Comments on "An Exact Method for the Minimum Feedback Arc Set Problem"

MARTIN GRÖTSCHEL, Institute of Mathematics, Technische Universität Berlin
MICHAEL JÜNGER, Department of Mathematics and Computer Science, Universität zu Köln
GERHARD REINELT, Department of Computer Science, Universität Heidelberg

We comment on the ACM Journal of Experimental Algorithmics article "An Exact Method for the Minimum Feedback Arc Set Problem" by Ali Baharev, Hermann Schichl, Arnold Neumaier, and Tobias Achterberg, and we point out that a straightforward implementation of an algorithm we published in 1985, with the same modern technology as used in the article we address, is competitive and mostly superior.

CCS Concepts: • **Mathematics of computing** → *Mathematical software*; *Mathematical software performance*; *Discrete mathematics*; *Combinatorics*; *Combinatorial optimization;*

Additional Key Words and Phrases: Minimum feedback arc set problem, acyclic subdigraph problem, exact methods for combinatorial optimization, experimental evaluation, branch and cut, integer programming

ACM Reference format:
Martin Grötschel, Michael Jünger, and Gerhard Reinelt. 2022. Comments on "An Exact Method for the Minimum Feedback Arc Set Problem". *J. Exp. Algorithmics* 27, 1, Article 1.3 (July 2022), 4 pages.
https://doi.org/10.1145/3545001

In [2], the authors propose to solve the feedback arc set problem to optimality with a lazy constraint version of the minimum set cover formulation of [7], and they give evidence that this approach, called "**(PM)**" for "**Proposed Method**", outperforms the "**Triangle Inequalities**" approach called "**(TI)**".

The authors concentrate on sparse unweighted directed graphs, and perform computational experiments with their PM-approach in comparison to the TI-approach. The latter consists of solving a big binary linear program with Gurobi [6]: Given a sparse weighted directed graph $D = (V, A)$, this program has $\binom{|V|}{2}$ variables and $2\binom{|V|}{3}$ nontrivial constraints. This formulation is trivially correct and has been one of the starting points of a series of articles [3–5] we published in the mid-1980s in the pursuit of finding better methods to deal both with the dense case (the linear ordering problem for complete weighted digraphs) and with the sparse case (the acyclic subdigraph problem for arbitrary weighted digraphs), laying the foundations of branch&cut approaches for the

Authors' addresses: M. Grötschel, Institute of Mathematics Technische Universität Berlin Straße des 17. Juni 136 D-10587 Berlin Germany; email: groetschel@bbaw.de; M. Jünger, Department of Mathematics and Computer Science Universität zu Köln, Albertus-Magnus-Platz, D-50923 Köln, Germany; email: juenger-sfb@cs.uni-koeln.de; G. Reinelt, Department of Computer Science, Universität Heidelberg, Im Neuenheimer Feld 205, D-69120 Heidelberg, Germany; email: ip121@uni-heidelberg.de.

1084-6654/2022/07-ART1.3 $15.00
https://doi.org/10.1145/3545001

respective problems. (The term "branch&cut" applies here but was coined only later.) In particular, the discussion in Section 5 of [5] (reference [45] in [2]) essentially says that in a cutting plane approach, violated directed cycles can be separated in polynomial time by using methods like the Dijkstra or the Floyd–Warshall method. Of course, a fair comparison would necessarily have been with this approach that is suited for sparse instances like their new PM-approach.

In their computational study in [2], the authors remark that "In all other cases, the proposed method was consistently and significantly faster than the method of Section 3.1." (Section 3.1 in [2] contains the TI binary linear program; "all other cases" refers to all instances except `Imase_Itoh_n_120_d_4` that could not be solved to optimality within 3 CPU hours.) From our point of view, this computational study just demonstrates that the TI-approach is not at all a suitable one, which was well known before.

It is a good practice in the area of Experimental Algorithmics to make fair comparisons of new approaches with the previous state-of-the-art. Even though two of the articles [3–5] are cited in [2], the authors refrain from comparing with the state-of-the-art of the mid-1980s, arguing that this would be beyond the scope of their article. On the other hand, they discuss "Lazy Constraints Compared to Cutting Planes" in Section 4.4 of [2]. A logical consequence would have been to compare with such algorithms rather than the obviously inferior TI-approach.

