Sorting Heuristics for the Feedback Arc Set Problem

Franz J. Brandenburg and Kathrin Hanauer

Department of Informatics and Mathematics, University of Passau
{brandenb;hanauer}@fim.uni-passau.de

UNIVERSITÄT PASSAU
Fakultät für Informatik und Mathematik

Technical Report, Number MIP-1104
Department of Informatics and Mathematics
University of Passau, Germany
February 2011

Sorting Heuristics for the Feedback Arc Set Problem[⋆]

Franz J. Brandenburg and Kathrin Hanauer

University of Passau, Germany
{brandenb,hanauer}@fim.uni-passau.de

Abstract. The feedback arc set problem plays a prominent role in the four-phase framework to draw directed graphs, also known as the Sugiyama algorithm. It is equivalent to the linear arrangement problem where the vertices of a graph are ordered from left to right and the backward arcs form the feedback arc set.
In this paper we extend classical sorting algorithms to heuristics for the feedback arc set problem. Established algorithms are considered from this point of view, where the directed arcs between vertices serve as binary comparators. We analyze these algorithms and afterwards design hybrid algorithms by their composition in order to gain further improvements. These algorithms primarily differ in the use of insertion sort and sifting and they are very similar in their performance, which varies by about 0.1%. The differences mainly lie in their run time and their convergence to a local minimum. Our studies extend related work by new algorithms and our experiments are conducted on much larger graphs. Overall we can conclude that sifting performs better than insertion sort.

1 Introduction

The feedback arc set problem (FAS) asks for a minimum sized subset of arcs of a directed graph whose removal or reversal makes the graph acyclic. The decision of whether a set of k arcs suffices or not is one of Karp's [15] original $\mathcal{NP}$-complete problems. Its complementary problem is the maximum acyclic subgraph problem. Furthermore, FAS is equivalent to the linear arrangement problem where the vertices of a directed graph are ordered from left to right and the feedback arc set consists of the backward arcs pointing from right to left. As such it is used for ranking problems, where a directed arc (u, v) expresses that u should be ranked before v. This binary relation is incomplete and inconsistent, and additionally it may have weights. Incompleteness means that there is no decision taken between two items or no arc between two vertices, whereas inconsistency is equivalent to the existence of a cyclic dependency. The feedback arc set and the linear arrangement problem ask for the smallest number of violations against the ranking imposed by the arcs.

The linear arrangement provides an adaequate drawing for FAS and makes the feedback arc set clearly visible if the vertices are placed on a horizontal line

[⋆] Supported by the Deutsche Forschungsgemeinschaft (DFG), grant Br835/15-1

with forward arcs being drawn on and above this line whereas the backward arcs appear below it.

Recent research on the feedback arc set problem has resulted in major advancements, particularly for tournaments. In 2005, Ailon et al. proposed a 3-approximation algorithm in the conference version of [1]. It was named KwikSort due to its resemblance with the famous Quicksort algorithm. We extend their algorithm to arbitrary directed graphs. Moreover, there is a polynomial time approximation scheme by Kenyon-Mathieu and Schudy [16], which allows the computation of a feedback arc set with a size of at most $1+\epsilon$ times the optimum for any $\epsilon > 0$. This result is achieved by applying KwikSort once and improving the arrangement by repeatedly removing vertices from the ranking and reinserting them at another position. This procedure equals our sifting algorithm.

FAS is an aggravating problem and seems to be harder than other $\mathcal{NP}$-complete problems, as there are only a few other $\mathcal{NP}$-complete problems which reduce from FAS and FAS is hard to approximate. The best known approximation ratio is $\mathcal{O}(\log n \log \log n)$ [12] and it cannot be approximated any better than 1.36 unless $\mathcal{P}$ is equal to $\mathcal{NP}$ [8]. For planar graphs, however, FAS can be solved in polynomial time, and the duality of linear programming holds, i.e., the size of the minimal feedback arc set equals the maximum number of arc disjoint cycles, see [3]. FAS is also solvable in polynomial time for reducible flow graphs [18], which are used in modeling the control flow of programs and in code optimization.

