges (arcs) in a directed graph such that, when removed, the graph becomes acyclic (directed acyclic graph - DAG). We were interested in determining which characteristics of a graph suggest that a particular heuristic method might perform poorly, providing a solution not close to optimum.

Due to the trivial nature, we were not interested in the impact of graph size and aimed to normalize this effect. We collected graphs from two existing graph datasets. From these, we built a database of their strongly connected components.

The database of components was enriched with characteristics: number of nodes and arcs, graph density, radius and diameter, information on whether the graph is planar or bipartite, node connectivity, transitivity and treewidth. We also added information about distribution of some characteristics computed on individual nodes (e.g., degree, different types of centrality).

2 FEEDBACK ARC SET

First, let us state that from now on, when we say 'graph', we mean a directed and strongly connected graph with n nodes and m arcs.

Definition 2.1. Feedback arc set (FAS) of a graph $G = (V, A)$ is $A' \subseteq A$ such that $G' = (V, A \setminus A')$ is DAG. *MFAS* (minimum FAS) is the smallest possible FAS.

In this article, we aim to approximate MFAS size using heuristics, as FAS problem is one of Karp's 21 famous NP-complete problems [10]. Unfortunately, it also does not have an approximation scheme. If a graph is already acyclic, its FAS is empty.

2.1 Upper Bound

Every arc in MFAS lies in at least one cycle. If an arc does not lie in a cycle, it does not need to be removed from the graph. This would contradict the minimality of MFAS. Thus, to break all cycles, it is sufficient to remove one arc from each cycle. However, the FAS composed of these arcs is not necessarily the MFAS, as removing one arc can break multiple cycles simultaneously, requiring fewer arcs to be removed than there are cycles. Therefore, the number of cycles is an upper bound on the size of MFAS.

However, this bound can be very loose, as shown in [3], where a graph with n nodes and m arcs can have up to 1.433^m cycles, and the number of arcs can be quadratic in the number of nodes, i.e., $m = O\left(n^2\right)$. In addition to being loose, counting the number of cycles is computationally demanding. For example, the algorithm in Python library Networkx [7] requires $O((n+m)(c+1))$ time steps, where c is the number of cycles. This means number of cycles is potentially exponential to number of vertices so we can only count all cycles for small graphs in a reasonable time.

Another upper bound adequate also for large graphs is represented by the best result of the heuristics. This is much better since it is computed much faster. Thus in this article we take the highest heuristic result as an upper bound.

2.2 Lower Bound

For the lower bound, we can use disjoint cycles in the graph. They provide a lower bound because we need to break all these (disjoint) cycles and due to disjointness, we cannot break two cycles by removing one arc.

Note that not all (exhaustive) sets of disjoint cycles in a graph are equally strong. For instance, consider a graph with arcs $A = \{(1,2),(2,3),(3,2),(3,1),(1,3)\}$. This graph has three cycles and two sets of disjoint cycles: $\{(1,2,3)\}$ and $\{(1,3),(2,3)\}$. The idea is that by removing the cycle $(1,2,3)$ from the graph, we also break all other cycles (exhausting the disjoint cycles).

The set of disjoint cycles we get will depend on the way we search for one cycle within each iteration. If we introduce randomness in selecting the starting node and the order of nodes during the search, we obtain a random algorithm. When running the algorithm multiple times, we only consider the largest set as we aim for a tighter lower bound. In this article we ran search for disjoint cycles 10 times.

3 DATA

3.1 Data Sources

We collected graphs from two sources.

In [8] they used different generation methods to build a collection of large weighted multi-digraphs that included different topologies. Graphs there were nicely divided in groups: *De Bruijn graphs, Delaunay 3D graphs, Kautz graphs, Triangulation graphs,*

Small world graphs and *Random graphs*. They derived them from ISPD98 Circuit Benchmark Suite [2].

A collection of graphs presented in [5] was intended for the problem of *optimum cycle mean and ratio*. It consists of graphs from ISPD98 Circuit Benchmark Suite [2] and random graphs that they generated themselves. In these article we refer to them as *unclassified*.

We read all these graphs, broke them into strongly connected components, as breaking them into such components is usually the first step in heuristics. We wanted to ensure that the sizes of the components (e.g., many small ones and one large one) did not obscure the impact of other interesting characteristics. We then stored the components in *pickle* format for faster re-reading. If a graph contained loops (self-directed arcs), we removed them beforehand and added them to the result at the end, as not all heuristics supported graphs with loops. Sixteen graphs with either more than 5000 nodes or more than 10000 arcs were classified as 'large' graphs and omitted from the study. Thus, the main database contained 11925 graphs.