For a directed graph $D = (V, A)$ with weights c_a on the arcs in A, the binary linear program

$$\begin{aligned} \text{maximize} \quad & \sum_{a \in A} c_a x_a, \\ & \sum_{a \in C} x_a \quad \leq |C| - 1 \quad \text{for all directed cycles } C \text{ in } D, \qquad (1) \\ & x_a \quad \in \{0, 1\} \quad \text{for all } a \in A, \qquad (2) \end{aligned}$$

models the weighted version of the *acyclic subdigraph problem* on D that is equivalent to the weighted version of the *feedback arc set problem* on D. The constraints (1) are called *dicycle inequalities*. An optimum solution x^* defines a maximum weight acyclic subdigraph (V, S) for $S = \{a \in A \mid x^*_a = 1\}$. Then, $F = A \setminus S$ is a minimum weight feedback arc set in D.

Starting with a trivial relaxation with no dicycle inequalities (1) and the integrality constraints (2) replaced by the trivial inequalities $0 \leq x_a \leq 1$ for all $a \in A$, we realize a cutting-plane algorithm in which a sequence of linear programs that represent increasingly stronger relaxations are solved. Given the solution of some relaxation, we solve the *separation problem,* which consists of finding at least one dicycle inequality that is violated by the solution or proving that no such inequality exists. In the former case, the violated inequalities are added to the relaxation, and the stronger relaxation is solved; in the latter case, the cutting plane phase ends. If all x_a are integral, the acyclic subdigraph problem is solved; otherwise, the integrality constraints (2) are re-installed and the problem is solved in a *branch&bound* procedure in which the bounds are strengthened using dicycle separation at nodes of the branch&bound tree. Such an approach is commonly called *branch&cut*.

We implemented a branch&cut algorithm based on nothing but the ingredients we published in 1985, using the same modern technology as the authors (that was not available in the mid-1980s), but we used C instead of Python.

This involved the implementation of a directed (general, not 3-) cycle separator applying the Dijkstra method (which can better exploit sparsity than the Floyd–Warshall method) within a cutting-plane algorithm using Gurobi as the LP-solver. If the optimum is fractional, the problem is turned into a binary problem with the same separator in a lazy fashion at every 10th node of the Gurobi branch&bound tree.

Table 1. Computational Results on de Bruijn Graphs

Nodes	Arcs	Parameter d	Optimum	$t_{PM}(s)$	$m_{PM}(MB)$	$t_{CI}(s)$	$m_{CI}(MB)$
100	296	3	58	9.85	63.46	4.96	33.46
100	396	4	91	2.10	39.11	1.00	9.58
100	492	5	116	1.86	39.83	0.76	14.96
100	590	6	158	31.23	120.30	9.68	41.46
110	326	3	63	1.94	37.60	0.76	15.34
110	436	4	97	50.69	100.81	92.99	98.95
110	544	5	134	39.06	102.01	1.65	13.05
110	650	6	172	1,214.80	161.73	2,317.63	176.44
120	356	3	66	2.47	36.97	0.53	8.24
120	474	4	108	5.14	41.80	2.21	26.92
120	592	5	150	2.74	37.07	1.50	13.84
120	710	6	180	340.19	161.79	88.99	104.53

Table 2. Computational Results on Graphs of Imase and Itoh

Nodes	Arcs	Parameter d	Optimum	$t_{PM}(s)$	$m_{PM}(MB)$	$t_{CI}(s)$	$m_{CI}(MB)$
100	300	3	66	1.02	30.64	0.08	3.69
100	400	4	90	1.82	38.27	0.51	5.50
100	496	5	126	2.11	39.72	0.78	5.54
100	594	6	156	51.74	143.24	6.73	33.86
100	696	7	192	19.46	53.41	7.10	33.18
110	328	3	62	1.90	37.61	0.56	8.64
110	440	4	100	2.09	36.88	1.02	12.68
110	546	5	135	9.70	51.82	2.61	7.27
110	654	6	172	78.71	135.46	11.18	66.17
110	764	7	210	3,437.95	200.35	358.32	155.77
120	360	3	72	2.15	38.19	0.23	4.77
120	480	4	114	2,488.63	227.50	3,626.30	169.73

With this implementation, henceforth referred to as "**(CI)**" for "**Cycle Inequalities**", we could verify all results in Table 1 of [2] with negligible computation times.