The feedback arc set problem also plays a prominent role in graph drawing. In their seminal paper Sugiyama, Tagawa, and Toda [23] introduced the four-phase framework for drawing directed graphs. The phases are: decycling, layering, crossing reduction, and coordinate assignment. The feedback arc set problem appears twice in the framework, namely, in decycling and in the penalty graph approach for crossing reduction.

The idea of applying sorting techniques for sets without a total linear order has been used elsewhere. It appears under different names and with different implementations. The deletion of an element from a sequence and its reinsertion corresponds to a mutation in genetic algorithms, and is also known as 1-OPT and sifting. As the latter it is successfully used, e.g., for crossing minimization in hierarchical drawings of graphs [17,13,2].

In this paper we examine several existing heuristics for the feedback arc set or the linear arrangement problem, and we regard them as extended sorting algorithms. We analyze and compare them, establish formal properties and discover some weak points. The comparison includes the convergence towards a local minimum, which shows the widest differences. Consequently, we combine the most promising approaches to hybrid algorithms for the 'best from all'. The thereby obtained algorithms are quite similar in their performance, but some differences can be established. The heuristics are tested on several data sets, including the ones where the basic algorithms perform badly. Ultimately, we contrast insertion sort and sifting as integral parts of hybrid algorithms. In our studies sifting performs better than insertion sort.

The presented algorithms are a selection and advancement of a study by Hanauer [14], which also includes variants of the algorithms by Berger and Shor [4], Saab [19], and Demetrescu and Finocchi [7]. These algorithms do not fit into this study since they are not affiliated with sorting. Moreover, they are not competitive if quality and time are taken into account. There are related studies by Coleman and Wirth [6] and by Schalekamp and van Zuylen [20], which, however, focus on tournaments and on rankings and thus operate with complete orders.

2 Algorithms

In this section we provide a description of our algorithms. We consider directed graphs $G = (V, E)$ with n vertices and m arcs. For the linear arrangement problem the vertices are ordered to $\pi = (v_1, \dots, v_n)$. In this case each vertex has incoming and outgoing arcs to the left and to the right. The *cost* $c(\pi)$ is the size of the feedback arc set under π. The objective clearly is to find a linear arrangement with minimum cost.

2.1 Preprocessing

The algorithms are usually designed to operate on simple directed graphs. For other graphs some preprocessing steps need to be taken. Obviously, each self-loop in the feedback arc set is unavoidable, but it has no further impact on other cycles. Thus self-loops are removed from the graph and are later added to the feedback arc set. Two-cycles consisting of opposite arcs (u, v) and (v, u) can be processed by first erasing both arcs and finally adding exactly one of them to the feedback arc set — the arc which conforms to the linear arrangement. Parallel arcs are transformed into a bundle of paths of length two by splitting each of them by an auxiliary vertex.

Finally, a directed graph can be decomposed into its strongly connected components, which again can be sorted topologically. In particular, the minimum feedback arc set is empty if and only if the graph is acyclic. Clearly, a minimal linear arrangement or feedback arc set must respect the order of arcs (u, v) if u and v belong to different strongly connected components and u must be listed before v in the topological order. If this holds, we say that the linear arrangement is *SCC-conform*.

We shall see that the pure algorithms do not respect the thereby obtained order. It has been observed in [14], however, that the loss is rather small in general. Moreover, it is too time consuming to compute strongly connected components and their topological order repeatedly, as in the approaches by Eades and Lin [9] and by Saab [19].

2.2 Postprocessing

In a postprocessing phase one may clean up the obtained feedback arc set or a linear arrangement π.

A set of arcs F is a *minimal* feedback arc set for $G = (V, E)$ if the reinsertion of any arc $f \in F$ induces a cycle, i.e., $G' = (V, E \setminus F \cup \{f\})$ is not acyclic. This check can be done in $\mathcal{O}(|F|) \times \mathcal{O}(n+m)$, which is quite costly. This particularly holds for dense graphs with large feedback arc sets where it comes to $\mathcal{O}(n^4)$.