3.2 Graph Characteristics

For every graph in our database we saved the number of nodes, number of arcs, graph density, planarity, bipartiteness, diameter, radius, node connectivity (minimum number of nodes that need to be removed to disconnect the graph), transitivity (probability that the ends of two arcs that share a common node are themselves connected), and treewidth (see Subsection 3.2.1).

For nodes, we calculated degree, closeness centrality (inverse average shortest path length from the node to all other nodes), betweenness centrality (frequency with which a node is part of the shortest paths between other nodes), degree centrality (the fraction of nodes it is connected to), clustering coefficient (how many triangles a node is part of out of the possible triangles), node centrality (node influence within the network, considering both the number of neighbors and neighbors of neighbors), and PageRank (a rank obtained by the PageRank algorithm measuring the importance of a node). For each graph, we recorded the min, max, median, and interquartile range of these values.

3.2.1 Treewidth. Treewidth represents how close a graph is to being a tree, with trees having a treewidth of 1. It represents the minimum width of the largest component across all tree decompositions of the graph. The treewidth of an undirected graph was extended to directed graphs in [9].

According to [6], for a directed graph G, it holds that

$$directed\_treewidth(G) \leq treewidth(undirected(G)).$$

Equality is achieved when all arcs are bidirectional. Two heuristics, $treewidth\_min\_degree$ and $treewidth\_min\_fill\_in$ implemented in NetworkX library [7], provide an upper bound for the treewidth, which also serves as an upper bound for 'directed treewidth'. The minimum degree heuristic method repeatedly selects and removes the node with the lowest degree, while the minimum fill-in heuristic method selects the node, whose removal minimizes the number of added arcs needed to make its neighborhood a clique.

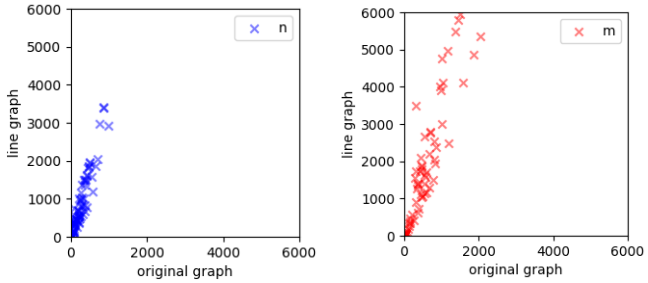

Figure 1: Increase in the number of nodes (left) and arcs (right) in the line graph.

4 HEURISTICS

We have tested five different heuristics.

4.1 SmartAE

We implemented the SmartAE algorithm from article [4]. First, we order the nodes, for example by indegree. Then remove outgoing arcs pointing to unvisited nodes in that order until the graph becomes acyclic. After obtaining an acyclic graph, we attempt to re-add removed arcs one by one if they do not cause cycles.

4.2 DiVerSeS - Using FVS Heuristics

The size of the FAS equals the size of the FVS (feedback vertex set, dual problem) on the line graph. The line graph is obtained by mapping arcs to nodes. In the line graph, two nodes representing arcs from original graph (u, v) and (w, x) are connected with arc (uv, wx) if $v = w$, meaning each arc in the line graph represents a directed path of length two in the original graph. This transformation is described in [12]. Line graphs are typically larger, making the problem harder. Increase in size on subsample of grafs from our database is depicted in Figure 1.

After transformation, we ran the winning FVS solver DiVerSeS from the PACE 2022 challenge [1], which dealt with FVS on directed graphs. We gave it 5 seconds to return a solution. If no solution was found, we ran it for 10 and then 40 seconds. Even then some graphs remained unsolved.

4.3 Graph hierarchy based approaches

We ran algorithms from [11]. Goal is to break cycles while still preserving logical structure (hierarchy) as much as possible. Hierarchy information identifies which edges need to be removed. Heuristics differ in a way they determine hierarchy based on different features.

We decided to test 3 approaches: greedy (FAS Greedy), PageRank rating (PageRank SCC) and Bayesian skill rating system (TrueSkill SCC), giving us another 3 heuristics. FAS Greedy was ran 5 times as it uses randomness and is also by far the fastest.

5 METHODOLOGY

We ran heuristics on the dataset and saved size of FAS and running time. [1] For each graph we determined heuristic method that found

[1] Implementations and dataset can be found at https://github.com/elgroselj/FAS.