The computational experiments underlying Tables 2 and 3 of [2] have been executed on an Intel Core i5-3320M CPU at 2.60 GHz, Operating System: Ubuntu 16.04.2 LTS with 4.15.0-43-generic Kernel; Gurobi 8.1.0, Python 3.6.6; NetworkX 1.11. In the pursuit of a fair comparison of the PM method with our CI method, we use the code for PM retrieved from [1] (reference [8] in [2]), on the same computer on which we implemented the CI method: an Apple MacBook Pro with a 2.9 GHz Quad-Core Intel Core i7, Operating System: macOS 11.4, Gurobi 8.1.0, gcc 11.0.3 for the CI implementation.

Using the Unix "`time -l`" command, we measured computation times in seconds ("`real time`") and the process peak memory consumption ("`peak memory footprint`"). The results are in Table 1 for the de Bruijn instances and in Table 2 for the Imase and Itoh instances of the corresponding Tables 2 and 3 of [2]. In both tables, TI is replaced by CI. The column headers are $t_{PM}(s)$ and $t_{CI}(s)$ for the execution times in seconds of PM and CI, respectively, as well as $m_{PM}(MB)$ and $m_{CI}(MB)$ for the peak memory consumption in megabytes of PM and CI, respectively.

A first observation is that the computational platform does matter: The running times $t_{PM}(s)$ are sometimes significantly better than those reported in [2].

The main observation is that CI outperforms PM in terms of computation time in all but three cases: `de_Bruijn_n_110_d_4`, `de_Bruijn_n_110_d_6`, and `Imase_Itoh_n_120_d_4`. In terms of peak memory consumption, CI outperforms PM in all but one case: `de_Bruijn_n_110_d_6`.

Our computational experiment indicates that the CI approach whose basis we published more than 35 years ago can compete with a new development of 2021. But, it also demonstrates the potential of the new PM approach that outperforms our classical branch&cut method CI for certain instances.

We have shared these comments with the authors of [2] before submission for publication, and we are glad to have their support.

We join the authors of [2] in the hope that their work in combination with our comments will trigger research targeting at new solution strategies for the feedback arc set problem and other combinatorial optimization problems.

REFERENCES

[1] Ali Baharev. Exact and heuristic methods for tearing. Retrieved June 28, 2021 from https://github.com/baharev/sdopt-tearing.

[2] Ali Baharev, Hermann Schichl, Arnold Neumaier, and Tobias Achterberg. 2021. An exact method for the minimum feedback arc set problem. *ACM Journal of Experimental Algorithmics* 26, Article 1.4 (April 2021), 28 pages. DOI: https://doi.org/10.1145/3446429

[3] Martin Grötschel, Michael Jünger, and Gerhard Reinelt. 1984. A cutting plane algorithm for the linear ordering problem. *Operations Research* 32, 6 (Dec. 1984), 1195–1220. DOI: https://doi.org/10.1287/opre.32.6.1195

[4] Martin Grötschel, Michael Jünger, and Gerhard Reinelt. 1985. Facets of the linear ordering polytope. *Mathematical Programming* 33, 1 (1985), 43–60. DOI: https://doi.org/10.1007/BF01582010

[5] Martin Grötschel, Michael Jünger, and Gerhard Reinelt. 1985. On the acyclic subgraph polytope. *Mathematical Programming* 33, 1 (1985), 28–42. DOI: https://doi.org/10.1007/BF01582009

[6] Gurobi Optimization. 2021. Gurobi Optimizer. Retrieved from http://www.gurobi.com.

[7] T. K. Pho and L. Lapidus. 1973. Topics in computer-aided design: Part I. An optimum tearing algorithm for recycle systems. *AIChE Journal* 19, 6 (1973), 1170–1181. DOI: https://doi.org/10.1002/aic.690190614

Received August 2021; revised October 2021; accepted October 2021