Also, if vertices u and v are adjacent in a linear arrangement π, then u must appear before v if there is an arc (u, v); otherwise they are swapped. This can be cleared in $\mathcal{O}(n^2)$ time.

2.3 Properties

Next, we establish some formal properties.

Definition 1. *An algorithm is monotone if the output is never worse than the input.*

Here the size of the set of feedback or backward arcs is never increased. Hence, it is promising to run the algorithm repeatedly.

Definition 2. *For a monotone algorithm A the convergence number is the number of runs of A that strictly improve the result. The resulting algorithm is denoted by A^* and described more formally in 1.*

Algorithm 1 A^* Algorithm

```
function ITERATE(Algorithm $A$, Cost $c$, Arrangement $\pi$)
    repeat
        $\pi' \leftarrow \pi$
        $\pi \leftarrow A(\pi)$
    until $c(\pi) \geq c(\pi')$
end function
```

The outcome of algorithm A^* is a first local optimum for an algorithm A. If A is not monotone, then the result of the next to last run should be taken. The enforcement of a strict improvement generally leads to a faster termination of A^* and a lower convergence number. Alternatively, one may repeat A until it worsens the result for the first time or repeats a configuration π. However, it is too time consuming to check this termination criterion.

The *local median order* in [3] coincides with the commonly used *1-OPT* property. An algorithm is said to be *1-OPT* if the removal of an item and its reinsertion does not lead to an improvement.

It should be noted that our repeated runs of algorithms do not guarantee *1-OPT*, although sifting performs these operations. For *1-OPT* one needs the aforementioned termination criterion.

Bang-Jensen and Gutin [3] report that the exchange of vertices is sifting for the feedback arcs set problem. The exchange is also called *2-OPT*. Their statement has been confirmed by Hanauer's experimental studies [14]. The explanation is that it is harder to find a partner for an exchange than looking for a good place.

2.4 Basic Algorithms

Our algorithms are based on sorting and in particular on selection sort, insertion sort and Quicksort. However, care must be taken with implementation details.

ELS and ELS-abs. The algorithm by Eades et al. [10] can be regarded as a *two-sided selection sort*, which looks for the smallest and largest elements and moves them left and right, respectively. It keeps a left and a right list of vertices, which are organized as stacks. At any stage, first sources are removed from the graph and appended to the left list, whereas sinks are removed and added to the right list. Then in [10] a vertex with maximum degree difference $\max_v\{out(v) - in(v)\}$ is chosen among the remaining ones and treated as a source. In a variation that has also been used in [6] the decision depends on the absolute value of the degree difference, $\max_v\{|out(v) - in(v)|\}$, and the vertex is treated either as a sink or a source, depending on whether $in(v)$ exceeds $out(v)$ or vice versa. Here $out(v)$ ($in(v)$) is the number of outgoing (incoming) arcs of v in the actual graph. The computation continues with the diminished graph. Finally, the left and right lists are concatenated. These two algorithms are called *ELS* and *ELS-abs*.

Both algorithms are fast with a running time of $\mathcal{O}(n + m)$ and they show a similar behavior, as can be seen in Section 3. A particular property is an upper bound of $m/2 - n/6$ for the size of the feedback arc set. At first glance this bound looks bad, since it takes almost half the number of arcs. However, the bound is tight for 3-cycles and it is almost the best one can achieve in the worst case, since the size of the feedback arc set is as large as $m/2 - \Theta(n^{3/2})$ [21,22].

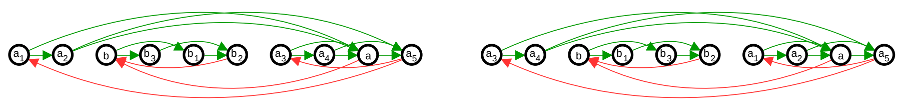
It is readily seen that the *ELS* algorithms are neither SCC-conform nor minimal. If ties are broken in the same way, a second run does not change the computed result. On distinguished graphs, the *ELS* algorithms may perform poorly and are trapped in a local optimum which is $\mathcal{O}(n)$ from the the global optimum, as Coleman and Wirth [6] have discovered.