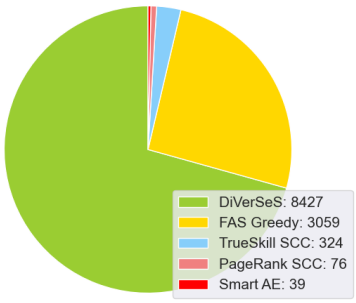

Figure 2: Best solvers.

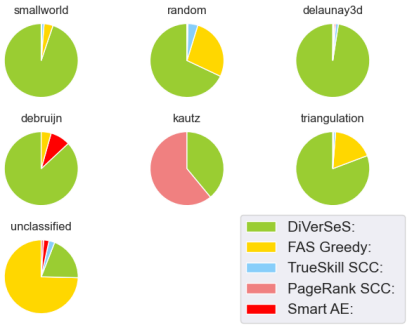

Figure 3: Best solvers by categories.

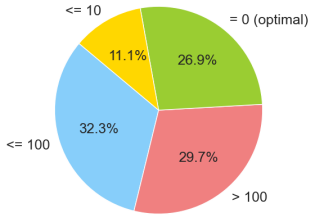

Figure 4: Gap between lower and upper bound.

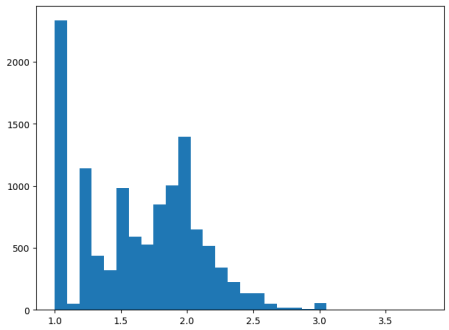

Figure 5: Histogram of the ratios between upper and lower bound.

tha smallest FAS. On a tie, heuristic method with lower running time won.

For determining which graph features are most important in determining which heuristic method is the best to use, we used random forest classifier: it is stable and not to difficult to explain. We trained and evaluated model using 5-fold cross-validation.

Then we evaluated feature importances. Firstly we used model's attribute $feature\_importances$, that represents accumulation of the impurity decrease within each tree. We also tested importance of features using permutation test - that is we randomly permuted values in one column at a time and observed performance degradation.

6 RESULTS

6.1 Heuristics

As shown in Figure 2, the DiVerSeS solver was the best on majority of the graphs. However, for some graph groups other heuristics gave better results as shown in Figure 3. Turns out that on *unclassified* graphs method FAS Greedy reported the best result. On Kautz graphs we recommend to use the method PageRank SCC, while DiVerSeS method dominated on all other graph groups.

If we closely examine differences between lower and upper bounds in Figure 4 we see that in most cases solutions are well constrained - that is lower and upper bound are relatively close, giving us a narrow interval of possible FAS sizes. We prooved optimality in 26.9% of examples. In Figure 5 we see that in most cases ratios between worst and best solutions is lower than two. We have clipped the graph in Figure 5 to show the great majority of ratios, however there are some individual cases where ratio is quite high (most extreme example has ratio of 17.3). For these examples it is good to know which heuristics works best as it makes a lot of difference.

6.2 Classification and features

In classification with random forest classifier we achieved the accuracy of 0.932. This is significant improvement over the majority classifier (predicts DiVerSeS as the winner for all inputs) with an accuracy of 0.707. Feature importance provided by model's feature_importance attribute is shown in Figure 6, while permutation_importance is shown in Figure 7. Features with very little importance are left out.

We can see that treewidth is the most important characteristic according to both figures. This is not very surprising since with edges removal we create acyclic graph or a tree. Characteristics pagerank_max, number of arcs m and Katz centrality_min also have a significant importance. It is also notable that while number of nodes n has accumulated a lot of impurity decrease according to model's feature importance, it lacks at being innovative in the sense that permuting it randomly does not affect the success much, which suggests that n does not provide new information.

Figure 8 shows us that DiVerSeS generally does the best for graphs with treewidth at least 10 (this is also true for graphs with

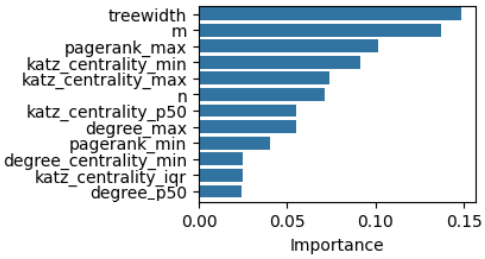

Figure 6: Feature importance.