Quicksort. Our Quicksort variation *KS3* is based on the *KwikSort* algorithm for tournaments by Ailon et al. [1]. On any initial linear arrangement, it moves the items to the left or to the right relatively to a random pivot element, based on whether there is an arc to or from the pivot. The algorithm then proceeds recursively. For tournaments this relation is complete for every pivot, so that there is a clear decision for each vertex. For arbitrary graphs we put the pivot vertex and the undecided vertices with no arc from or to the pivot in the middle and recurse on the left, middle, and right subsets. When ties must be broken, e.g., if the remaining vertices are disconnected, their order is kept.

KS3 is fast with an average running time of $\mathcal{O}(n \log n)$. As Quicksort is a randomized algorithm, it achieves different results in independent runs. This is used for a comparison with the hybrid heuristics. Formal properties cannot be established.

Insertion Sort. The *Sort* operation used by Chanas and Kobylanski [5] for the linear arrangement problem equals sorting by insertion. Given an order of the items, say, $(v_1, \dots, v_n)$, the items are processed in this order. In the i-th round v_i is inserted at the best possible position among the already sorted set of the first $i-1$ items. In case of a tie the leftmost position is taken. In order to find the optimal position the number of backarcs induced by v_i is taken into account. Thus only the arcs between v_i and the first $i-1$ vertices $v_1, \dots, v_{i-1}$ are relevant. This is a conceptional weakness, since some relevant information is omitted.

Sort runs in $\mathcal{O}(n^2)$ time and is monotone. However, it is neither SCC-conform nor minimal, as the example in Fig. 1 shows. The graph depicted obviously has



(a) Initial Arrangement and Second Run

(b) First Run

Fig. 1. Counterexample for SCC-Conformity for *Sort*$^{(*)}$, *Sift*$^{(*)}$, and *Move*$^{(*)}$

two strongly connected components, consisting of the nodes $a_{(i)}$ and $b_{(i)}$, respectively. The left-hand figure shows the initial arrangement and thus the order the vertices are processed in. On the right-hand side, the vertices were sorted once. Running the algorithm another time results in the initial arrangement again, so SCC-conformity can never be reached.

Sifting. *Sifting* can be regarded as a two-sided insertion sort. It has frequently been used in graph drawing for crossing minimization problems [17,2]. Iteratively, each vertex v is removed from the current linear arrangement and is then reinserted at its best possible position, which is determined by the least number of backward arcs induced by v. The *1-OPT* method from [3] is the generic version, which describes the move of v in terms of a neighborhood.

There are several versions and implementations of sifting. Our *Sift* method iterates over the vertices from left to right as they are given by the linear arrangement before the start. In case of a tie for a best position it takes the leftmost one. Additionally, we introduce the *Sift*$_r$ method, which does exactly the same but iterates over the vertices from right to left. In one sifting round both methods consider each vertex exactly once.

In contrast, the *Move* method iterates over the positions from left to right. In the i-th round it picks the vertex which currently is in the i-th place and moves it to its best possible position, again taking the leftmost to break ties. However, *Move* may consider a vertex v more than once if v is reinserted to the right and as a compensation it skips other vertices which are shifted while another vertex is moved. *Move* has first been used in the *Chanas-Both* algorithm of [6]. However, their *MOVES* heuristic is different. For any stage it can remove and reinsert any vertex at any position and it chooses the best such move. This takes $\mathcal{O}(n^2)$ time per step and slows down the algorithm.

All these sifting methods run in $\mathcal{O}(n^2)$ time, and their run time is quite close in practice. They are monotone. However, they are neither SCC-conform nor minimal, as example 1 shows. In addition, Coleman and Wirth [6] have discovered bad examples both for insertion sort and sifting with a gap of $\mathcal{O}(n)$ to an optimal solution.

2.5 Hybrid Algorithms

The goal of hybrid algorithms is to overcome some weak points of monolithic algorithms and to merge the "good properties" of them. The drawback is a higher run time. So there is a trade-off between quality and time.

The *ELS* algorithms are stable in the sense that a second run does not change the result. So repetitions are worthless.

Quicksort is a randomized algorithm and each run may yield a different result. *KS3-200* runs *KS3* 200 times on random input orders and keeps the best result. The value of 200 was chosen after test runs showed that the convergence numbers of the other algorithms are up to 200 for the largest graphs with 1000 vertices and 249750 arcs.

Next we use the basic algorithms from above and run them repeatedly up to their convergence number.

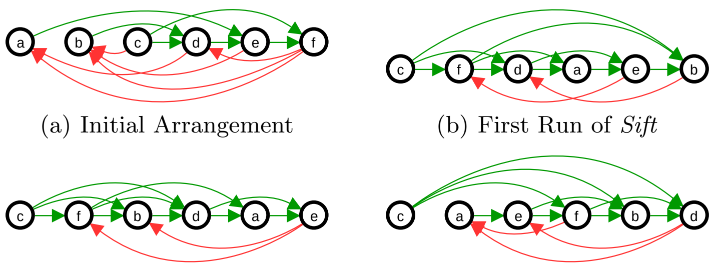
As an auxiliary function, we use the reversal function from [5] for inverting a sequence by $\mathit{Reversal}((x_1, \ldots, x_n)) = (x_n, \ldots, x_1)$. The reversal function puts everything upside down and switches forward and backward arcs. However, in combination with insertion sort it is monotone, as Chanas and Kobylanski [5] have shown.

Lemma 1. *Let $\pi = (v_1, \ldots, v_n)$ be a linear arrangement with cost $c(\pi)$. Let $\pi' = \mathit{Sort}(\mathit{Reversal}(\pi))$. Then $c(\pi) \geq c(\pi')$.*

In contrast, sifting and reversal do not cooperate. The subsequent example and also our experiments (see Fig. 2) show that there are instances of linear arrangements π with $c(\pi) < c(\mathit{Sift}(\mathit{Reversal}(\pi))$. But after a (long) while it finally reaches a stable state.

For a concise description we use the notation A^* to denote the repeated application of an algorithm up to its convergence number. This number is also employed to compare the algorithms.

In the worst case, the repetitions of algorithm A to A^* increase the run time by a factor of $\mathcal{O}(m)$, which results from the bound on the size of the feedback arc set. In practice, the algorithms converge much faster, as Fig. 5 clearly shows.



(a) Initial Arrangement (b) First Run of *Sift*

(c) Second Run of *Sift* (d) Reversal and Third Run of *Sift*

Fig. 2. Counterexample for Non-monotonicity of *Sift* and *Reversal*

We tested the following hybrid heuristics:

$$\begin{aligned}
\textit{It-Sort} &= \textit{Sort}^* \\
\textit{It-Sift} &= \textit{Sift}^* \\
\textit{It-Move} &= \textit{Move}^*, \text{ which is called } \textit{Chanas-Both} \text{ in [6]} \\
\textit{CK-Sort} &= \textit{Sort}^* \circ (\textit{Reversal} \circ \textit{Sort}^*)^* \\
\textit{CK-Sift} &= \textit{Sift}^* \circ (\textit{Reversal} \circ \textit{Sift}^*)^* \\
\textit{It-2-Sift} &= (\textit{Sift}^* \circ (\textit{Sift}_r \circ \textit{Sift}^*)^* \\
\textit{X-Sift} &= \textit{Sift}^* \circ ((\textit{Reversal} \circ \textit{Sort}) \circ \textit{Sift}^*)^*.
\end{aligned}$$

Thus *X-Sift* first applies sifting until the convergence number is reached. Then it reverses the arrangement and runs sort once followed by sifting, until sifting reaches it convergence number. The latter is repeated up to the convergence number of the combined methods.