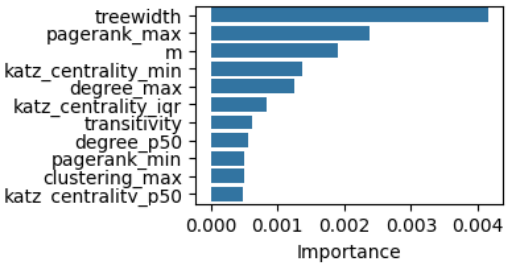

Figure 7: Permutation importance.

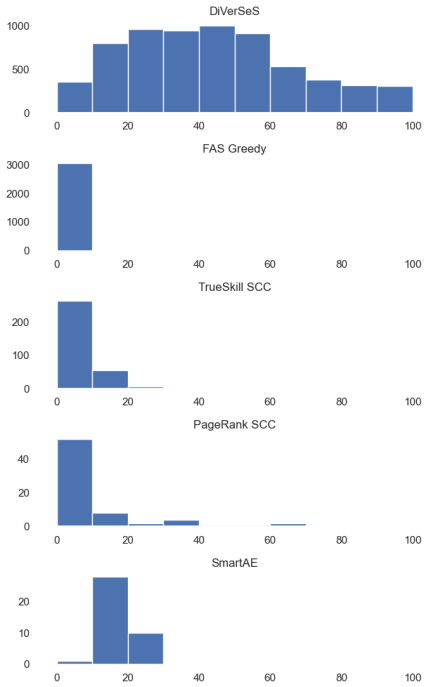

Figure 8: Histograms of treewidths by best solvers.

treewidth ≥ 100). For less than that FAS Greedy heuristic method gives the best results.

7 CONCLUSIONS

Treewidth is the most important graph characteristic in determining the best heuristic for graph. Number of arcs and Katz centrality also have significant impact. For graphs with higher treewidth we recommend using DiVerSeS and for lower treewidth FAS Greedy heuristic.

REFERENCES

[1] 2022. PACE2022. https://pacechallenge.org/2022/tracks/. [Accessed 26-05-2024].
[2] Charles J Alpert. 1998. The ISPD98 circuit benchmark suite. In *Proceedings of the 1998 international symposium on Physical design*. 80–85.
[3] Andrii Arman and Sergei Tsaturian. 2017. The maximum number of cycles in a graph with fixed number of edges. *arXiv preprint arXiv:1702.02662* (2017).
[4] C. Cavallaro, V. Cutello, and M. Pavone. 2023. Effective heuristics for finding small minimal feedback arc set even for large graphs. In *CEUR Workshop Proceedings*, Vol. 3606. https://ceur-ws.org/Vol-3606/paper56.pdf
[5] Ali Dasdan. 2004. Experimental analysis of the fastest optimum cycle ratio and mean algorithms. *ACM Transactions on Design Automation of Electronic Systems* 9, 4 (2004), 385–418. https://doi.org/10.1145/1027084.1027085
[6] Frank Gurski, Dominique Komander, and Carolin Rehs. 2021. How to compute digraph width measures on directed co-graphs. *Theoretical Computer Science* 855 (2021), 161–185.
[7] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. 2008. Exploring network structure, dynamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conference (SciPy2008)*, Gäel Varoquaux, Travis Vaught, and Jarrod Millman (Eds.). Pasadena, CA USA, 11–15.
[8] Michael Hecht, Krzysztof Gonciarz, and Szabolcs Horvát. 2021. Tight localizations of feedback sets. *Journal of Experimental Algorithmics (JEA)* 26 (2021), 1–19.
[9] Thor Johnson, Neil Robertson, Paul D Seymour, and Robin Thomas. 2001. Directed tree-width. *Journal of Combinatorial Theory, Series B* 82, 1 (2001), 138–154.
[10] Richard M Karp. 2010. *Reducibility among combinatorial problems*. Springer.
[11] Jiankai Sun, Deepak Ajwani, Patrick K. Nicholson, Alessandra Sala, Alessandra, and Srinivasan Parthasarathy. 2017. Breaking cycles in noisy hierarchies. In *Proceedings of the 2017 ACM on Web Science Conference*. 151–160.
[12] Jin-Hua Zhao and Hai-Jun Zhou. 2016. Optimal disruption of complex networks. *arXiv preprint arXiv:1605.09257* (2016).