As formal properties we obtain from the aforesaid:

Theorem 1. *It-Sort, It-Sift, It-2-Sift, It-Move, CK-Sort, CK-Sift and X-Sift are monotone.*

However, we have observed that these algorithms are not *1-OPT*, although they use sifting repeatedly.

3 Test Suites and Measurements/Data/Diagrams

We have tested our algorithms on several test suites, but only report on two of them here. The first consists of random graphs according to the Erdős and Rényi model [11] $G(n, p)$. The size n ranges from 100 to 1000 in steps of 100 and an edge between any two vertices is chosen with probability p. Here $p = 0.5$, so the graphs have about 249750 directed edges. Sparser graphs with $p = 0.1$ and other sets of graphs were tested in [14] with a similar outcome.

Figures 3 and 4 show the average values for performance and run time which were computed from 1000 random graphs for each configuration. The average size of the feedback arc sets of the graphs with 1000 vertices is additionally listed in Table 1. Furthermore, we accomplished an in-depth analysis of the algorithms' progress until they converge. Figure 5 graphs the development again taking the mean over 1000 repetitions. The curves stop at the algorithms' maximum convergence number. Table 2 provides information on the average value.

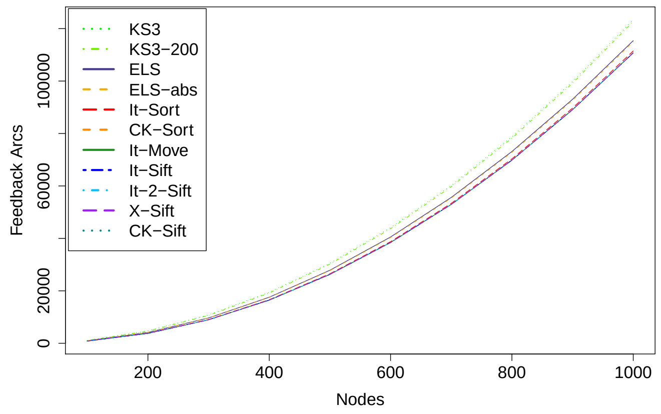


Fig. 3. Average Algorithm Performances

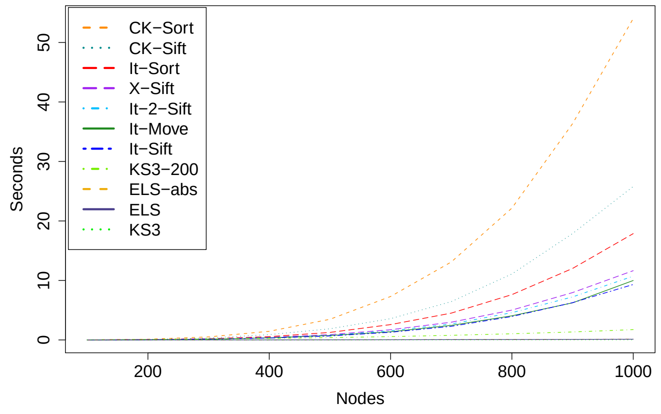


Fig. 4. Average Running Times

Our second test suite consists of the counterexamples by Coleman and Wirth [6]. These counterexamples are designed such that insertion sort and sifting are trapped in a local optimum which is $\mathcal{O}(n)$ from the global optimum provided that the graphs start with an unfavorable linear arrangement. The graphs consist

Table 1. Average Algorithm Performances on Graphs with 1000 Nodes

Algorithm	*It-Sort*	*It-Sift*	*It-Move*	*CK-Sort*	*CK-Sift*	*It-2-Sift*	*X-Sift*
Feedback Arcs	111472.2	110736.9	110749.0	110814.2	110661.1	110730.1	110729.8

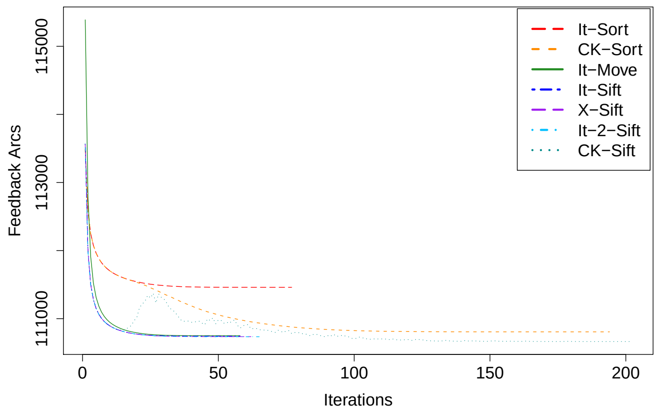


Fig. 5. Average Convergence

of two sets of $n/2$ vertices (black and white), each of which is internally transitive. There is a one-to-one correspondence between white and black vertices represented by an arc (u, v), where u is white and v is black. For all other pairs of white and black vertices there is an arc from black to white. Hence, these graphs are tournaments. The global optimum is achieved with all black vertices to the left of all white ones, resulting in a feedback arc set of size $n/2$. If the vertices are arranged as a sequence of pairs of alternating white and black vertices, then neither insertion sort nor sifting shall move a vertex, since there is no proper improvement by any move. However, this particular order is very unlikely and, starting from a random arrangement, the hybrid algorithms most of the time found the optimal solution for these graphs over 1000 runs per configuration.

Table 3 shows the optimal number of feedback arcs as well as the average amount found by each algorithm.

4 Evaluation of the Results and Conclusion

All experiments were conducted on a AMD Phenom II X6 1090T machine with 8GB of memory. The algorithms were developed and executed in a Java$^{\text{TM}}$ SE Runtime Environment in version 1.6 using the Java$^{\text{TM}}$ HotSpot$^{\text{TM}}$ Server VM.

Table 2. Average Convergence Numbers

Algorithm	*It-Sort*	*It-Sift*	*It-Move*	*CK-Sort*	*CK-Sift*	*It-2-Sift*	*X-Sift*
Iterations	31.00	24.34	26.89	99.51	74.47	27.83	27.87

Table 3. Average Performance on Counterexample Graphs

Sizes	10	50	100
OPT	5	25	50
ELS	6.08	27.79	53.52
ELS-abs	6.08	27.79	53.52
KS3	9.40	69.10	143.67
KS3-200	8.00	48.00	98.00
It-Sort	5.42	25.04	50.03
It-Sift	5.12	25.00	50.00
It-Move	5.11	25.00	50.00
CK-Sort	5.42	25.04	50.03
CK-Sift	5.00	25.00	50.00
It-2-Sift	5.12	25.00	50.00
X-Sift	5.12	25.00	50.00

The algorithms have been tested on several data sets, however, only two are reported here.

The main experience is the similarity of the performance of the hybrid algorithms. The size of the computed feedback arc sets differ by about 0.1%. This may be an indication that it is close to the optimum, which, however, remains unknown. Nevertheless, sifting clearly outperforms sorting.

Concerning their *convergence number* the hybrid algorithms behave quite differently, as Fig. 5 illustrates. This has a direct impact on the run time, see Fig. 4. Here, sifting is faster than insertion sort and pure sorting and sifting heuristics are in turn faster than the CK algorithms.

Overall, our results compare with those in [6,20]. All studies show competitiveness of the designed heuristics, but some details and conclusions differ. We took a much closer examination, particularly on the convergence. The significance of our study is based on much larger graphs, which are not tournaments.

In detail we conclude a slight superiority of sifting over sorting.

5 Conclusion

Our study has shown several competitive heuristics for the feedback arc set or the linear arrangement problem. These should be put into a framework with a pre- and a postprocessing.

Still, a major challenge is the estimation between these heuristics and the optimal solution. There are no good lower bounds for the feedback arc set problem, even on tournaments.

References

1. Ailon, N., Charikar, M., Newman, A.: Aggregating inconsistent information: Ranking and Clustering. Journal of the ACM 55(5), Article 23 (Oct 2008)

2. Bachmaier, C., Brandenburg, F.J., Brunner, W., Hübner, F.: A global k-level crossing reduction algorithm. In: WALCOM. LNCS, vol. 5942, pp. 70–81 (2010)
3. Bang-Jensen, J., Gutin, G.: Digraphs: Theory, Algorithms and Applications. Springer-Verlag, London, 2. edn. (2006)
4. Berger, B., Shor, P.W.: Approximation algorithms for the maximum acyclic subgraph problem. In: Proc. First ACM-SIAM Symposium on Discrete Algorithms. pp. 236–243 (1990)
5. Chanas, S., Kobylánski, P.: A new heuristic algorithm solving the linear ordering problem. Computational Optimization and Applications 6(2), 191–205 (1996)
6. Coleman, T., Wirth, A.: Ranking tournaments: Local search and a new algorithm. Journal of Experimental Algorithmics (JEA) 14, 2.6–2.22 (2009)
7. Demetrescu, C., Finocchi, I.: Combinatorial Algorithms for Feedback Problems in Directed graphs. Information Processing Letters 86(3), 129–136 (2003)
8. Dinur, I., Safra, S.: On the hardness of approximating minimum vertex cover. Annals of Mathematics 162, 439–485 (2005)
9. Eades, P., Lin, X.: A Heuristic for the Feedback Arc Set Problem. Australasian Journal of Combinatorics 12, 15–25 (1995)
10. Eades, P., Lin, X., Smyth, W.F.: A fast and effective heuristic for the Feedback Arc Set problem. Information Processing Letters 47(6), 319–323 (Oct 1993)
11. Erdős, P., Rényi, A.: On random graphs, i. Publicationes Mathematicae (Debrecen) 6, 290–297 (1959)
12. Even, G., Naor, J.S., Schieber, B., Sudan, M.: Approximating Minimum Feedback Sets and Multi-Cuts in Directed Graphs. In: Proceedings of the 4th International IPCO Conference on Integer Programming and Combinatorial Optimization. pp. 14–28. Springer-Verlag, London, UK (1995)
13. Günther, W., Schönfeld, R., Becker, B., Molitor, P.: k-layer straightline crossing minimization by speeding up sifting. In: Graph Drawing. LNCS, vol. 1984, pp. 253–258. Springer-Verlag, London, UK (2001)
14. Hanauer, K.: Algorithms for the Feedback Arc Set Problem. Master's thesis, Department of Informatics and Mathematics, University of Passau, Passau, Germany (2010), http://www.infosun.fim.uni-passau.de/~hanauer/pub/mastersthesis.pdf
15. Karp, R.M.: Reducibility Among Combinatorial Problems. Complexity of Computer Computations pp. 85–103 (1972)
16. Kenyon-Mathieu, C., Schudy, W.: How to rank with few errors: A PTAS for Weighted Feedback Arc Set on Tournaments. Electronic Colloquium on Computational Complexity (ECCC) 13(144) (2006)
17. Matuszewski, C., Schönfeld, R., Molitor, P.: Using sifting for k -layer straightline crossing minimization. In: Graph Drawing. pp. 217–224 (1999)
18. Ramachandran, V.: A minimax arc theorem for reducible flow graphs. SIAM Journal on Discrete Mathematics 3(4), 554–560 (1990)
19. Saab, Y.: A Fast and Effective Algorithm for the Feedback Arc Set. Journal of Heuristics 7, 235–250 (2001)
20. Schalekamp, F., van Zuylen, A.: Rank aggregation: Together we're strong. In: ALENEX. pp. 38–51 (2009)
21. Spencer, J.: Optimal ranking of tournaments. Networks 1, 135–138 (1971)
22. Spencer, J.: Optimally ranking unrankable tournaments . Periodica Mathematica Hungarica 11(2), 131–144 (Jun 1980)
23. Sugiyama, K., Tagawa, S., Toda, M.: Methods for Visual Understanding of Hierarchical System Structures. IEEE Transactions on Systems, Man and Cybernetics 11(2) (Feb 1981)